Seamless Interoperability and Data Portability in the Social Web for Facilitating an Open and Heterogeneous Online Social Network Federation

vorgelegt von
Dipl.-Inform.
Sebastian Jürg Göndör
geb. in Duisburg

von der Fakultät IV – Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
- Dr.-Ing. -

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Thomas Magedanz
Gutachter:	Prof. Dr. Axel Küpper
Gutachter:	Prof. Dr. Ulrik Schroeder
Gutachter:	Prof. Dr. Maurizio Marchese

Tag der wissenschaftlichen Aussprache: 6. Juni 2018

Berlin 2018

A Bill of Rights for Users of the Social Web

Authored by Joseph Smarr, Marc Canter, Robert Scoble, and Michael Arrington[1]
September 4, 2007

Preamble:

There are already many who support the ideas laid out in this Bill of Rights, but we are actively seeking to grow the roster of those publicly backing the principles and approaches it outlines. That said, this Bill of Rights is not a document "carved in stone" (or written on paper). It is a blog post, and it is intended to spur conversation and debate, which will naturally lead to tweaks of the language. So, let's get the dialogue going and get as many of the major stakeholders on board as we can!

A Bill of Rights for Users of the Social Web

We publicly assert that all users of the social web are entitled to certain fundamental rights, specifically:
Ownership of their own personal information, including:

- their own profile data
- the list of people they are connected to
- the activity stream of content they create;
- Control of whether and how such personal information is shared with others; and
- Freedom to grant persistent access to their personal information to trusted external sites.

Sites supporting these rights shall:

- Allow their users to syndicate their own profile data, their friends list, and the data that's shared with them via the service, using a persistent URL or API token and open data formats;
- Allow their users to syndicate their own stream of activity outside the site;
- Allow their users to link from their profile pages to external identifiers in a public way; and
- Allow their users to discover who else they know is also on their site, using the same external identifiers made available for lookup within the service.

[1]The text *A Bill of Rights for Users of the Social Web* has been published on September 4, 2007 by Joseph Smarr et al. at `http://www.opensocialweb.org/2007/09/05/bill-of-rights` [1] and has been reprinted with permission.

Abstract

Online Social Networks (OSN) have become an integral part of our everyday lives. We express ourselves, create and collect content such as images or videos, share content and information with our friends and colleagues, exchange messages, or keep track of what's happening in the world. Yet, despite social communication being implicitly a distributed, decentralized phenomenon, most OSN services are built in a central, monolithic fashion, concentrating all knowledge and power in one company or organization. This contradicts the idea of the social web, as proprietary and isolated walled gardens keep users from being able to freely choose an OSN platform provider or to effectively control their privacy. In order to mitigate the problem, alternative architectures that distribute control and data to multiple independent services were proposed. Unfortunately, the implicit network effects existing in large OSN services still prevent users from migrating to alternative solutions in significant numbers. Moreover, technical protocols for facilitating holistic and seamless interoperability and furthermore data portability in OSN services do not exist. Ultimately, today's OSN market is dominated by one single service which has been able to attract a significant amount of users, while a large number of competing services and alternative solutions exist that combine a comparably small number of users.

Two main issues have been identified that contribute to the current situation of one OSN service heavily dominating the entire market, being the lack of data portability and interoperability between different OSN services.

This work proposes *Sonic*, a solution that aims to interconnect arbitrary OSN services into one open and heterogeneous federation of OSN services. Sonic introduces an open communication protocol and data formats that are able to facilitate interconnectivity of OSN services. The proposed architecture supports core features implemented in today's most popular OSN services and facilitates extended functionality through an extensibility framework.

Zusammenfassung

Online Social Networks (OSN) sind zu einem integralen Bestandteil unseres täglichen Lebens geworden. Wir artikulieren unsere Meinung, erstellen und sammeln Bilder und Videos, teilen Inhalte und Informationen mit unseren Freunden und Kollegen, kommunizieren und halten uns auf dem Laufenden über Geschehnisse in der Welt. Doch obwohl soziale Kommunikation inhärent ein verteiltes, dezentrales Phänomen ist, folgen die meisten OSN Dienste in einer zentralistischen und monolithischen Konzeption, welche alles Wissen und alle Macht einer einzigen Organisation überlässt. Dies läuft der Idee des sozialen Webs zuwider, da proprietäre und isolierte Plattformen Nutzer daran hindern, selbstbestimmt eine Netzwerkplattform zu wählen oder die Kontrolle über ihre Daten zu behalten. Um das Problem zu beheben, wurden alternative Architekturen entworfen, durch welche Kontrolle und Daten auf mehrere unabhängige Dienstplattformen verteilt wird. Allerdings hindern die impliziten Netzwerkeffekte großer Netzwerke Nutzer daran, in signifikanter Anzahl zu alternativen Angeboten zu migrieren. Darüber hinaus existieren keine Protokolle, welche ganzheitliche und nahtlose Interoperabilität und darüber hinaus Datenportabilität in OSN Diensten ermöglichen. Im Resultat wird der heutige OSN Markt von einem einzigen Dienst dominiert, welcher in der Lage war, eine signifikante Anzahl an Nutzern zu gewinnen, während eine große Anzahl an konkurrierenden Diensten und alternativen Angeboten existiert, auf welche sich eine verhältnismäßig kleine Zahl an Nutzern vereint.

Zwei Probleme konnten hierbei als Hauptursache für die aktuelle Situation, in welcher ein OSN Dienst den gesamten Markt stark dominiert, identifiziert werden. Hierbei handelt es sich um die fehlende Unterstützung von Datenportabilität und Interoperabilität zwischen verschiedenen OSN Diensten.

Das vorliegende Werk stellt *Sonic* vor, eine Lösung zur Anbindung beliebiger OSN Dienste untereinander in einer heterogenen Föderation von OSN Diensten. Sonic spezifiziert ein offenes Kommunikationsprotokoll sowohl als auch Datenformate, durch welche Interkonnektivität von OSN Diensten ermöglicht wird. Die vorgestellte Architektur unterstützt die Kernfunktionalitäten heutiger OSN Dienste und ermöglicht darüber hinaus eine Unterstützung erweiterter Funktionalitäten über ein Erweiterungsframework.

Acknowledgements

The work described in this thesis wouldn't have been possible without the help and support of several people. I am deeply thankful for the support I received and would like to mention those who helped and supported me.

First of all, my most sincere thank goes to my supervisor Prof. Dr. Axel Küpper for giving me the opportunity to work on my research as part of the research group *Service-centric Networking*. He not only provided a professional and inspiring working environment, but also gave me the opportunity to learn and grow. In the years I worked on my thesis, he always supported my research and provided invaluable feedback and motivation. Furthermore, I would like to express my sincere gratitude to Prof. Dr. Ulrik Schroeder and Prof. Dr. Maurizio Marchese for supervising this thesis and providing invaluable feedback and support.

I also would like to thank my colleagues at the research group for inspiring discussions and valuable feedback and support. I owe special gratitude to Dirk Thatmann, who managed to keep work off my back in the Cyclone project in the last phase of writing my dissertation.

Work on this thesis essentially was started in 2014 with the research project *Sonic*, which was funded by the Software Campus initiative. As part of this project, I worked with and received support from several people. Most importantly I would like to thank my mentor, Riccardo Pascotto, who always managed to have time to discuss the various issues I faced and provided valuable advice and support. Furthermore, I would like to thank the researchers and student workers who worked with me in the Sonic project, specifically Senan Sharhan, Felix Beierle, Hussam Hebbo, Evren Küçükbayraktar, and Markus Beckmann. Special thanks go out to the entire Software Campus team for their wonderful and professional support, specifically to Erik Neumann, Maren Lesche, Susanne Kegler, and Kerstin Potemka.

Proofreading a scientific dissertation is a cumbersome and time-consuming, yet utterly important task. I would hence like to express sincere and special gratitude to Jan Benzenberg, Maria João Ruiz, and Steffen Bretzke for spending countless hours proofreading the thesis, pointing out mistakes I made, and providing invaluable feedback and criticism.

Finally, I would like to thank my parents, Jürgen Göndör and Elisabeth Smetka-Göndör, as well as Maria João Ruiz for their invaluable support and motivation over the years I worked on this thesis. Thank you for believing in me!

Publications

Some of the ideas, methods, and results that significantly contributed to this thesis have been published in scientific publications and presented at international workshops and conferences. To a certain extent, parts of this thesis are therefore contained in the following publications:

- Göndör, S. and Devendraraj, J. (2013). C2M: Open and Decentralized Cloud Contact Management. Procedia Computer Science, International Conference on Computational Science (ICCS) 2013. Elsevier. [2].
- Göndör, S. and Hebbo, H. (2014). SONIC: Towards Seamless Interaction in Heterogeneous Distributed OSN Ecosystems. 1st Workshop on Dynamic Social Networks (DSoNets) 2014. IEEE. [3].
- Göndör, S. and Beierle, F. and Küçükbayraktar, E. and Hebbo, H. and Sharhan, S. and Küpper, A. (2015). Towards Migration of User Profiles in the SONIC Online Social Network Federation. 10th International Multi-Conference on Computing in the Global Information Technology (ICCGI) 2015. IARIA. [4].
- Göndör, S. and Beierle, F. and Sharhan, S. and Hebbo, H. and Küçükbayraktar, E. and Küpper, A. (2015). SONIC: Bridging the Gap between Different Online Social Network Platforms. 8th International Conference on Social Computing and Networking (SocialCom) 2015. IEEE. [5].
- Beierle, F. and Grunert, K. and Göndör, S. and Schlüter, V. (2017). Towards Psychometrics-based Friend Recommendations in Social Networking Services. 1st International Conference on AI & Mobile Services (AIMS) 2017. IEEE. [6].
- Göndör, S. and Beierle, F. and Sharhan, S. and Küpper, A. (2016). Distributed and Domain-Independent Identity Management for User Profiles in the SONIC Online Social Network Federation. 5th International Conference on Computational Social Networks (CSoNet) 2016. Springer. [7].
- Javed, I. and Copeland, R. and Crespi, N. and Beierle, F. and Göndör, S. and Küpper, A. and Bouabdallah, A. and Emmelmann, M. and Ancuta, A. and Corre, K. and Crom, J.-M. and Oberle, F. and Friese, I. and Caldeira, A. and Dias, G. and Chaves, R. and Santos, N. (2016). Global Identity and Reachability Framework for Interoperable P2P Communication Services. 19th conference on Innovations in Clouds, Internet and Networks (ICIN) 2016. [8].

- Beierle, F. and Göndör, S. and Küpper, A. (2015). Towards a Three-tiered Social Graph in Decentralized Online Social Networks. 7th International Workshop on Hot Topics in Planet-scale Mobile Computing and Online Social Networking (HOTPOST) 2015. ACM. [9].
- Friese, I. and Copeland, R. and Göndör, S. and Beierle, F. and Küpper, A. and Pereira, R. and Crom, J.-M. (2017). Cross-Domain Discovery of Communication Peers - Identity Mapping and Discovery Services (IMaDS). European Conference on Networks and Communications (EUCNC) 2017. IEEE. [10].
- Javed, I. and Copeland, R. and Crespi, N. and Emmelmann, M. and Corici, A. and Bouabdallah, A. and Zhang, T. and El Jaouhari, S. and Beierle, F. and Göndör, S. and Küpper, A. and Corre, K. and, Crom, J.-M. and Oberle, F. and Friese, I. and Caldeira, A. and Dias, G. and Santos, N. and Chaves, R. and Pereira, R. (2017). Cross Domain Identity and Discovery Framework for Peer-to-Peer Calling Services. Annals of Telecommunications. 2017. ISSN: 1958-9395. Springer. [11].
- Göndör, S. (2017). The Importance of Data Portability and Interoperability in the Social Web. In: Practical Implementation of the Right to Data Portability - Legal, Technical and Consumer-Related Implications. Ed. by N. Horn and A. Riechert. ISBN: 978-3-00-058336-0. Stiftung Datenschutz, Nov. 2017. [12].
- Göndör, S. and Küpper, A. (2017). The Current State of Interoperability in Decentralized Online Social Networking Services. 4th International Conference on Computational Science and Computational Intelligence (CSCI) 2017. IEEE. [13].

Contents

1 Introduction

> *"The web is more a social creation than a technical one. I designed it for a social effect — to help people work together — and not as a technical toy. The ultimate goal of the Web is to support and improve our weblike existence in the world. We clump into families, associations, and companies. We develop trust across the miles and distrust around the corner."*
>
> – Tim Berners-Lee, *Weaving The Web, 1999*

Ever since the advent of the web in the 1990s, people welcomed this new technology as a basis for novel ways of information retrieval, collaboration, and communication [14]. Being mostly a place for computer scientists and tech-savvy users in the early days, it was quickly adopted by the broader public and managed to conquer more and more aspects of our world. As of today, the Internet has become an integral part of our everyday lives, being completely interwoven with everything we do. It has become a commodity, something most people would not want to - or cannot - miss in their daily routines [15]. We read the news, communicate with our relatives and colleagues, seek for entertainment and distraction, work and collaborate, buy and sell goods, watch movies and listen to music, research and share information, express opinions, and manage our bank accounts using a set of protocols and standards that allow countless servers and client devices around the globe to seamlessly interact with each other. And we are able to access this technology from mostly any device, at any time we wish, anywhere [16]. The *Global Village*, as envisioned by Marshall McLuhan in 1962 [17], has become a reality.

What has become normal to be available anytime, anywhere [18] once was cumbersome and complicated to most people. In its early stages, the Web presented itself in a much more basic fashion. Access technology was scarce, expensive, and slow [15], while early web pages were mostly static HTML files, occasionally formatted using CSS, and interlinked via URLs[1]. In these days, the Web could be consumed but not interacted with. Bulletin Boards (BBS), Usenet Newsgroups, and Internet Relay Chat (IRC) offered ways of communication between users, but were rather limited in usability and complicated to use [19]. Despite these limitations, the groundbreaking idea behind the underlying technology of the Web was able to attract millions of users, building a global network of interlinked documents and information [20].

Yet, when Tim Berners-Lee invented the World Wide Web (WWW) at CERN [21], he thought of it as a social communication and collaboration medium, not a technical tool for the mere transfer and retrieval of data [22]. This vision

[1] Archived version of the first webpage on the WWW by Tim Berners-Lee: `https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html`. Accessed: 15.7.2017

came closer to manifestation with the Web 2.0 in the early 2000s. Web 2.0 technology presented a paradigm shift towards bidirectional communication and collaboration, allowing web browsers to communicate with websites on behalf of users [23]. This allowed people to express opinions, create content, and collaborate in completely different ways and set the stage for the social web.

1.1 The Social Web

> *"The static web is an artifact of the past, having been replaced by the idea that sites or applications should, as a standard practice, provide their users with an experience customized to their preferences. The Internet has quickly become a vast community of people who find relevance in their online social experiences and interactions."*
>
> – Jonathan LeBlanc, *Programming Social Applications, 2011*

In the late 1990s, a new kind of web service emerged that allowed people to connect to each other on the web. According to Boyd and Ellison, SixDegrees.com was the first Online Social Network (OSN) or Social Network Site (SNS) to be launched in 1997 [24]. Sixdegrees.com allowed users to create a user profile to represent themselves and specify a list of other users as *friends*, where this list could then be traversed by other users who could view the social profiles of one's friends. This allowed users to visualize their circle of friends and acquaintances and create a digital representation of themselves for others to see. While Boyd and Ellison acknowledge that similar functionality had existed before, the site was the first to combine certain functionality in one single service, satisfying their definition of an SNS. According to [24], an SNS is a web based service that allows *"[...] individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system"*. Friendster, another popular SNS launched in 2002, extended the idea of browsing through a user's friends list by allowing friends of friends to be introduced as potential new acquaintances. Until its shutdown in 2015, the service managed to attract an unprecedented number of 75 million users in 2008 [25], and continued to grow to over 100 million users [26].

The enormous success of Friendster caused new venture-backed OSN services to emerge all over the world, causing a stiff competition for attention of users. Suddenly, small startups competed with large organizations and established OSN services altogether. MySpace, which was launched in 2003, allowed users to build a highly personalized profile page and managed to attract former Friendster users who were put off by Friendster's terms of services. Until the beginning of its decline in the beginning of 2011, MySpace managed to *"attract 95 million unique visitors"* [27]. An extensive overview of the history of OSN services is described in [24].

Popularity of OSN services did not evolve evenly around the globe, but showed regional differences. For example, Orkut, an OSN service founded by Google in 2004, failed to be successful in the US but managed to become extraordinarily

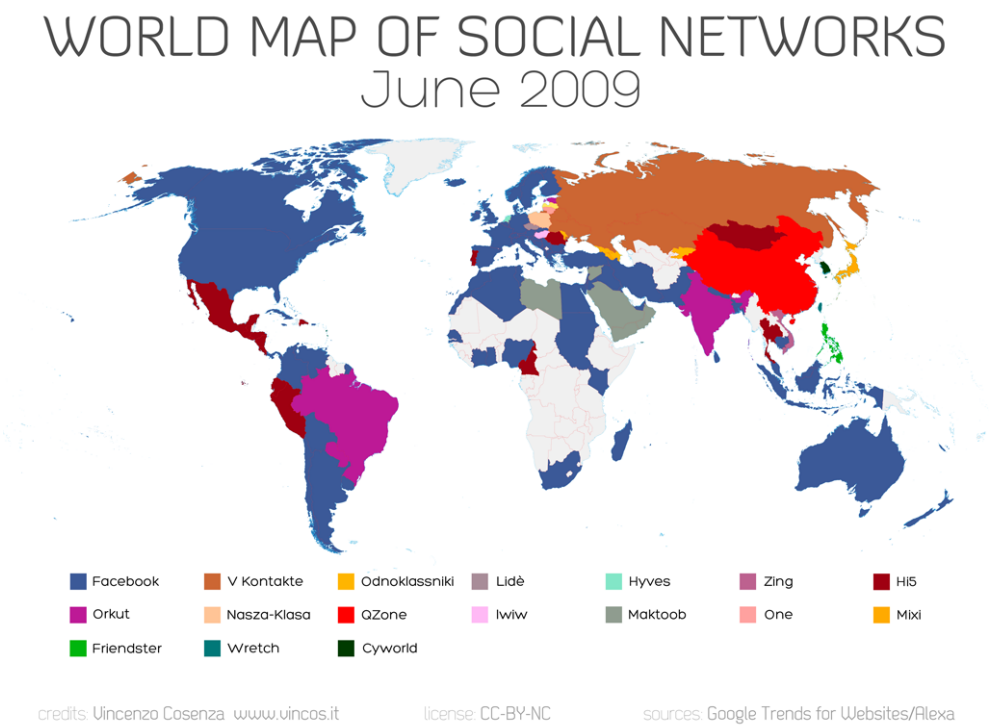


Figure 1.1: World map of OSN use in June 2009. Image source: `http://vincos.it/world-map-of-social-networks/`

popular in Brazil, while VKontakte was the most popular OSN service in Russia at the same time (see Figure 1.1) [28]. The social web was diverse with many different services available and competing for the attention of users. This was, until Facebook started to gain a critical mass of users and hence began to dominate the social web on a global scale.

Facebook was launched in 2004 by Mark Zuckerberg as a closed OSN service only available for students of Harvard university. Named *The Facebook* in its early days, it quickly attracted a significant user base and grew beyond the borders of Harvard university. The service offered users to create a social profile page including a profile picture, and specify information such as name, hobbies, or relationship status. Furthermore, users were able to specify a list of friends, join groups, and post status updates, being textual messages that were displayed in a feed for other users to see. In 2004, Facebook expanded its reach to the universities Stanford, Columbia, and Yale, which led to an increase in its user base to one million users by the end of 2004 [29]. To attract even more users, the service extended and improved its functionalities, such as introducing the Like button, a mobile application for smartphones, or photo support [29]. Consequently, Facebook was opened in 2005 to accept users from further educational facilities, such as colleges, high schools, and international schools. By the beginning of 2006, Facebook had reached six million users worldwide and started to accept registrations from everyone around the world. From hereon, Facebook's success seemed to be unstoppable. Within only four years, Facebook managed to attract 500 million users in total in 2010 [30], reaching one billion users worldwide who accessed the service at least once per month only two years later [31].

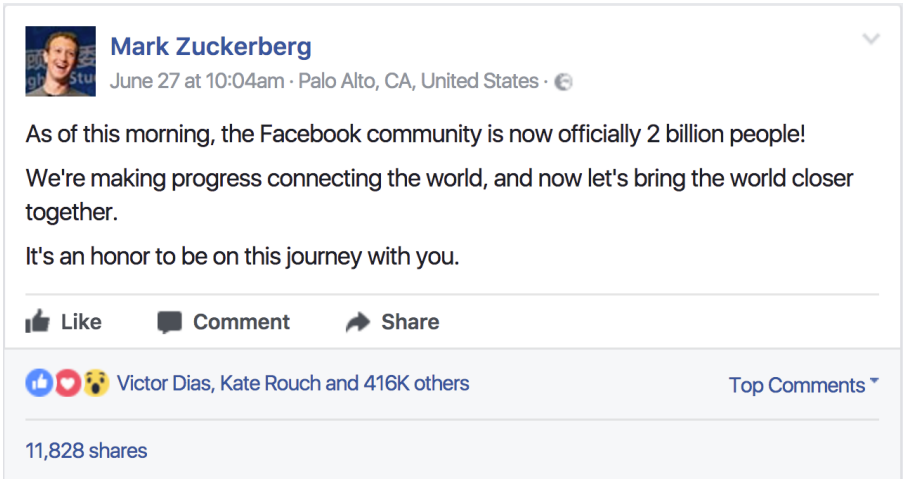


Figure 1.2: Mark Zuckerberg announcing 2 billion Facebook users in June 2017. Source: `https://www.facebook.com/zuck/posts/10103831654565331?pnref=story`. Accessed: 2.7.2017

With Facebook's extraordinarily powerful position came doubts and concerns regarding to data privacy and protection [32]. People started to feel apprehensive about the fact that one company had access to all their social profile data while claiming full ownership of all content uploaded to or created on the platform at the same time. The fact that the company used the – partly very personal – information for targeted advertisements [33] led to Facebook being perceived as an all seeing, all knowing *Big Brother* [34]. Concerns regarding to privacy violations and misuse dominance of the market even started to attract attention from the European Commission [35].

Yet, with more and more users registering at Facebook, it became increasingly difficult for competing OSN services to attract or keep customers. Many social networks were forced out of the market or decided to focus on niche markets that Facebook did not cover [24][36]. Google+ for example, which was launched in 2011 by Google with the objective to challenge Facebook's dominant position in the OSN market, ultimately failed to achieve this goal. Diversity of the social web started to decline, as users in more and more countries around the world concentrated in the market leader's OSN service. Between 2009 and 2012, global diversity of the most popular OSN services by country decreased from seventeen different OSN services to just seven, with Facebook taking leadership in more and more countries around the globe [28]. Finally, in June 2017, Mark Zuckerberg announced that Facebook had reached two billion users (see Figure 1.2) [37] – approximately 80% of all OSN users worldwide [38][39].

1.2 The Social Graph

By specifying who their friends and acquaintances are and generating content in their social profiles, users of OSN services have been creating a global network of social connections being a *"global mapping of everybody and how they are related"* [40]. This *Social Graph* represents users as nodes and connections between them as edges, where different classes of edges can be defined. The most common

edge type is the bidirectional friend relation, indicating that two users (nodes) are friends. Other edge types can also be unidirectional, such as following, liking, or commenting on a another user's content [41]. While management and storage of such an enormous dataset requires special methods [42], the social graph, as depicted by Facebook in Figure 1.3, has been attracting the interest of researchers from all over the world [43] [44] [45] [46] [47] [48]. Analysis of the characteristics of the social graph showed that a resemblance of a small world network, being an almost fully connected graph with a short average path length and high clustering [49] [43]. This can be explained by the fact that circles of friends tend to be well-connected among themselves, with fewer connections to other clusters. In the resulting graph, users have many connections to other nodes with a regional similarity and a smaller number of long range connections that help to traverse the social graph in a few steps to reach any specific node [50]. As shown by Travers and Milgram in their famous *small world* experiment in 1967 [51], individuals are connected to any other individual in the world by up to 6 hops, coining the term *"six degrees of separation"* [52]. Research on the Facebook social graph in 2011 showed that this number is actually even smaller and individuals are connected on average by 4.7 hops globally, while US American users are connected by just 4.3 hops [50]. Most of these connections are not arbitrary, as a survey by the PEW Research Center found in 2011, as only a small fraction of users are willing to connect themselves on OSN services to people whom they never met or only met once [53]. This showed, how densely connected we are as individuals, even though the average Facebook user has a median friend count of 99, where a small portion of users referred to as *hubs* are connected to several hundred or thousand users [50].

A dataset of how a significant portion of the world's population is connected, including information about personal preferences, is obviously *"valuable information to marketers, employers, credit rating agencies, insurers, spammers, phishers, police, and intelligence agencies"* [54]. As the social graph is constructed of highly personal information, individuals must be able to control who accesses what kind of their data. Yet, as stated by Bonneau et al. in [54], it is much more difficult to protect the social graph as a whole than it is to protect personal information in general.

1.3 Locked into Walled Gardens

By long, people had started organizing their everyday lives with the help of OSN services [55]. The original use case of modeling connections to friends and acquaintances had been augmented and extended, motivating users to disclose partly very personal and sensitive information *"due to the convenience of maintaining and developing relationships and platform enjoyment"* [56]. Organizing events and inviting guests, representing companies and brands including handling customer feedback, expressing opinions, text and voice-based communication, and reading the news have moved to OSN services [38]. Surveys found that 11% of American adults found it 'very hard' to give up on social networks, with another 17% claiming it would be 'somewhat hard' [15],

Figure 1.3: Visualization of the Facebook Social Graph in June 2017 [37]. Image source: `https://newsroom.fb.com`

with 90% of Facebook's users accessing the service 10 times a day [57]. With the dependence on OSN services came the problem of being bound to the service one signed up with in the first place, as services did not support interoperability between each other. Existing network effects were cleverly exploited by the platforms, as the more users are using the same service, the more value a membership for any user has [58]. Eventually, users began to realize that they were locked into walled gardens, where all their created content legally belonged to the OSN platform operator and connectivity to other OSN services was inhibited.

To be able to access more than just one OSN service, many users started to setup and maintain social profiles in multiple OSN services [38]. In 2015, GWI reported that the average user of OSN services had 5.54 OSN accounts, of which 2.82 were actively used [59]. To alleviate the organizational overhead, OSN aggregator services were introduced. Aggregator services would sign into multiple OSN accounts of a user simultaneously on his behalf and fetch content that could then be displayed in a unified user interface. Subsequently, content created by users would then be pushed to and published in all connected OSN services. While this made management of multiple OSN accounts easier, problems of data privacy were not solved, as content was simply replicated to multiple OSN services who in turn gained ownership of that content according to their terms and conditions. The social web has evolved into a landscape of isolated islands, where protocols and standards for connecting different networks. Communication and collaboration between individuals, by its very own nature decentralized and open, is now managed by centralized service operators.

Centralized communication architectures anyhow introduce several drawbacks, such as single point of failures, the threat of a service being discontinued, and often missing interoperability with competing solutions. History has shown that centralized services and architectures can be replaced by open, decentralized solutions, which ultimately are able to overcome

the drawbacks introduced by their centralized counterparts. Email, Instant Messaging (IM) services, and even the WWW itself are open and decentralized alternatives that managed to replace their closed and centralized predecessors by building on open standards and protocols [60].

Today's situation of incompatible OSN services, lacking support for data portability and interoperability between services mirrors the situation of IM services at the end of the past century. IM became a popular alternative to web-based communication via Bulletin Boards, email, or IRC in the 1990s [19] [61]. While not considered to be OSN services by the European Commission [62], IM allowed users to directly exchange messages over the web using a simple and easily understandable user interface, thus making communication feel more real-time and direct compared to predominant communication techniques such as email or bulletin boards [63]. Especially for younger users, IM quickly became the most important form of communication [64][65][66]. The first and probably most famous IM application was ICQ, which was released by the Israeli company Mirabilis in 1996. ICQ allowed users to search for other users in a directory, add friends to a contact list, and exchange text messages [67], defining the primary use of IM services as social communication, such as to connect to or stay connected with friends [68]. ICQ introduced a proprietary protocol for client-server communication, where clients would register with the central ICQ server and were identified by a numeric identifier, the UIN. To use the service, clients would login at the central ICQ server, which relayed all communication between clients and maintained the *presence* of all users, thus allowing others to see who was currently online or not [61]. The service's popularity peaked in 2001, where ICQ reported a user-base of 100 million registered users [69]. The enormous success of ICQ led to the development of several alternative messenger services such as AOL Instant Messenger (AIM), Microsoft Messenger (MSN), or Yahoo Messenger, which provided similar functionality but each implemented individual proprietary protocols and were hence incompatible with each other[2] [67].

To alleviate the struggle of users being forced to have multiple IM applications installed simultaneously in order to stay connected with friends in different IM networks, multi-messengers were introduced. Multi-messengers such as Trillian[3], Miranda IM[4], or Pidgin[5] allowed users to handle multiple user accounts from one single application by simply utilizing the proprietary protocols of the respective IM services and connecting to all of them at the same time. While the overall user experience was streamlined this way, users still needed to register with multiple IM service providers in parallel, as communication between different IM services was mostly not supported. Moreover, group chat communication or any form of collaboration with users from different networks was not possible.

[2] AOL bought ICQ from Mirabilis in 1998, eventually replacing the ICQ's native protocol with AOL's OSCAR protocol [70][71]. OSCAR was developed as an proprietary protocol for AOL's IM service AIM and was opened to support third party applications in 2006: `https://en.wikipedia.org/wiki/OSCAR_protocol`. Accessed: 13.5.2017

[3] Trillian: `https://www.trillian.im/`. Accessed: 14.5.2017

[4] Miranda IM: `http://www.miranda-im.org/`. Accessed: 14.5.2017

[5] Pidgin: `https://pidgin.im/`. Accessed: 14.5.2017

Jabber, which was later renamed to XMPP [72], revolutionized IM in 1999 using an extensible, federated, and open protocol and architecture. Similar to email, user accounts in Jabber could be created at any server, which then relayed messages to the server of one's communication partner, thus eliminating the need to entrust one's data to more than one IM service provider. This way, Jabber not only freed users from the need of having multiple user accounts with different IM service providers, but also from being forced to trust the companies behind the IM services as everyone could run a Jabber server and host his own user account, or simply choose a Jabber provider one trusted.

Today's situation of incompatible OSN services, lacking support for data portability and interoperability between services reflects the situation of IM at the end of the past century. Missing standards and protocols prevent forming a truly open and decentralized social web, in which users control and own their data, while not being locked into the walled garden of the OSN service they once decided to sign up with. In this thesis, a solution for interoperability of OSN services is presented, allowing users to break free of the firm grip of OSN providers.

1.4 Motivation

"We all are on Facebook because we all are on Facebook"

– Michael "MSPro" Seemann, *Re:publica 14*

Today's major OSN platforms are mostly implemented in a closed, proprietary fashion. Even though access to parts of the services' functionality is being made available via proprietary APIs, OSN platforms keep their users from seamlessly connecting to users of other services to create well-calculated *lock-in effects*. This hinders users from being able to freely communicate with services of competitors while the individual cost of migrating to a competitor's service is intentionally kept high. In addition, the created *network effects*, described as effects where *"the consumption benefit of a network good is proportional to the total number of consumers [...]"* [60], are utilized to decrease the attractiveness of alternative services with less users. Accordingly, the more users a certain OSN platform is able to attract, the more each user benefits from the network effect of the platform as finding a specific individual's user profile on this platform becomes more likely. Vice versa, these network effects keep users from using other OSN services, leading to the current situation of isolated islands of OSN services, in which users are locked into *walled gardens*, being proprietary platforms or ecosystems in which the provider has full control over applications, content, and communication, effectively binding their users to the platform. Here, *"[...] pictures, videos and everything else is stranded [...], cut off from the rest of the web"* [73].

As a result of this intentional isolation of OSN services in terms of user interaction, a user's choice for a certain OSN platform automatically becomes a choice against all other options, unless one is willing to actively maintain more than one social profile simultaneously. In June 2017, Facebook as the market leader reported 2 billion monthly active users (MAU) [37]. As depicted in Figure

1.4, the more successful competitors of Facebook were able to attract between 100 and 600 million monthly active users in the same time[6] [74] – a difference of over a billion users. Even Google+ is believed to have attracted only approximately 500 million monthly active users in 2015[7] [75] and is expected to be discontinued [76].

Facebook being today's most dominant OSN service can be explained as a intentional consequence of the platform's design as a walled garden, thus creating network effects or *social gravitation*. The term *social gravitation* describes the phenomenon where a social web platform is more attractive to users the more users it already has, thus being able to attract even more users [77] and fueling the principle of *the rich get richer*. As analyzed by Westland [58], OSN services don't experience the benefits of a first-to-market effect after being launched, as network effects in OSN services are weak in early stages. Yet, being the first service to be launched helps to beat competitors in the race of attracting a critical mass of users, which again is crucial for sustaining an OSN service in the long term [78]. After such a critical mass of users is reached, a giant cluster forms in the social graph, causing implicit network effects [79] to manifest that cause the service to become *"self sustainable"* [58]. Seemann claims that these effects cause smaller services to be forced out of the market, as most users will choose to sign up with the dominating OSN service, leaving no room for a *"second place"* on the market [77]. The simple explanation for this effect is that for a potential new user of OSN services, it is more reasonable to register with an OSN platform most of his friends are already using instead of signing up with a different service, even though it might be better suited for this user's demands or expectations. As OSN services do not allow seamless communication between different OSN platforms, choosing a platform not used by the majority of OSN users will essentially cut one off from communication with most other OSN users. Even though this might be desirable in some special scenarios, being locked out contradicts with the basic idea of the social web of being able to communicate and sharing content with each other. The consequence of this being that *"we are all on Facebook because we all are on Facebook"* [77]. OSN platform operators have learned to use this to keep existing customers and attract new ones simultaneously. As stated by Seemann [77], timing is an important factor in becoming the OSN service with the highest social gravitation, as being the first service on the market might allows it to attract a critical mass of users before competitors enter the market. An example of this effect is the success of Twitter, which managed to attract more users than StatusNet, a service with a similar concept that was being developed at the same time, but wasn't market ready yet when Twitter was publicly launched. Consequently, StatusNet had no chance to compete with Twitter, as Twitter, which was launched in 2006, had already gained a large user base as described by Westmann [58] at the time StatusNet was released in 2009.

The design of OSN services as walled gardens and the resulting lock-in effects for customers [73] hinder users from moving away to a competitor's platform in case they become dissatisfied with the current service. The term *lock-in*

[6]Messenger services are not considered as OSN services per definition by Boyd and Ellison [24] and the European Commission [62] and are hence not considered here.
[7]Google does not release official numbers regarding to MAUs in Google+.

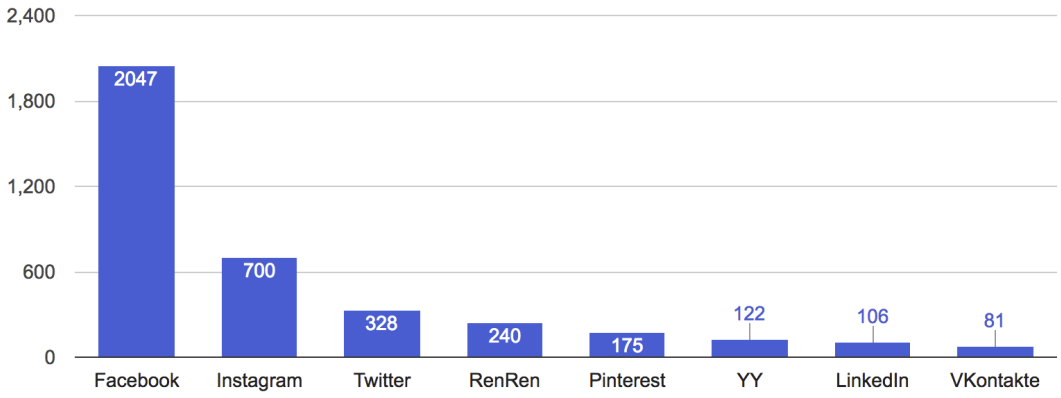


Figure 1.4: Number of active users of OSN services (in millions) in August 2017 [74][83].

effect originally stems from the field of economics where *vendor lock-ins* inhibit customers to use a competitor's product, and make switching to a competitor's product unreasonably expensive [79]. The term was applied to social web services in 2010 by The Independent [80], describing a technological lock-in where abandoning an OSN platform for another one results in losing one's social profile along with all content and connections to other individuals. The more content and information one had created by using the service and the more friends are using the same OSN service, the stronger the OSN lock-in effect is. Technologically, lock-in effects are created by the use of proprietary interfaces and data formats, disallowing interoperability and data portability. This way, third party services, including competitor OSN platforms, are not able to communicate from the outside with an OSN service unless explicitly allowed by the operator, while users are kept from using their social profile data with services or migrating to a competitor's OSN platform altogether [81]. For example, befriending users, sending invitations to an event managed on an OSN platform, or simply posting on another user's wall is not possible if the addressed user is using a different OSN platform. Essentially, a *"stickiness"* was created that kept users from moving away from the networks they were caught in [82].

As network effects and social gravitation cause the global user-base to converge in few OSN platforms (see Figure 1.4), sensible and personal data of billions of users is stored and managed by only a small number of companies [73]. This practice resulted in an ever ongoing conflict of interests, as OSN providers often automatically claim ownership of the managed data, including personal information, pictures, and text messages, which is then used for targeted advertisement and data analysis [84][85]. This enables OSN service providers to create highly detailed profiles of their users, which are based on personal information most users wouldn't want to be used by anyone. Facebook, for example, states in their terms and conditions[8] that they claim *"non-exclusive, transferable, sub-licensable, royalty-free, worldwide license to use any IP content"* that is posted on the platform, and further sharing information about their users with

[8]Facebook terms and conditions: `https://www.facebook.com/terms`. Accessed: 18.7.2017

"vendors, service providers, and other partners who globally support our business"[9]. The implications of this concerns users, as control over one's data privacy is essentially lost [40]. As a result, the amount of information people willingly specify in their OSN profiles, such as a home address or phone number, declined over time [86] as users realized that *"if you are not paying for it, you're not the customer; you're the product being sold"* [87]. The prevalent loose handling of personal data by OSN providers resulted in a *"well documented frustration"* of users [81] who wanted to *"jump out of the walled gardens"* [85]. Researchers and users of OSN services started to demand that OSN services should *"open up"* [73], allowing the social graph to be transformed into a *"community asset"* [40].

In 2007, Smarr et al. proclaimed a *Bill of Rights for Users of the Social Web*, in which they assert an entitlement of all users of the social web to certain fundamental rights [1]. These rights comprised ownership of personal information such as profile data, friend lists, and activities, including full control over how this data is shared with and used by other users and services. OSN services were urged to allow their users to *"[...] syndicate their own profile data, their friends list, and the data that's shared with them via the service, [...] using a persistent URL or API token and open data formats; [...] syndicate their own stream of activity outside the site; [...] link from their profile pages to external identifiers in a public way; and [...] discover who else they know is also on their site, using the same external identifiers made available for lookup within the service"* [1].

The situation of the social web being controlled by a small number of OSN providers led to the development of distributed, federated OSN services [85][88]. A number of decentralized OSN (DOSN) services relying on peer-to-peer (P2P) technology [89][90][91][92], a federation of loosely coupled servers [93][94], or hybrid approaches [95] were proposed. For example Diaspora[10], a highly anticipated approach, aimed for an open and decentralized OSN service, where users would host their social profiles on *pods*, which are connected in a federated fashion. The service raised high expectations and aimed at developing the foundation for a federated social web, allowing users to choose their OSN server or even host their own pod [96]. Ultimately, users would start developing their own pod software, which would lead to a federated social web connected by the Diaspora protocols [97]. Yet, even with the existing federated or P2P solutions, people are still confined within one OSN system where communication with other OSN systems is not possible at all or limited to few options. In the majority of federated or P2P environments, even though users are able to control their own data by either hosting their profiles themselves or entrusting profile data to a provider of their choosing, seamless communication with other platforms is still not possible. Moreover, data portability, being the ability to move a profile to another provider, is not possible without manually copying all data associated with a social profile [98][99]. Exceptions to this are Friendica[11] and Diaspora, which allow limited exchange between the two networks by implementing the respective other platform's native protocol and allowing basic data portability through manual extraction and re-import of social profile information.

[9] Facebook data policy: `https://www.facebook.com/about/privacy/`. Accessed: 21.5.2017
[10] Diaspora: `https://diasporafoundation.org/`. Accessed: 22.5.2017
[11] Friendica: `http://friendi.ca/`. Accessed: 22.5.2017

Other approaches to connect incompatible OSN platforms make use of integration plugins or apps, such as SocialDeskApp[12], or middleware platforms, such as SNSAPI [100] or OneAll[13]. Both approaches have in common that communication with connected OSN platforms is facilitated via the proprietary APIs and protocols these platforms offer. Consequently, users need to be registered with each connected OSN platform in order to be granted permission to use these APIs to exchange information with different platforms [101]. Data is then fetched from all connected remote OSN services and combined in the user interface, while write operations, such as posting status updates, lead to the content being replicated to all connected profiles [101]. Using these solutions, most profile data is accessible across OSN platform borders, but the approach fails to allow users to seamlessly connect to each other. Moreover, core functionality of OSN platforms such as group discussions, event management, or friend list management cannot be realized using these approaches. As building connections between profiles and communicating with other individuals is one of the most important features of the social web, using integrator plugins and middleware solutions cannot solve the issue of walled gardens. In fact, OSN integrators or middleware platforms even aggravate the problem of data privacy issues by replicating data to multiple servers and accounts, making it accessible to multiple OSN platform providers in the process [101].

Besides proposing alternative OSN architectures and services, efforts to connect OSN services resulted in an extensive collection of microstandards such as Activity Streams 2.0 [102], Web Finger [103], or Friend-of-a-Friend (FOAF) [104]. While many OSN services adopted some of these standards in their APIs and protocols, their use is still kept proprietary and does not allow seamless inter-platform communication [13].

1.5 Problem Statement

The current situation of the social web [12] can be characterized as a landscape of different OSN services, designed as isolated data silos [85]. These silos are shut off from each other by the use of proprietary APIs, protocols, and data formats. Users of OSN services tend to have multiple accounts in order to be able to keep in touch with friends in different OSN services [59], resulting in an overlap of information [98]. Yeung et al. note that given that OSN services are preventing interconnectivity with competing services, users *"would like to 'jump out of the walled gardens' [...] to share their data with their friends who may be members of other social networking sites."* [85]. After all, social communication and collaboration is, by its own nature, peer to peer and not centralized [101]. In the envisioned decentralized social web, users would not be restricted to the OSN service they signed up with in the first place anymore [85].

With the aim to *"build their own social web"* [73], programmers and researchers proposed alternatives to the closed, proprietary silos, driven by the belief that *"a truly universal, open, and distributed social web architecture is needed"* [105] and

[12] SocialDeskApp: `http://socialdeskapp.com/`. Accessed: 22.5.2017
[13] OneAll: `https://www.oneall.com/`. Accessed: 22.5.2017

that the Internet needed *"a way to take the features of the popular social networks and make them available to the world at large"* [73]. Applequist et al. identified four major problems for users of closed OSN services, being the ability to move social profile data between OSN services (portability), using one identity across the entire social web instead of separated ones (identity), the ability to link to social profiles and data from outside the OSN service storing this data (linkability), and being in control over access and use of one's social profile data (privacy) [105]. They proposed an approach for a *"standards-based, open and privacy-aware social web"*, in which users would own and control their social profile data in a trusted location while disclosing selected parts of it to selected OSN services. This way, online personas could be created with different scopes, for example for personal and professional use, where only content suitable for the respective persona would be accessible in the respective OSN service [105]. Still, following this approach would require users to sign up with multiple OSN services, where data is replicated to the connected social profiles [101], a practice that users are *"sick of"* [40].

While proposing alternative architectures to address network and lock-in effects [82], researchers acknowledged that a major challenge of DOSN services is adoption by users [85]. Applequist et al. state that *"participation is the life blood of social networks. If [...] too few people participate, a social networking application dies"* [105]. As found by Westland [58], OSN services experience network effects after they reached a critical mass of participating users. According to perlocation theory, at this point a phase change happens in the social graph and a giant cluster of users forms in the network [58]. An OSN then becomes self-sustaining and is able to attract even more users for the simple reason that most likely, most of the friends, relatives, or acquaintances of a certain user are already using the OSN service in question. Hence, a user is not free anymore in his choice of an OSN service and has to sign up with the dominant OSN service as otherwise, he would be cut off from the rest of the social web. As a result, OSN services that manage to achieve a certain critical mass of users as the first service in the market massively benefit from implicit network effects and are therefore able to easily dominate the market [58].

1.5.1 Challenges

To address the issues of the current social web, Fitzpatrick and Recordon defined in their much-noticed article *Thoughts on the Social Graph* several goals, including to *"make the social graph a community asset [...]"*, namely by establishing a *"[...] non-profit and open source software"* ensuring *"[...] that the design [of components] is such that others can run their own instances, sharing data with each other"* [40]. As to this date solutions built with these requirements in mind were not able to attract a significant number of users, Paul et al. defined a list of requirements in their survey on DOSN services [106]:

- **Transparency**: The distribution of services needs to be entirely transparent to the user.
- **Integration**: Access to data and functionality needs to be accessible via a single integrating interface covering search, publication, and retrieval of information.
- **Functionality**: All data related functions of central OSN, such as chats, posts, pictures, or profiles, need to be provided.
- **Relations**: Relations between users in the social graph need to be represented.
- **Availability**: Availability of data and services must not be interrupted.
- **Confidentiality**: Confidentiality of data and communication must be ensured.
- **Access Control**: Access control must be implemented and enforced, even though a central authority is missing.
- **Privacy**: Privacy must be supported for users and their data.

Also, Koll et al. analyzed factors of success of DOSN services and provided a list of nine challenges DOSN services have to accomplish in order to be successful in the market [101]:

- **Independence**: Storage and management of social profiles and data must not depend on any provider.
- **Free-of-Charge**: Membership and usage of the service must be free of charge for the users.
- **Online Times**: The service needs to be available at all times.
- **Mobile Support**: Access to the service must be possible from mobile devices, ideally via a dedicated mobile application.
- **Efficiency**: Communication and storage design must minimize unnecessary overhead.
- **Scalability**: The service must be scalable to several millions of users.
- **Resiliency**: The service must implement protective measures against attacks and other kinds of misuse.
- **Privacy**: Users of the service must be able to control access to their data on a very fine-grained basis.
- **Performance**: The service has to be performant.

Koll et al. argue that while an ideal OSN service would satisfy all listed challenges, some of them are rather mutually exclusive [101]. Hence, a balanced compromise needs to be found for a DOSN service to be successful. For example, independence of external resource providers and efficiency are hard to satisfy at

the same time, as a central system is able to prevent communication overhead far better than a decentralized one. Hence, a DOSN service can either reduce communication overhead or dependence on (central) providers. Other examples are that performance and high availability is financially hard to achieve when the service is free of charge for its users, and that privacy control prevents business models as employed by Facebook. Koll et al. conclude that a distributed, federated approach where social profiles and associated data is hosted on a user's home gateway provides the best option to build a successful DOSN service [101], where the feasibility of this approach has been proven [107].

1.5.2 European Law Perspective

The situation of a social web consisting of closed, proprietary walled gardens has also been addressed by European law. In an analysis of European regulatory and competition law issues, Graef argues that OSN services are mainly multi-sided businesses with users on the one and advertisers on the other side, as indirect network effects between users and advertisers can be clearly identified [35]. These indirect network effects, as described by Katz and Shapiro [79], result in an increased utility of the service for customers as the overall number of consumers of the service increases. In case of OSN services, a higher number of users results in an increased value of the service for advertising companies, as more users can be reached by publishing advertisements in the OSN service. Graef further notes that OSN markets are *"[...] typically quite concentrated, since it is necessary to have a critical mass of customers"* to be able to succeed in the market, making it *"[...] difficult for competing platforms to gain a foothold in the [OSN] market"*, what ultimately tends to *"[...] limit the number of viable firms in a market"* [35]. Following the European Court of Justice's definition of dominance, being *"a position of economic strength enjoyed by an undertaking which enables it to prevent effective competition being maintained on the relevant market by giving it the power to behave to an appreciable extend independently of its competitors, customers, and ultimately of its consumers"* [108], Graef sees Facebook in a dominant market position, effectively hindering competitors being successful in the OSN market. Seeing OSN services as communication platforms, Graef argues that legal action should ensure that competition is made possible again. To alleviate the current situation of one OSN platform dominating the entire OSN market, Graef proposes to mandate regulatory action based on European competition laws where two aspects should be addressed in specific, being data portability and interoperability.

Graef describes data portability in the domain of OSN services as a user's ability to automatically move their social profile data including photos, posts, and friend lists to a competitor's service, where *"technical standards have to be developed to ensure that data portability can be effectively implemented [...]"* so that *"it [is] possible for data extracted from one social network to be seamlessly inserted into another [OSN service]"* [35]. Yet, data portability as proposed in the General Data Protection Regulation of the European Commission merely grants the extraction and transfer of data that would allow to identify a user and would therefor not necessarily allow users to transfer all social profile data to a competitor's OSN platform. Graef argues hence that regulation is necessary to ensure

interoperability is implemented, giving users the ability to *"[...] connect and interact with each other irrespective of their social network provider"* [35], thus being even more powerful than data portability alone. Graef argues that implementing OSN network interoperability would be *"a way to redress network effects and increase competition in the [OSN] market [...]"* furthermore reducing *"[...] switching costs and the degree of user lock-in [...]"* as *"[...] the number of people that a user can reach is not limited anymore to the number of users on the social network that the user decided to join"* [35]. Furthermore, Graef states that *"[...] interconnection requirements should be imposed in general on all social networks and in all situations"* as *"real interoperability can only be established when all social network providers are obliged to participate in the process"* [35]. Finally, Graef notes that mandating support for data portability and interoperability may affect the business models of OSN services, yet will encourage new services to enter the market, leading to more and healthy competition and consumer choice, and will ultimately result in a better protection of the rights and interests of users of OSN services [35]. As of 2016, data portability has been regulated in the General Data Protection Regulation (GDPR) by the European Union (EU) [109]. The regulation addresses a mandatory ability to export personal information *"[...] in a structured, commonly used, machine-readable, and interoperable format [...]"* including the ability to transmit the exported information to other services. With the intention to *"further strengthen the control"* over one's data, the regulation is enforceable as of May 2018, yet lacks a technical implementation of how data should be exported, described, or re-imported. As pointed out by Sperlich [110], this creates technical challenges, as service providers mostly use proprietary data formats which are tailored to the individual data processing systems with the intention to create lock-in effects for customers.

1.6 Research Questions

To address the prevalent situation of centralized OSN service dominating the social web, a novel kind of service and architecture is required. Following the requirements for an open, distributed, and privacy-aware, social web as discussed in Section 1.5, the following research questions have been identified:

Research Question RQ1: Enabler for OSN Diversity

The existing strong network effects in today's OSN service landscape threaten the diversity of OSN services. As social gravitation draws users to the OSN services with the most users, it is nearly impossible for new OSN services to attract users. A solution needs to provide means to weaken existing network effects to allow new and smaller OSN services to succeed in the market.

Research Question RQ2: Mapping OSN Features

Today's OSN services provide a rich and diverse set of functionality. Building a service that aims to connect all possible OSN solutions needs to be able to provide a solution for communication between OSN services with different or incompatible featuresets, while covering functionality of existing OSN services.

Research Question RQ3: Enabling OSN Interoperability

Today's OSN services are mostly not allowing interoperability between each other. While microstandards and protocols exist for specific issues, a holistic standard for interoperability of OSN services does not exist. Hence, a solution needs to implement a holistic protocol including data formats that are able to provide interoperability between arbitrary OSN services.

Research Question RQ4: Enabling Data Portability

Today's OSN services offer only limited manual export of profile information, if at all. To this date, a complete and automated profile export from one and re-import at another OSN service is not possible. Furthermore, moving profile data between service domains will inevitably break links to other profiles and data. Hence, a solution does not only need to implement portability of profile data, but also ensure that connections to other profiles stay intact.

Research Question RQ5: Identification

User profiles in today's OSN services are usually identified by a locally unique identifier or username, which is used in combination with the service's domain name. As of this, user identifiers are bound to the OSN service they were created in. Allowing migration of social profiles between service domains hence needs to implement a domain-agnostic identification architecture that allows user identifiers to remain valid after an account has been migrated to another service domain.

1.7 SOcial Network InterConnect

To address current issues of mostly proprietary and closed OSN architectures, several alternative architectures and paradigms have been proposed. Existing architectures can be categorized into centralized and decentralized approaches, of which the latter can be distinguished further into federated, P2P, and hybrid [111]. All existing approaches have in common that users need to register a separate OSN account, which cannot be used with other OSN services, while seamless interoperability between services is not possible or severely limited. Users and their data are therefore still locked in with alternative OSN architectures, what contributes to the lacking willingness of users to abandon their accounts in the predominant OSN services. Designs for *Yet Another Social Networking Service* (YASNS) would hence face the same problems as existing solutions have, unless successfully addressing and solving the aforementioned issues existing in OSN services.

In contrast, a solution that addresses the identified research questions and challenges, thus providing interoperability, seamless connectivity, and data portability between arbitrary OSN services would introduce several advantages to the entire landscape of OSN services. As stated by Palfrey and Gasser, interoperability of systems and services does not only provide a greater choice and autonomy for the consumer, but also supports competition and innovation,

while additionally providing benefits for both customers as well as providers in terms of systemic efficiency [60].

1.7.1 The Sonic Vision

In this thesis, SOcial Network InterConnect (Sonic), a distributed architecture and design for an open and decentralized Online Social Network Federation (OSNF) is presented, being a *"heterogeneous network of loosely coupled OSN platforms using a common set of protocols and data formats in order to allow seamless communication between different platforms"* [3]. Sonic is built on the idea of utilizing and combining existing OSN microstandards to provide a holistic framework for OSN interconnectivity. The vision of Sonic is an open, seamlessly interconnected, heterogeneous ecosystem of OSN platforms, in which users are not restricted to communicating with connected users of the same OSN platform, but can seamlessly interact and collaborate with users of other OSN services as platform borders become transparent. Lock-in effects, keeping users from abandoning an OSN service they are dissatisfied with, are eradicated as user profiles may be freely migrated from one OSN platform to another at any time without losing established relationships in the social graph. This would allow users to freely choose an OSN platform of their liking instead of being limited in their choice to the platform used by one's friends. In this thesis, the following definition of the OSNF is used:

> **Definition 1: Online Social Network Federation (OSNF)**
> An *Online Social Network Federation* (OSNF) is a heterogeneous network of loosely coupled OSN platforms using a common set of protocols and data formats in order to allow seamless communication and interoperability between different OSN platforms. While heterogeneity allows the participating OSN platforms to support different OSN featuresets or different implementations of OSN features, the Sonic core featureset must be supported.

To realize this vision of an ecosystem of freely interconnected ONS services, Sonic proposes a holistic approach comprising an open and extensible social API as well as data formats. The APIs and data formats proposed by Sonic are built around existing open standards to allow for a high compatibility with existing implementations while easing implementation and integration overhead for developers at the same time. This allows any OSN service provider to implement and integrate the required protocols and interfaces and connect existing OSN platforms to a global ecosystem of OSN services as envisioned by Yeung et al. in [85]. Furthermore, as more and more smaller OSN services connect themselves in the OSNF, a critical mass of users and services could be used to make the OSNF self-sustainable as described by Westland in [58]. This could cause a disruption of currently employed business models focused on centralization.

Eradicating lock-in effects in the way envisioned by Sonic also allows users to freely communicate between different platforms and even migrate between OSN platforms at any time without losing any social profile data or connections

to other users. Addressing the issue of social profiles being bound to the OSN platform they were created in by providing a holistic solution allows users to maintain social profiles at any server they want while allowing them to migrate profiles to a new server if wanted.

Hence, Sonic introduces a solution for the effect of social gravitation as users of Sonic-compliant OSN services are not locked-in anymore and the individual choice of a user for a specific OSN service does not automatically cut him off from communicating with other OSN services. As choosing a small OSN service instead of the one with the largest user base does not isolate users anymore, smaller or new OSN services stand a much better chance on the market. If OSN service providers cannot rely on network effects anymore, they need to compete for users via alternative benefits, such as better functionality, performance, support, or privacy. Consequently, Sonic re-introduces competition and innovation in the consolidated OSN market.

The outcome is a federated, heterogeneous social ecosystem, in which users are able to maintain full control and ownership of their social profile data. Such an ecosystem allows OSN platforms to be connected by loose coupling and exchange informations using common data formats and APIs. The resulting OSNF as of Definition 1 would therefore allow any OSN platform to communicate freely with any other Sonic-compliant OSN platform as depicted in Figure 1.5.

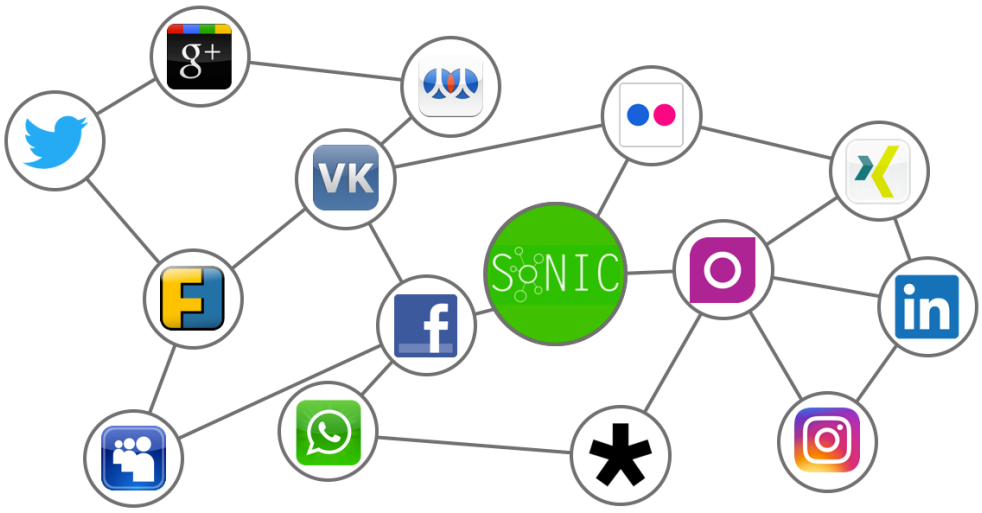


Figure 1.5: Sonic OSNF: Any OSN platform is loosely coupled with any other OSN platform using common data structures and protocols. The result is an open and heterogeneous OSN federation.

1.8 Contribution

This thesis proposes a holistic architecture for a distributed, heterogeneous OSN ecosystem, the OSNF. The envisioned solution addresses existing issues such as data privacy, lock-in effects, and missing interoperability. Sonic not only allows users to connect to each other across platform borders and seamlessly communicate, but also to move their profiles between OSN platforms without

losing data or connections to other user's profiles. Current issues of centralized, proprietary OSN platforms, such as the lack of data privacy, lock-in effects or the walled garden phenomenon, can be avoided altogether with the proposed solution. Challenges for DOSN services as identified by Applequist et al, [105], Paul et al. [111], and Koll et al. [101] are either addressed directly by design, or can easily be implemented by OSN platforms. The following solutions for the area of DOSN ecosystems are a contribution of this thesis:

1.8.1 Definition of a Core Featureset of OSN Platforms

OSN platform implementations vary in the features and functionality they provide as OSN platform providers implement new features to follow the latest trends and keep up with competitors to attract new customers and defend market shares [112]. Still, OSN platforms provide a set of basic core functionality, which usually does not differ from the implementations of other OSN platforms. Such features comprise for example profile pages, messaging functionality, or liking content of other users. To allow the definition of a common social protocol and set of data formats, Sonic defines a taxonomy for features of OSN platforms and derives a core featureset of OSN functionality [5]. The features comprised by the core featureset are supported by almost all existing OSN implementations, while implementation details rarely diverge. To support features not comprised by the core featureset, feature extensions are used to allow OSN platforms to create a unique user experience by providing unique features to their users.

1.8.2 Privacy Preserving OSN Architecture

Most of today's popular OSN platforms, such as Facebook, Google+, or Twitter, are built in a centralized manner. This allows the operators of these services to exert full control over all user data. Alternative architectures have been proposed to allow users to regain control about their data, including mobile-hosted, decentralized approaches, some of those being organized as federated or peer-to-peer based, or even hybrid approaches [113][111]. The idea behind these alternative architectures is that a user should be free to chose the provider to host the social profile and control access to the profile's contents. Sonic proposes a decentralized, federated OSN architecture, similar to Friendica or Diaspora, but builds on the idea of seamlessly connecting various different kinds of OSN implementations. The architecture is fully decentralized to prevent situations in which control of a vital component of the social ecosystem lies in the hand of a single individual or company, allowing for potential abuse. This allows users to host their own social profile if desired and remain in full control of the comprised data. This builds the foundation for an open, heterogeneous OSN federation, the OSNF.

1.8.3 APIs and Data Formats for Seamless OSN Interoperability

Even though a variety of data standards and protocol exist that address communication and data exchange in the social web [105], a holistic approach that facilitates seamless communication between different OSN platform implementations does not exist. Existing solutions focus on mostly isolated issues and tasks, such as describing activities [114], discovery of user profiles [103], or modeling links between users [104]. Sonic addresses this issue by proposing a holistic approach, comprising data formats and APIs that allow seamless exchange of data between different OSN platform implementations. The proposed data formats are built to be compatible with existing microstandards such as Activity Streams 2.0 [102], Open Social [115], or OStatus [116] to ensure easy integration in existing OSN implementations. The protocols proposed by Sonic cover functionality supported by the majority of existing OSN implementations, allowing an easy integration into an existing OSN platform.

1.8.4 Global User and Object Identification

The way how OSN platforms identify users and data objects, such as a user's profile page or a posted status update, differs between OSN platforms. Normally, OSN platforms issue locally unique identifiers for users, which can be resolved to the user's identity or data object only in conjunction with the issuing platform's domain name. As of this, the issuing platform always remains responsible for routing requests for user identities or data objects, even after migrating to a competitor's service. Users of migrated accounts would therefore have to trust their original OSN provider to allow resolving of identities of remote social profiles. To prevent this kind of situation, Sonic proposes domain-agnostic identifiers for all user identities as well as all social profile data. Sonic introduces the concept of GlobalIDs, which are domain-agnostic, globally unique identifiers for users. Data objects are identified via Unique Object IDs (UOIDs), which comprise the object creator's GlobalID. GlobalIDs - and therefore UIODs - are resolved via the Global Social Lookup System (GSLS), a distributed DHT-based directory service. GlobalIDs are self-issued, where the ownership of any identifier can be verified using digital certificates.

1.8.5 Data Portability for User Accounts

The well calculated lock-in effects of today's centralized OSN platforms keep users from moving their social profile to a competitor's service. In case a user is not satisfied anymore with his current OSN platform or its terms of usage, he is unable to move to a competitor's platform without losing all his social profile data and connections to other users. The use of open OSN architectures such as federated or P2P-based approaches cannot solve this issue, as a user would again be caught within the boundaries of this architecture. Without means to freely move a social profile to any other OSN platform, federated approaches still lock the user into their domain.

Sonic addresses this issue through migration functionality, which allows to extract a complete social profile from any Sonic-compliant OSN platform and import it to another OSN platform of the user's choosing. Connections between social profiles, such as the friend roster or comments from other users, are kept intact by the migration process.

1.9 Research Methodology and Outline

The research in this thesis follows the methodology of design research as described by Hevner et al. in [117]. The design research methodology aims at innovation by creating new and innovative artifacts in order to solve specific problems in research, where artifacts can be constructs, models, methods, or instantiations. Hevner et al. [117] defined a set of guidelines for conducting design research, where the research process can be separated into three research cycles that provide a framework for research activities [118]. Here, the relevance cycle describes the definition of the research problem and an assessment of its relevance. The rigor cycle then evaluates and applies knowledge from related work from other researchers to ensure a novel contribution is created by the research activity. Finally, the design cycle designs, implements, and evaluates artifacts, and furthermore provides a detailed formal description of each artifact. The contribution of this thesis and its relevance have been motivated and assessed in the relevance cycle, where the State-of-the-Art has been analyzed in the rigor cycle. The contribution of this thesis can be distinguished into four separate artifacts, being the definition of a common core featureset of OSN services, the concept and design of a distributed user identity management, the Sonic architecture, and the Sonic protocol. These four building blocks address the aforementioned research questions and provide a holistic solution for the identified problem. The designed artifacts have been designed, implemented, and evaluated in the design cycle.

The structure of this thesis reflects the individual tasks addressed by the three cycles defined by design research. Chapter 1 provides an extensive overview of the history of distribution of OSN services and describes the motivation for the research as well as research questions to be addressed. Chapter 2 then gives an overview of approaches for facilitating distribution of OSN platforms. Chapter 3 presents use cases motivating the idea and describing benefits of an open OSNF, followed by a list of identified requirements for the proposed solution. The rest of the chapter is organized in four main sections, of which each describes the concept and design of one of the aforementioned designed artifacts, each being a major building block of the holistic Sonic solution. Each section describes its own overview of related work as well as the concept and design of the proposed solution. These sections describe a taxonomy of OSN features (Section 3.4), user identification management (Section 3.5), the OSN platform architecture (Section 3.6), and the Sonic protocol (Section 3.7). Chapter 4 describes the implementation of the distinct components, followed by the evaluation of the concepts and components of Sonic in Chapter 5. Chapter 6 concludes the thesis.

2 Related Work

As of today, OSN services have become an integral part of our everyday digital lives. We express ourselves, communicate, and even collaborate using social services and applications. While functionality of early social platforms, such as Classmates.com or Sixdegrees.com was mostly limited to discussion boards and modeling and maintaining relationships with friends and colleagues, today's OSN services have become one of the main communication and collaboration platforms. Here, users are able to communicate via text, audio, and video, and furthermore share content, plan events, or just stay in contact with friends and relatives. While early OSN platforms were mostly organized in a closed, proprietary fashion, several architectures, services, and protocols have been proposed that aim at decentralizing aspects of OSN services, storage and management of the users' personal data and information, or entire service architectures altogether. This chapter provides an overview of existing definitions and categories of OSN services as well as of the proposed approaches to decentralize OSN services.

2.1 Online Social Networks

To describe and analyze OSN services, several definitions of OSN services have been given in scientific literature. Existing literature employs various terms for social web services, including *Online Social Network* (OSN), *Social Web Site* (SWS) and *Social Networking Site* (SNS). While the terms are mostly interchangeable, this thesis exclusively uses the term *Online Social Network* (OSN) to prevent inconsistencies and possible misunderstandings. Following, an overview of definitions and classifications of OSN services is provided.

2.1.1 Definition

One of the most accepted and used definitions of OSN services was given by Boyd and Ellison in [24], being a web based service that allows *"[...] individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system"*. Boyd and Ellison gave another, extended definition in [119], defining OSN services as *"[...] networked communication platforms in which participants 1) have uniquely identifiable profiles that consist of user-supplied content, content provided by other users, and/or system-provided data; 2) can publicly articulate connections that can be viewed and traversed by others; and 3) can consume, produce, and/or interact with streams of user-generated content provided by their connections on the site."*

Datta et al. extended the original definition given by Boyd and Ellison in [88] by defining an OSN as *"an online platform that (1) provides services for a user to build a public profile and to explicitly declare the connection between his or her profile with those of other users; (2) enables a user to share information and content with the chosen users or public; and (3) supports the development and usage of social applications with which the user can interact and collaborate with both friends and strangers"*.

Another definition was given by Schneider et al. [120] according to which *"OSNs form online communities among people with common interests, activities, backgrounds, and/or friendships"* where users can *"[...] upload profiles (text, image, and video) and interact with others in numerous ways"*. Adamic and Adar [121] describe OSNs as services that *"[...] gather information on users' social contacts, construct a large interconnected social network, and reveal to users how they are connected to others in the network."* Pallis et al. [122] define an OSN service as a website that *"acts as a hub for individuals to establish relationships with other persons [...], includes a wide range of tools for people to build a sense of community in an informal and voluntary way, [...] and contain specific components that allow people to: define an online profile, list their connections [...], receive notifications on their activities [...], participate in group or community activities, and control permission, preference, and privacy settings."* Kim et al. [123] define social websites as *"web sites that make it possible for people to form online communities and share user-created contents"*. *People* in this definition can be individuals of a particular organization or are arbitrary Internet users; *communities* may be *"a network of offline friends [...], online acquaintance, or one or more interest groups [...]"*; *content* may be photos, videos, bookmarks of web pages, user profiles, status updates, comments, or posts in a blog or microblog; and *sharing* describes actions such as posting, viewing, commenting, voting on, saving, and re-transmitting content. Kaplan and Haenlein describe OSN services as *"applications that enable users to connect by creating personal information profiles, inviting friends and colleagues to have access to these profiles, and sending e-mails and instant messages between each other"* where *"[...] profiles can include any type of information, including photo, video, audio files, and blogs"* [124]. Finally, Richer and Koch [125] defined OSN services as *"applications systems that offer users functionalities for identity management (1) [...] and enable furthermore to keep in touch (2) with other users"*.

While the existing definitions vary, certain aspects are mentioned in most definitions. These recurring aspects are (1) a public social profile, (2) the ability to create, maintain, and publish a list of connections to other users, (3) means to create and publish content in various forms, and (4) communication with other users.

2.1.2 Classification

Besides defining the essence of social applications and services, researchers also have analyzed how OSN services can be categorized, where a distinction can be made based on functionality and scope of a service or based on it's architecture. To distinguish OSN services from other forms of social web services, Kaplan and Haenlein presented a categorization of *Social Media* services in general based on two criteria, being *social presence/media richness*

Table 2.1: Classification of Social Media by social presence/media richness and self-presentation/self-disclosure as of Kaplan and Haenlein [124]

		Social presence / Media richness		
		Low	Medium	High
Self-presentation /Self-disclosure	High	Blogs	Social networking sites (e.g., Facebook)	Virtual social worlds (e.g., Second Life)
	Low	Collaborative projects (e.g., Wikipedia)	Content communities (e.g., YouTube)	Virtual game worlds (e.g., World of Warcraft)

and *self-presentation/self-disclosure* [124]. As shown in Table 2.1, a high social presence and media richness is found in virtual social worlds or social game worlds, while a low social presence and media richness is found in blogs and collaborative projects such as Wikipedia[1]. According to their classification, OSN sites and other community web services, such as YouTube[2] exhibit medium values of social presence and media richness, where OSN sites distinguish themselves via a high level of self presentation and self disclosure compared to community web services [126].

Heidemann et al. proposed a classification of OSN services based on their respective targeted audiences [36]. Following her classification as depicted in Figure 2.1, OSN platforms can be distinguished based on three criteria being *primary usage*, *focus*, and *access*. While *primary usage* describes whether an OSN service targets private or business-related connections (private vs. business), *focus* describes whether an OSN service focuses on a certain topic (general vs. special interest), such as publishing photos or music. Finally, *access* distinguishes between networks with open registration and networks that restrict access to a specific group of users (open vs. closed) [113].

Other categorizations distinguish OSN services based on their architectures. As the term Online Social Network generally applies to all OSN architectures, OSN services can be distinguished into traditional, centralized OSN (COSN) and

[1]Wikipedia: `https://www.wikipedia.org`. Accessed: 5.9.2017
[2]YouTube: `https://www.youtube.com`. Accessed: 5.9.2017

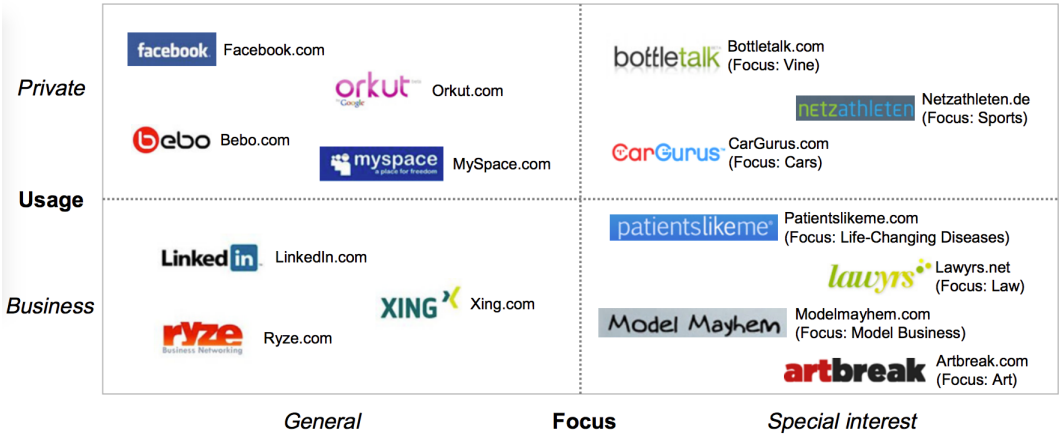


Figure 2.1: Classification of Online Social Networks proposed by Heidemann et al. Image source: [127].

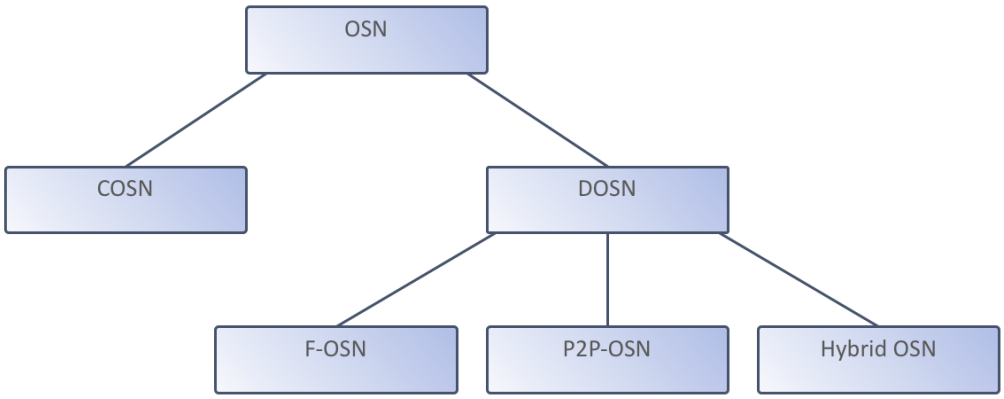


Figure 2.2: Categorization of OSN architectures.

decentralized OSN (DOSN) services [106]. In comparison to COSN services, DOSN services can be further categorized based on their architectural model. Here, Paul et al. note that decentralization of OSN services has more than one dimension and distinguishes between technical and authorial decentralization [111]. Technical authorization describes a distribution of resources, where parts of an OSN service are run on different machines, while authorial decentralization describes that a number of *"distinct and independent authorities run and maintain technical resources"*. According to the definition, a DOSN service is *fully decentralized*, when all basic functionality of a DOSN service does not rely on a centralized component.

To further categorize DOSN services, Paul et al. proposed a classification based on how storage of data is organized in a DOSN service [111]. While peer-to-peer-based OSN services (P2P-OSN) distribute storage and control entirely to individual peers of a p2p overlay network, federated OSN service (F-OSN) rely on multiple interconnected servers. Finally, hybrid solutions (Hybrid-OSN) rely on a mixture of both solutions.

A similar classification was proposed by Koll et al. in [101], which distinguishes between *server-based*, *cooperation-based*, and *hybrid solutions* [101]. According to this classification, distributed server-based architectures are based on permanently available resources to host data, allowing continuous access to all information. This type of architecture usually comprises functionality and data hosted on web servers or cloud services, hence providing high availability. Distributed architectures following the cooperation-based approach solely rely on end-user devices to host functionality and data instead of fixed servers. As user-operated devices usually do not provide high availability and moreover may change their logical and physical location in the network, such architectures need to provide sophisticated mechanisms to ensure data availability and methods for data retrieval. This is usually achieved through (implicit) cooperation of users, for example via self-organizing peer-to-peer-based overlay networks. Finally, hybrid approaches combine architectural elements of both approaches, where data availability and functionality is provided by both permanently available resources and user cooperation. An overview of the distinct categories of ONS architectures is depicted in Figure 2.2.

Chowdhury et al. defined criteria for comparing of peer-to-peer-based DOSN services in [128], which focus on distribution of control and storage in a DOSN service. Inspired by classification of peer-to-peer networks, the proposed organization of both aspects can either be structured, semi-structured, or unstructured. Structured approaches provide a organized overlay, which provides means for routing messages and requests to their destination in a limited number of steps, where the overlay in structured approaches is self-maintained and distributed between all participating entities. In comparison, unstructured approaches lack an overlay for organization of the network. Messages and queries are hence flooded throughout the entire network without a guarantee that a message reaches it's destination. Semi-structured approaches introduce a small subset of super-peers, which organize a sort of overlay or index in the system. Other nodes connect to super-peers, who then are managing the routing of messages and queries.

2.2 OSN Services

In the beginning of the social web, OSN services were organized as central web services. With only a few services available that modeled connections between users, interoperability and exchange of information was not prioritized. Early OSN services such as Friendster or Classmates.com still mostly focused on modeling connections between users, where interoperability issues were not addressed. A comprehensive overview of the market of centralized OSN services including the history of the early social web itself has been described by Boyd and Ellison in [24], Heidemann et al. in [127], as well as by Pallis et al. in [122], where Figure 2.3 depicts the timeline of the foundation of OSN services in the earlier years of the social web. With growing importance of social functionality and services, OSN providers aimed at attracting and keeping users on their platforms. Creating lock-in effects proved to be a valuable asset in the pursuit of this goal and resulted in the isolated islands of OSN services we know today. Facebook, which was founded by Marc Zuckerberg in 2004, changed the landscape of OSN services. With more than two billion monthly active users worldwide [37], the service managed to establish worldwide dominance in the OSN market and continues to grow. The strong network effects made it nearly impossible for competitors to persist in the market. For example Google+, even though implemented and maintained by a large cooperation, couldn't attract a significant number of users and is today expected to be discontinued [76]. Other examples are the discontinued OSN services StudiVZ/MeinVZ [129] as well as Orkut [130].

As of today, relatively few large OSN services manage to prevail on the OSN market, where alternatives to Facebook often focus either on niche markets or managed to acquire regional dominance. For example RenRen, which translates to "everyone's network", managed to attract a high number of Chinese users with design, functionality, and business model strikingly resembling Facebook [131]. While benefiting in its early years from the Internet blockade of non-Chinese services including Facebook and Google, RenRen experienced

a decline in user numbers in the past years, as Chinese users moved towards OSN services of competitors. VKontakte, being a very popular OSN service in Russia, also resembles Facebook's design and functionality to great extends, yet introduces additional and unique functionality such as music and playlists. At the same time, services such as Linkedin or Instagram focused on special niches such as modeling business networks or posting photography.

2.2.1 DOSN Services

Due to the prevalent loose handling of personal information of users by OSN operators and the restraining lock-in effects, distributed OSN services were proposed. As pointed out by Paul et al. in [111], OSN services can be distributed both technologically and authorially. While technological decentralization distributes the various components of an OSN service on different machines, authorial decentralization causes *"distinct and independent authorities [to] run and maintain technical resources [of the service]"*. Paul et al. argue that due to the large numbers of today's major OSN services, most services are technologically decentralized. Yet, within these services, one single entity, usually being the company running the service, holds all power over the entire service. Such OSN services are therefore considered to be of a centralized nature. According to Paul et al., an OSN service is considered to be decentralized, if core functionality of OSN service is supported and furthermore at least one core functionality of OSN services does not rely on centralized architecture. Following their definition, an OSN service is considered to be fully decentralized when all functionality is implemented independent of central control or a central component.

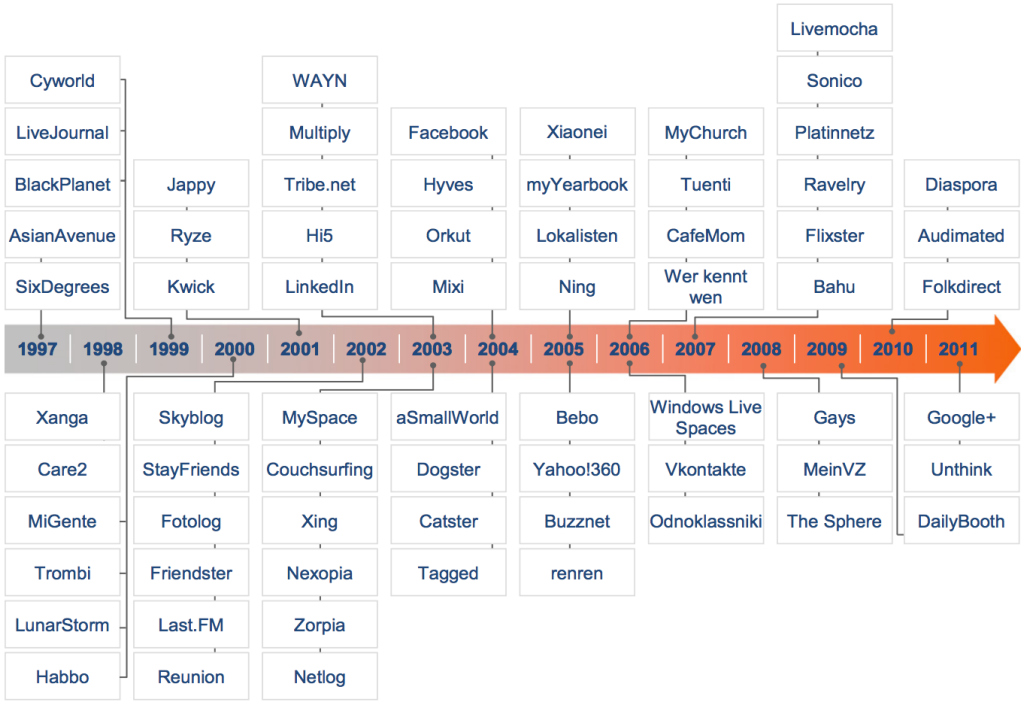


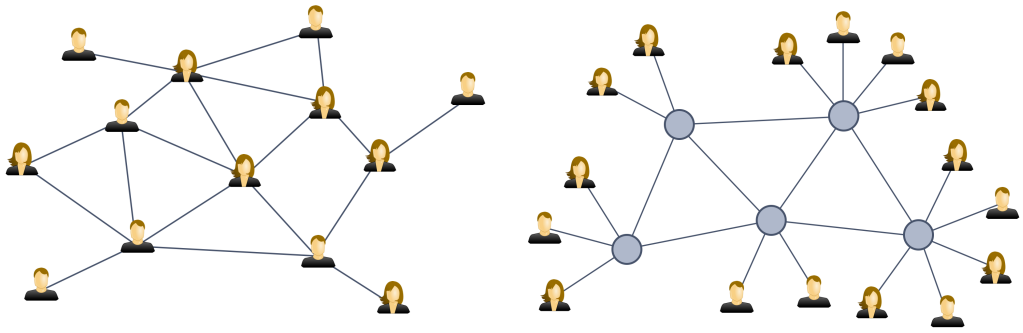
Figure 2.3: Timeline of the foundation of Online Social Networks between 1997 and 2011. Image source: [127].

Based on the categorizations of DOSN architecture models by Paul et al. [111] and Koll et al. [101] presented in Section 2.1, DOSN architecture models can be categorized into peer-to-peer-based OSN services (P2P-OSN) that distribute storage and control entirely to individual peers of a P2P overlay network, federated OSN services (F-OSN) that rely on multiple interconnected servers, and hybrid solutions (Hybrid-OSN) that rely on a mixture of both solutions. In the remainder of this section, an overview of existing solutions based on the categorization of Paul et al. [111] is given.

P2P DOSN Services

As social communication and interaction is by it's own nature decentralized without relying on central entities, researchers aimed to eliminate central components entirely by moving functionality and data storage capabilities to the devices of users, which are connected in a peer-to-peer (P2P) fashion as depicted in Figure 2.4a. Offloading functionality and storage of a OSN service to the individual user's devices introduces several challenges. Not only are such devices usually not running and reachable at all times, smaller devices such as smartphones or tables provide less storage and processing power compared to a regular server. Furthermore, P2P-based architectures introduce potential threats into the P2P-OSN, as all functionality is executed on untrusted devices. P2P-based DOSN services hence face challenges such as availability, efficiency, or trust.

Buchegger et al. proposed hosting user profiles and associated data on user-controlled devices connected in a P2P fashion, thus building a *"peer-to-peer infrastructure that supports the most important features of Online Social Networks in a distributed way"* called PeerSoN [89]. The architecture of PeerSoN comprises two tiers, of which the first tier is a Distributed Hash Table (DHT) based P2P network and the second tier are the nodes representing users. The system uses the DHT for routing and storage for offline messages, while the individual nodes connect to each other directly. Another P2P-based DOSN service, Safebook [95], aims at protecting user and data privacy by replicating data to the nodes of connected users, which are logically organized in shells called *Matroyschkas* around a user, where more direct connections are located closer to the node storing the actual profile. Requests for that data are then routed through the shells surrounding a profile to ensure anonymity and data privacy. The concept was adapted and improved with Proofbook [132], where replication of content is implemented with Blockchain technology [133][134] that ensures authenticity of data through a consensus protocol. LotusNet [135] implements OSN functionality in form of widgets, where data is stored in a DHT. The relationships defined in the social graph are used for access control, where each user specifies types of relationships that can access specific functionality and information. Prometheus [136] implements *social sensors* that collect information about a user's activity in the social network. Data in Prometheus as well as information acquired from social sensors is stored in a DHT, where a implicit meta-graph is constructed that describes connections between users based on their activities and interests. Several other proposed architectures, such as LifeSocial.KOM [137], Decent [138],



(a) P2P-OSN (unstructured) (b) F-OSN

Figure 2.4: Architectures of DOSN services. In comparison to the depicted architectural styles, Hybrid-OSN services combine a mixture of both approaches.

or DiDuSoNet [139], propose approaches which store data objects in a DHT where special replication techniques are applied to ensure uninterrupted availability of content. Many of the proposed approaches rely on asymmetric cryptography to encrypt the stored content and furthermore provide secure communication and access control.

Federated DOSN Services

In order to combine the benefits of reliable and fast servers with the freedom for users to host their social profiles and data at any server, F-OSN architectures were introduced. F-OSN architectures usually rely on a network of loosely-coupled servers, where users can register at any server they trust - or even run a self-hosted instance that connects to the rest of the federation. An example architecture of an FOSN service is depicted in Figure 2.4b. F-OSN services can rely on a better availability and performance compared to P2P-based architectures and were able to attract a large number of users.

Diaspora[3] is a federated OSN service that bases on the three principles decentralization, freedom, and privacy. Diaspora connects servers called *pods* in a loosely coupled federation, where users are free to either choose an existing server or host their own instance. The service allows users to follow each other, where contacts can be organized in so-called *aspects* representing categories. Diaspora is written in Ruby on Rails and implements a broad variety of open standards and protocols for communication between individual pods. Similar to Diaspora, Friendica[4] implements a federated OSN service that allows users to create a user profile on an arbitrary Friendica server instance. While originally built with the use case of microblogging in mind, Friendica evolved past this use case and provides a broad variety of OSN functionality. All Friendica servers are connected in a loosely coupled fashion, where messages between distinct Friendica instances are sent via HTTP. Mastodon[5] is a decentralized microblogging service built with the intention to provide a decentralized microblogging service [140]. Mastodon builds on a federated architecture, where

[3]Diaspora homepage: `https://diasporafoundation.org/`. Accessed: 17.9.2017
[4]Friendica homepage: `http://friendi.ca/`. Accessed: 17.9.2017
[5]Mastodon homepage: `https://mastodon.social/about`. Accessed: 17.9.2017

users are able to register with arbitrary services. Built on open protocols, Mastodon allows users to create follow relationships between each other regardless of the server they are registered on. Messages are then automatically exchanged between the individual Mastodon instances. Several other federated OSN services following similar approaches were introduced, such as GnuSocial[6], HubZilla[7], or Pump.io[8].

While Diaspora, Friendica, and Mastodon are the only DOSN services in operation with a significant number of users, several other, mostly academic, architectures and services have been proposed. Vodafone's initiative OneSocialWeb was driven by the vision of enabling free, open, and decentralized social networking platforms [141]. The project built on XMPP, using XMPP servers to form a federation of social platforms while adapting existing standards such as Activity Streams or vCard. OneSocialWeb was abandoned as of 2011 without a given reason. The initiative drafted a list of standards as XEP extensions [142], which unfortunately were never finalized. For example, Social Relationships [143] was proposed as a data format and protocol to define and publish information on relationships between individuals in a social application or network. The standard was built to allow defining nature and status of a relationship, as well as built-in access control rules. Furthermore, OneSocialWeb proposed standards for Activity Streams and vCard via XMPP, as well as a Personal Eventing Storage for storage and distribution of content[9]. PrPl [144] aims at distributing contents of a user's profile to selected servers and cloud services. The key concept of PrPl is an agent referred to as *butler* that manages and controls all information stored by that user. Users can specify rules for access control, which are enforced by the butler. SoNet [145] is a federated DOSN architecture that addresses user privacy while providing optimal data availability. In SoNet, clients connect to servers of OSN providers in an architecture similar to XMPP. Content is hidden from OSN providers using encryption techniques, while employing replication strategies to compensate for node failures. Finally, SoNet introduces a special pseudonym system, that obfuscates identities of users. Mantle [146] is a DOSN service that implements all data-related functionality on a user's device, where content is off-loaded to arbitrary cloud storage services. The architecture builds on a publish-subscribe model to facilitate interaction between users, which is implemented without any central component. Persona [147] uses servers as data storage for content, where each user has to maintain his own server on which his own content is then stored and published in an encrypted fashion. Servers in Persona also manage read and write requests from other users, where access control lists (ACL) are used to manage access to individual data items, which again are encrypted using attribute based encryption (ABE) [148].

[6]GnuSocial homepage: `https://gnu.io/social/`. Accessed: 8.10.2017
[7]Hubzilla homepage: `https://project.hubzilla.org`. Accessed: 8.10.2017
[8]Pump.io homepage: `https://pump.io`. Accessed: 8.10.2017
[9]OneSocialWeb draft specifications: `http://onesocialweb.org:80/developers-protocol.html`. Accessed: 2.8.2017 via `http://web.archive.org/web/20140927180256/http://onesocialweb.org:80/developers-protocol.html`

Hybrid DOSN Services

In order to combine the advantages of reliable and performant servers with the control provided by P2P-based approaches, hybrid architectures were introduced.

Via-à-Vis [149] employs Virtual Individual Servers (VIS) on cloud services such as Amazon EC2[10] or Microsoft Azure[11], on which OSN profiles and contents are stored. This way, availability of the stored content is ensured, while search and discovery is implemented via a DHT-based P2P lookup service. With the objective to *"[...] preserve monetary incentives for OSN providers"*, Polaris [92] allows users to distribute their personal information and data to arbitrary specialized cloud services, for example photo hosting platforms or microblogging services. This way, content is distributed over a number of content hosting services without impairing their business models. Highly sensitive content is stored at a user's personal mobile device, which is assumed to be able to maintain the content's availability at all times. The authors argue that due to the distribution of content, the overall data privacy is improved. In order to limit unintended disclosure of personal data and the extent of privacy breaches, Vegas [90] limits browsing the social graph to a user's direct connections referred to as the ego network. Vegas is designed as a mobile-based P2P OSN that uses reliable data stores to guarantee availability of content. To ensure data privacy and confidentiality, Vegas employs a public key infrastructure (PKI), where connected users exchange their public keys, while symmetric encryption is used to encrypt the content. Liu et al. propose an architecture for protection of OSN related data called Confidant, where unencrypted personal data is replicated to storage servers of trusted peers based on the assumption that other storage servers are not trustworthy [150]. The proposed architecture focuses on decentralized performant processing of OSN content while preserving data privacy via access control policies. SuperNova [151] connects clients in an unstructured P2P-overlay, in which clients act as storage nodes while organizational functionality is offloaded to super peers. Super peers are selected automatically by the underlying P2P system based on an above-average performance and available resources.

The aforementioned proposals for federated, P2P-based, and hybrid DOSN architectures were designed and implemented with the intention to decentralize functionality and storage of OSN services in order to allow users to keep control over how data is accessed and used. While privacy, confidentiality, and lack of control over one's data have been addressed by the numerous aforementioned architectures and systems, data portability and interoperability are not considered by most approaches. The resulting services, even though they implement decentralization of functionality and data storage, again introduce closed ecosystems, in which users are confined in the respective service. Moreover, the discussed architectures mainly focus on architectural and organizational issues, while the definition of open communication protocols and data formats are mostly omitted or neglected. As a consequence, most

[10] Amazon Elastic Compute Cloud (EC2): `https://aws.amazon.com/ec2/`. Accessed: 17.9.2017
[11] Microsoft Azure: `https://azure.microsoft.com/`. Accessed: 17.9.2017

DOSN services implement native protocols and data formats, which are often not documented. Notably, most proposed solutions are of rather academic nature and fail to attract a significant amount of users as analyzed by Koll et al. in [101].

Solutions to connect different OSN services in a heterogeneous federation has been addressed by only few OSN services, where the scope has been limited to microblogging. A holistic standard for interoperability and interplay of general OSN services covering all important functionality of today's OSN services does not exist.

2.3 Connecting Microblogging Services

[12] The idea of building a federated ecosystem of social web services was originally introduced in the area of microblogging services. Possessing characteristics such as creation of *ambient awareness*, *push-pull communication*, and being a *"platform for virtual exhibitionism and voyeurism"* [126], microblogging services differ from OSN services in the sense as they are mainly focused on publishing and consuming status updates, with less focus on direct communication, managing a social profile page, or creating list of friends and acquaintances. Kaplan and Haenlein distinguish traditional OSN services form microblogging, as OSN services provide a higher 'social presence', being any kind of contact between individuals, and 'media richness', defined as the *"amount of information that can be transmitted in a given time interval"* [124][126].

Early approaches to provide interoperability through open protocols and standards were introduced with StatusNet, a PHP-based software suite that allowed blogs and websites to link to each other and subscribe to feeds, forming a federation of blogging services. This way, new published content as well as updates to already published content are synchronized to all subscribers. The solution based on open protocols and standards, such as OStatus [116], Salmon [152], Microformats [153], XMPP [154], and PubSubHubbub [155], to build a network of microblogging serviced in which publish-subscribe functionality notified subscribed users if a user published new content. The idea of OStatus is that blogs and microblogging accounts require a form of real-time distribution system that automatically routes newly published content to an author's subscribers, where content is encoded using a set of data formats and standards while distribution is handled by the Salmon protocol. This way, content could be shared with users from other servers and services in an automated fashion. Users in OStatus are identified by URIs, from which structured user information can be retrieved using HTTP, including information about a user's feed location and an endpoint for PubSubHubbub.

The underlying Salmon protocol addresses the issue of distributed publishing of content that arises when content is replicated over multiple hosts or services [152]. Content, once replicated to and stored in multiple locations, needs to be kept synchronized when updated. While publishing of updated versions of the original content by the author is relatively simple, decentralized creation of comments and other forms of annotations is rather complex. Salmon addresses

[12] This section has previously been published in [13] ©2017 IEEE.

this issue using a publish-subscription management, where updates for content, comments, and other forms of annotations are managed and distributed in an automated fashion. Salmon introduces the roles of *publisher* and *aggregator*, where aggregators subscribe to publishers. Content created or updated by the publisher is then automatically distributed to all aggregators, resulting in a synchronized version of the content being available from multiple locations. Vice versa, publishers subscribe to all linked aggregators, which forward comments and annotations to the content back to the publisher who then updates the original content, causing the update to be distributed to all existing replicas. Salmon works for publicly available content, yet misses support for privacy and access control.

In 2010, StatusNet merged with two other microblogging platforms, FreeSocial and Laconica [156]. The resulting GNUSocial project, defining itself as *"social communication software"*, maintains and releases the OStatus protocol. Identi.ca, a microblogging platform founded in 2008, originally based on GNUSocial, but switched to Pump.io later, as it was seen as a more progressive standard for decentralized communication in microblogging based on Activity Streams 1.0 and OAuth 1.0. As of today, OStatus is used by several DOSN services such as Diaspora, Friendica, or Mastodon. Yet, as OStatus focuses on distribution of status updates and discussion threads in form of comments, support for most other functionality of today's OSN services is omitted. Using OStatus, users can follow other user accounts by subscribing to other user's status updates using PubSubHubbub [155], retrieve and comment on posted content using the Salmon protocol [152], where content is formatted as Atom Activity Streams [157]. Most use cases and features of traditional OSN services are not covered by OStatus-based solutions, including messaging functionality, support for collections of media items such as photos and videos, or pages.

Approaches similar to OStatus were proposed. Until it's abandonment in 2015, Tent.io [158] aimed at a subscription-based distribution of microblogging content similar to OStatus using Activity Streams and PubSubHubbub. Tent.io proposed an *evented data storage* architecture and data formats for content based on JSON, where the atomic unit for content is called a *post*. Post objects encapsulate published content and comprise metainformation such as the content's type, id, as well as permissions. While Tent.io was designed for distribution of posts in microblogging applications and blogs, it also supported further use cases such as calendar synchronization, polls, or collaborative editing.

Being implemented as a *"stream server that does most of what people really want from a social network"* [159], Pump.io follows an approach similar to OStatus to decentralize microblogging services. The server is written in node.js and is based on Activity Streams 1.0 [114] and follows parts of the Atom Publishing Protocol (APP) [160] with authentication being realized via OAuth 1.0 [161]. Using Activity Streams 1.0 as a container, Pump.io aims at supporting *"almost anything"*, including *"short or long text, bookmarks, images, video, audio, events, [or] geo checkins"* [159] with the intent to allow everyone to host a microblogging profile on his own server. Pump.io's *"API uses REST-ish principles"* and allows to

exchange activity objects between servers and user profiles, where user accounts specify a set of inboxes in form of URLs, to which content can be pushed. Content can then be retrieved via a REST-based API, such as `/api/activity/:id`, where individual objects are addressed via their identifiers.

In the attempt to build a federated OSN service, the *Distributed Friends & Relations Network* (DFRN) was specified as an XML-based protocol. DFRN was designed *"to provide an open and distributed social communication platform with server requirements comparable to that of a typical hosted blog"*, where multiple nodes instead of one central server *"communicate with each other on your behalf"* [162]. Following the federated approach, DFRN-compatible nodes would host user profiles and associated data, where users from arbitrary nodes could connect to each other and access and exchange content using the DFRN protocol. DFRN is based on the Atom Syndication Protocol [163] and provides polling and pushing of content between DFRN compliant nodes. DFRN defines two roles for actors in the protocol, being owner and author. An author is a user who created content such as a status update, where the owner is the user in whose profile the content was posted. Functionality covered by DFRN comprises a profile page, sending and handling of friend requests, a subscription mechanism that notifies subscribers of newly published content, and content retrieval, where *"content [...] can be most anything and is designed to be extensible"*. To identify users across different nodes and domains, identifiers in DFRN are designed in an *"email style"*, comprising the node's domain name and a locally unique user handle. When connecting to another user, DFRN supports sending friend requests to another user, who can accept or reject the request where a friend relationship is created upon acceptance. DFRN interprets friend relationships as granting the respective other user the permission to access one's content, where connections are designed as unidirectional follow relationships. While DFRN generally targets distributed microblogging and social networking, the distributed OSN platform Friendica was built as a reference platform for the protocol [162] which managed to gain some publicity and attracted developers who extended and improved the platform[13]. Besides DFRN, Friendica uses OStatus [116] for dissemination of content and WebFinger for discovery of user profiles. The Friendica API supports access to accounts, messages, favorites, groups, profiles, photos, friends lists, status posts, while aiming to provide compatibility with the GNU Social and the Twitter API[14].

As DFRN as the protocol was considered as too cumbersome and bulky by the Friendica developers, development of an alternative protocol called ZOT was initiated, which was designed to connect different OSN platforms by using publicly available APIs and protocols and building on widely adopted open standards. In 2012, ZOT was discontinued because of tendencies of centralized OSN platforms to apply *"strict limits to outside services accessing their proprietary platforms"*[15] but was reintroduced in 2015 as the federation protocol of the microblogging service Hubzilla [164]. ZOT is based on JSON and implements

[13]Friendica protocol: `https://github.com/friendica/friendica/wiki/Protocol` Accessed: 23.6.2017
[14]Friendica API: `https://github.com/friendica/friendica/wiki/Friendica-API` Accessed: 23.6.2017
[15]The Friendica Blog: `http://friendica.com/node/24`. Accessed: 9.7.2017 via archive.org: `https://web.archive.org/web/20120323163030/http://www.friendica.com/node/24`

an API inspired by REST[16]. Content in Hubzilla is published in *channels*, which can represent an entity of arbitrary type, such as persons, forums, or pages[17]. Individual channels may be then cloned and replicated to multiple hosts in the Hubzilla network in order to improve availability of the content. Updates to a channel's contents are then synchronized with all clones. Authorization in Hubzilla is managed via authentication through proof of possession of a key pair. This way, Hubzilla introduces the concept of *nomadic identities*, allowing users to use their key pair as proof of identity with any server in the network, allowing to publish content in one's channel via any server. Today, ZOT and its reference implementation Hubzilla are still in early stages with parts still unspecified and not implemented.

Communication between pods in Diaspora is realized utilizing a variety of open protocols and standards to interface with other Diaspora pods, including Salmon [152], WebFinger [103], hCard [165], or Magic Signatures [166], where specifically outdated versions of hCard and WebFinger are used [167]. The implementation of the Diaspora protocol mandates to listen for HTTP POST requests on two API endpoints for public and private messages, being `/receive/public` and `/receive/users/:guid` that accept XML-encoded content[18]. Information sent via the simplistic interface can comprise information such as comments, conversations, events, likes, photos, polls, profiles, or status messages.

Mastodon describes an API for accessing user profiles and content, and furthermore facilitates access to a user's own account, allowing simple integration of the service in third party applications[19]. Being based on REST, Mastodon allows other services to retrieve and push content from and to defined endpoints. The API supports accessing and manipulating timelines, individual statuses, and user accounts, as well as publishing media items, creating and editing follow relationships. Furthermore, Mastodon supports a streaming API that allows a client to subscribe to status updates being posted for a specific tag or user, or for the general public timeline.

Based on the experience of OStatus and Pump.io, work on ActivityPub, a novel *"decentralized social networking protocol"* for management and delivery of notifications and content was started [168]. ActivityPub is based on Activity Streams 2.0 and replicates published content to special *inboxes* of a user's followers. Inboxes are OrderedCollections as of the Activity Streams 2.0 specification and are grouped by type of content, being *followers* and *following*, *liked* and *likes*, *public addressing*, and *shares*. This way, the standard aspires to cover microblogging functionality using recent standards and data formats. As of mid 2017, the protocol is considered a W3C Candidate Recommendation and is yet to be finalized and still misses a reference implementation.

[16] ZOT API: `https://project.hubzilla.org/help/en/developer/api_zot`. Accessed: 19.10.2017

[17] Hubzilla Documentation: `https://project.hubzilla.org/help/en/about/about#Glossary`. Accessed: 22.11.2017

[18] The described version of the Diaspora API "*[...] provides the documentation for the future federation protocol for diaspora*. Current diaspora* servers still use an older protocol [...]*" where the current "*Diaspora release 0.6.0.0 and newer has support to receive entities with this protocol, but still sends entities with an older protocol.*" [167]

[19] Mastodon API Documentation: `https://github.com/tootsuite/documentation/tree/master/Using-the-API`. Accessed: 7.8.2017

While several approaches have been proposed and implemented to open up and decentralize OSN services, a truly open and interoperable solution does not exist to this date. P2P-based DOSN architectures mostly exclusively focus on the technical distribution and management of nodes, where protocols and data formats for interconnectivity with other OSN and DOSN approaches are not addressed. In such an architecture, ownership and control over one's data and privacy is given back to the users, yet data portability as well as interoperability with other OSN services is still not supported. The resulting OSN services hence are again isolated islands, in which a technological distribution of users and content is implemented, yet in which users are still locked in.

Similar to P2P-based DOSN architectures, hybrid and federated DOSN service mostly from a lack of support for data portability and interoperability. Even though Friendica implements rudimentary support for manual export and re-import of user profiles, migration of user accounts is limited to OSN platforms of the same service. This way, user profiles can be migrated to a new server and operator, yet causes user identifiers to change, due to the domain-bound nature of user identifiers used by Friendica. While Friendica addresses this issue by notifying all connected OSN profiles about the change of the identifier of the migrated OSN profile, other links, including links from outside services, inevitably break.

While data portability in form of OSN profile migration is not supported by almost all DOSN implementations, interoperability is supported by some federated DOSN services with a limited extent. As Diaspora, Friendica, and Mastodon implement OStatus for dissemination of content, exchange of content is supported for status updates, yet is mostly not implemented for other types of content. For example, viewing photo albums between a Diaspora and a Friendica instance is not possible, while likes for remote content can mostly be accessed. Furthermore, viewing the social profile of another user hosted on another OSN service will redirect. While Friendica redirects the user to the domain of the hosting OSN service, Diaspora only displays basic information about the remote OSN profile.

2.4 Cross-platform Interoperability

[20] In order to evaluate existing solutions for interoperability of OSN services, communication between existing OSN and microblogging services was assessed in a realistic scenario [13]. For this evaluation, user accounts have been created in each of the evaluated OSN services, where freely available services were chosen to create a realistic scenario for interoperability. From each user account, an attempt was made to create a follow relationship with all other OSN services [13]. After all possible connections between the OSN profiles were created, content was posted by each OSN profile with the intent to access it from each of the other OSN services and user accounts. This way, support for interoperability of the supported OSN features between the different OSN service implementations could be assessed. For the evaluation, user accounts were created on OSN servers

[20] This section has previously been published in [13] ©2017 IEEE.

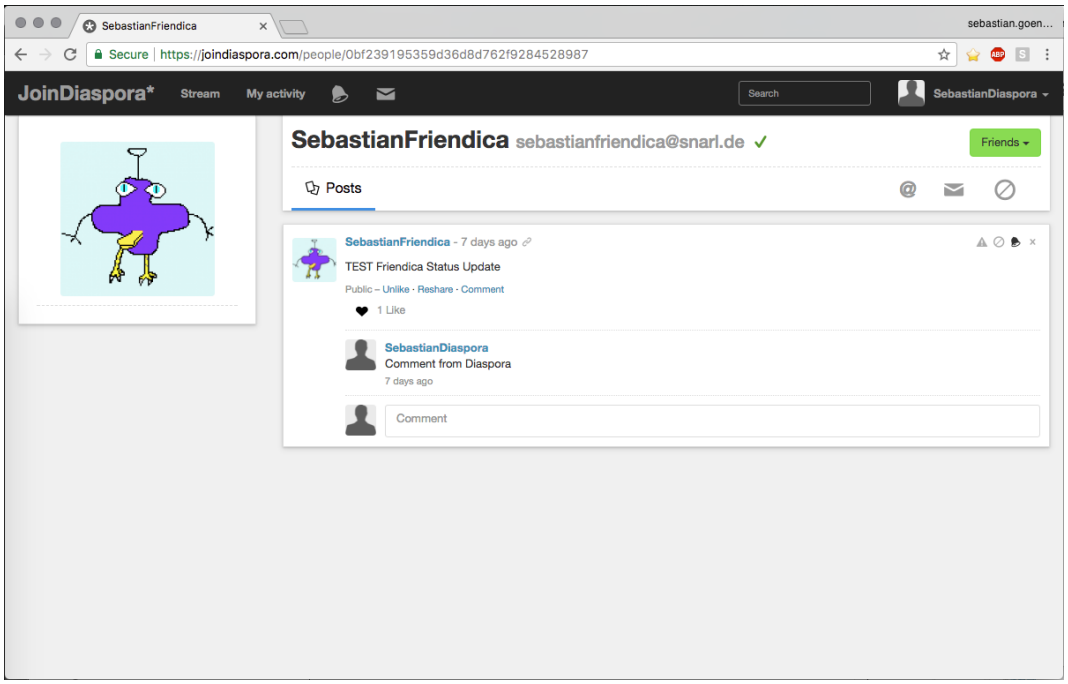
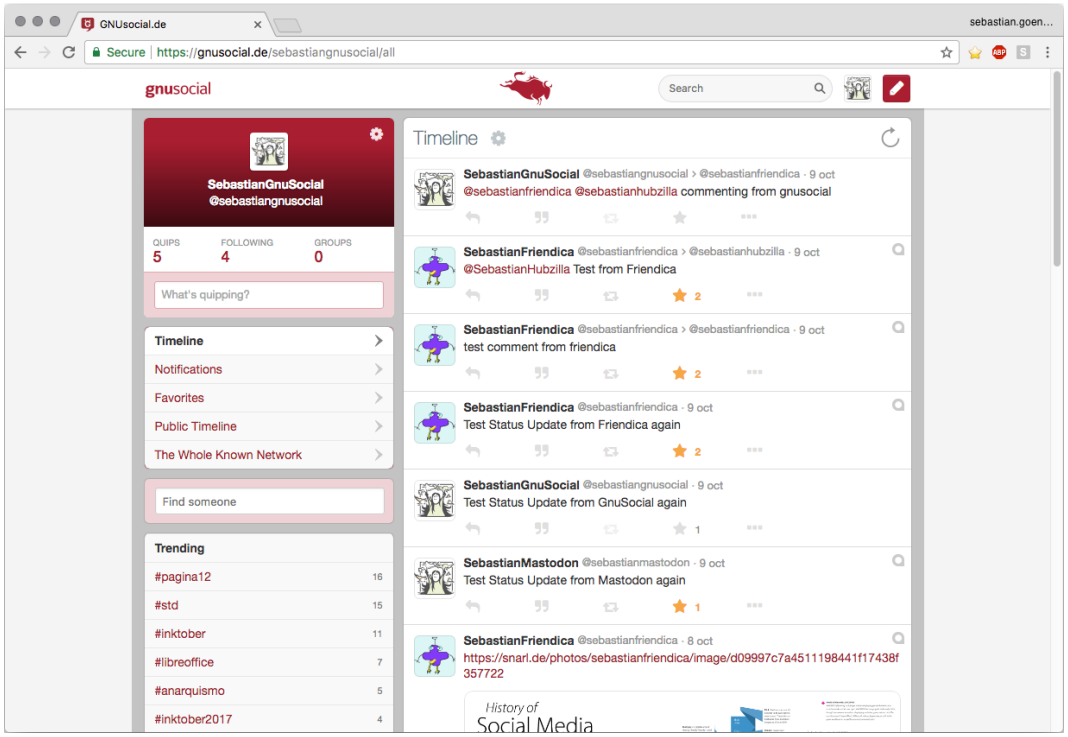


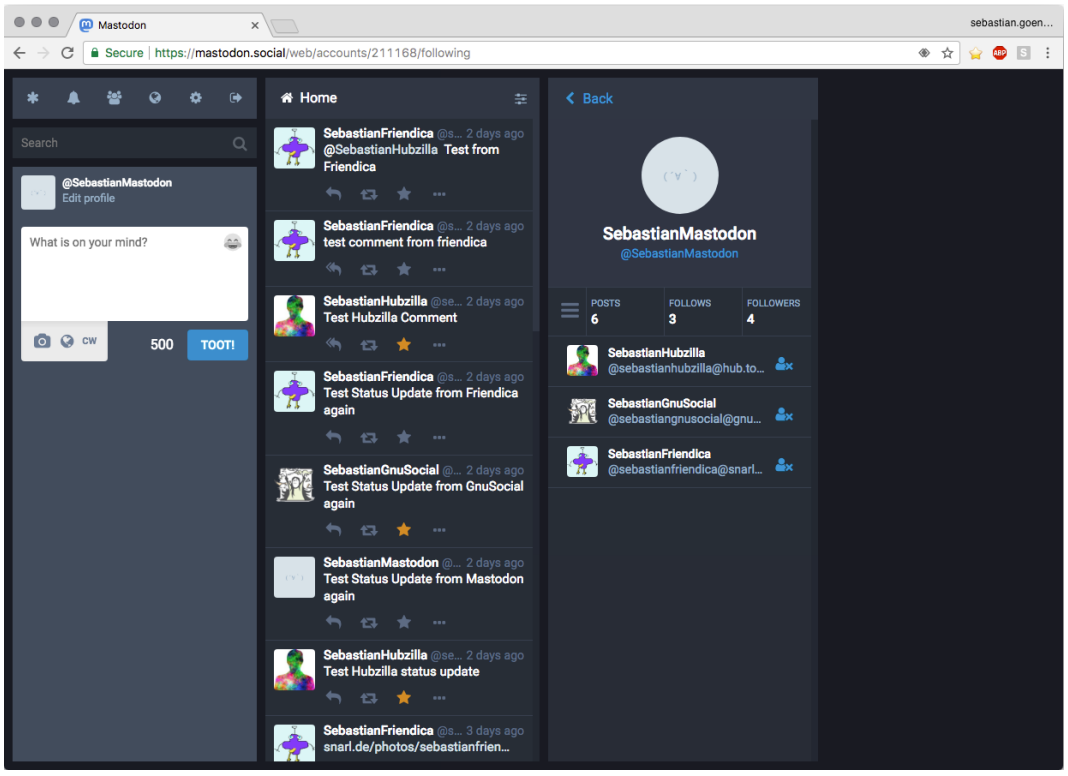
Figure 2.5: Display of a remote Friendica profile in Diaspora. The profile view only shows the username and profile picture. While the remote profile's stream of status updates can be accessed, other content of the remote profile such as photo albums, or friends lists are not accessible. ©2017 IEEE.

that accept public registrations. The services include Diaspora on `joindiaspora.com` running version `0.7.0.1-p6f542522`, Friendica on `snarl.de` running version `3.5.3`, GnuSocial on `gnusocial.com` running version `1.1.3-release`, Mastodon on `mastodon.social` running version `1.6.1`, Pump.io on `datamost.com` running version `4.1.3`, and Hubzilla on `hub.togart.de` running version `2.6.3`. The results of the survey [13] are summarized in Table 2.2.

- **Diaspora** Federation in Diaspora is based on a set of open protocols and standards and allows users to connect to remote Friendica profiles, yet fails to discover and connect to profiles hosted on GnuSocial, Mastodon, or Hubzilla services. Consequently, content created by users of GnuSocial, Mastodon, or Hubzilla can't be accessed by Diaspora accounts, ultimately disallowing any form of communication and interoperability between Diaspora and these service platforms. Diaspora supports accessing Friendica user profiles, yet only shows status updates posted by the remote profile and omits the list of connections and other form of content. As depicted in Figure 2.5 other content posted in a Friendica OSN profile, such as image or video collections, generally cannot be accessed from Diaspora. Status updates posted by Friendica users can be accessed, commented on, and liked by a Diaspora account. Diaspora can also send and receive messages to and from Friendica servers. Finally, Diaspora supports poll functionality, which can only be used by other Diaspora users, while users from other services cannot vote.



(a) Timeline in GnuSocial



(b) Timeline in Mastodon. ©2017 IEEE.

Figure 2.6: Timelines of the microblogging platforms GnuSocial (a) and Mastodon (b). While GnuSocial is able to receive and display status updates from connected users in Friendica and Mastodon, Mastodon is able to access content from user profiles of Friendica, Hubzilla, and GnuSocial.

Table 2.2: Interoperability of selected DOSN and microblogging services. The table shows which service (active network) is able to access what features and content types of other services (target network). For example, a profile created by a Hubzilla user cannot be accessed by a Diaspora user, while a profile created by a Diaspora user can be accessed by a Hubzilla user, where status updates created by the Diaspora account cannot be accessed by a Hubzilla user. ©2017 IEEE.

Active network	Target network	Follow [a]	Profile	Stream	Comment	Like	Message
Diaspora	Friendica	●	◐[b]	●	●	●	●
	GnuSocial	○	○	○	○	○	○
	Mastodon	○	○	○	○	○	○
	Hubzilla	○	○	○	○	○	○
	Pump.io	○	○	○	○	○	○
Friendica	Diaspora	●	◐[c]	●	●	●	●
	GnuSocial	●	◐[c]	●	●	●	○
	Mastodon	●	◐[c]	●	●	●	○
	Hubzilla	●	◐[c]	○	○	○	○
	Pump.io	◐[d]	○	○	○	○	○
GnuSocial	Diaspora	●	◐[b]	○	○	○	○
	Friendica	●	◐[b]	●	●	●	○
	Mastodon	●	◐[b]	●	●	●	○
	Hubzilla	●	◐[b]	○	○	○	○
	Pump.io	○	○	○	○	○	○
Mastodon	Diaspora	○	○	○	○	○	○
	Friendica	●	◐[b]	●	●	○	○
	GnuSocial	●	◐[b]	●	●	○	○
	Hubzilla	●	◐[b]	●	●	○	○
	Pump.io	○	○	○	○	○	○
Hubzilla	Diaspora	●	◐[c]	○	○	○	○
	Friendica	●	◐[c]	○	○	○	○
	GnuSocial	●	◐[c]	●	●	●	○
	Mastodon	●	◐[c]	●	●	●	○
	Pump.io	○	○	○	○	○	○
Pump.io	Diaspora	○	○	○	○	○	○
	Friendica	○	○	○	○	○	○
	GnuSocial	○	○	○	○	○	○
	Mastodon	○	○	○	○	○	○
	Hubzilla	○	○	○	○	○	○

● Content in the target network can be accessed by the active network
◐ Content in the target network can be accessed with limitations
○ Content in the target network can not be accessed

a) To *follow* indicates that the active network is able to create a follow relationship to a user in the targeted network.
b) Only access to a stub profile with username and profile picture.
c) Access to profile only via redirect to remote server.
d) Friendica could detect the Pump.io endpoint, yet couldn't retrieve any information from it.

- **Friendica** Friendica showed the highest degree of interoperability compared to the other services based on OStatus and the DFRN protocol. Following accounts from all other services is supported, yet when accessing a remote social profile, Friendica redirects the user to the server and domain on which the targeted OSN profile is hosted, hence not displaying user profiles in an integrated view. Friendica is able to retrieve status updates, comments on status updates, as well as likes from Diaspora, GnuSocial, and Mastodon, yet fails to access streams of Hubzilla users. Friendica supports creation of photo and video collections, which cannot be accessed by other OSN networks. Additional features, such as starring or disliking status updates, or creating events can only be seen in the Friendica network, but not in any other OSN service. Finally, Friendica supports exchanging messages with other users. Messages sent by a Friendica account can be received by user accounts in Diaspora, but cannot be sent to or received by Mastodon and GnuSocial services, as they both do not support messaging functionality.

- **GnuSocial** GnuSocial, as depicted in Figure 2.6a, is based on OStatus and is able to create follow relationships to all other services, yet is not able to access profile information such as a user's connections. GnuSocial is only able to access status updates, comments, and likes from Friendica and Mastodon users, where access to any form of content from Diaspora and Hubzilla user accounts is not supported.

- **Mastodon** Similar to GnuSocial, Mastodon is based on OStatus and allows to create follow relationships with Friendica, GnuSocial, and Hubzilla, while interoperability with Diaspora servers is not supported. Mastodon shows remote user profiles in an integrated view but is not able to access a remote account's list of connections. As depicted in Figure 2.6b, status updates and comments posted by remote user profiles can be accessed by Mastodon. Surprisingly, likes were not synchronized between services. Finally, Mastodon supports a view of media items published by a user profile but only shows media items published in form of a status update. Collections of images or videos as available in Friendica cannot be accessed. Mastodon and GnuSocial both don't support messaging functionality, yet status updates can be directed at a specific user via a mentioning a username.

- **Hubzilla** Hubzilla implements the ZOT protocol, which showed only limited compatibility with other services. For interoperability, Hubzilla implements the native protocols of OStatus and Diaspora, while advertising that *"basic communications are supported to/from Diaspora, Friendica, GNU-Social, Mastodon"*. The evaluation showed, that while a Hubzilla user is able to create follow relationships with all other networks, yet was only able to access streams, comments, and likes from GnuSocial and Mastodon. Surprisingly, Hubzilla allowed to send messages to connected users of GnuSocial and Mastodon, even though these services do not support private messaging. Subsequently, messages were not received by the recipients in

GnuSocial and Mastodon. Message exchange with Diaspora or Friendica users was not possible.

- **Pump.io** Pump.io implements a REST-based protocol based on Activity Streams and JSON to communicate with other servers and services. After creating a user profile, the service could not connect to any other OSN service, nor could a connection from other remote OSN profiles be established with the Pump.io profile. Friendica as the only service in the survey was able to detect the Pump.io profile's endpoint, but was unable to retrieve any information. As a consequence, communication of any kind with other OSN services was not possible for the Pump.io service.

Even though a number of open protocols and data formats exist for interoperability and communication between different OSN service, the interplay of today's open and distributed OSN platforms presents itself as flawed and error prone. While interoperability between distinct OSN services based on the same protocol suite generally works better with less incompatibilities, interoperability between OSN services based on different protocols and standards shows many inconsistencies ar fails to work entirely. For example, both GnuSocial and Mastodon are based on OStatus and are able to subscribe to each other's content streams, allowing exchange of status updates and comments. Still, likes created by a GnuSocial account are not synchronized to a Mastodon user's stream, while likes from a Mastodon user are synchronized to a GnuSocial user's account without problems. At the same time, interoperability between Diaspora and Friendica services shows a high quality, even though both networks use different protocol suites. One of the possible reasons for the incompatibilities between the surveyed OSN services may be the lack of proper reliable documentation of interfaces and data formats. Some interfaces are not described at all, while others rely on listing API endpoints to which requests should be directed. A lack of a thorough and detailed description of all API endpoints, including data formats, response messages, and communication flows might be the main reason causing problems and incompatibility in communication between different service platforms. Still, the implemented interoperability is limited to the use case of microblogging and thus ignores most of the elemental functionality of today's OSN services [5]. For example, creating photo albums and video collections is a feature supported by Friendica, which cannot be accessed by any other OSN service and is not covered by the protocols evaluated in this analysis. Similarly, Diaspora supports the creation of polls to allow users to vote, where only Diaspora users can access the functionality. The survey [13] shows that a holistic standard for OSN interoperability that covers all functionality of today's OSN services is needed.

3 Concept and Design

With the aspiration of forming an open and heterogeneous ecosystem of loosely coupled OSN services, Sonic provides means allowing existing OSN platforms to connect to each other in a transparent way. This chapter describes requirements and use cases as well as the concept and design of the various building blocks of the Sonic OSNF.

Sonic is based on the idea that social profiles and associated data should be stored on a server of a user's choice, where connectivity and interoperability between servers and profiles is implemented in a transparent manner. It is hence rendered irrelevant whether a social profile is hosted on the same or another OSN platform, allowing users to freely choose an OSN service platform of their liking instead of being forced to sign up with the market leader's OSN service. This allows an open and distributed federation of heterogeneous OSN services to emerge. Users in Sonic are identified by globally unique, domain-agnostic identifiers, which are resolvable by a decentralized directory service. By resolving an identifier via the directory service a user's profile location is retrieved, allowing to access the respective user's social profile and initiate communication. As the identifiers are designed in a domain-agnostic fashion, social profiles can be easily moved between OSN services without connections between users or content objects being broken. In order to allow any OSN service to connect to the federation, a basic set of core features of OSN services is supported. Sonic proposes a set of commonly used interfaces and data formats that allow content being able to be interpreted and used by any OSN service of the federation, where a protocol for feature extensions implements support for arbitrary features to be supported by individual OSN platforms.

The individual concepts and components of Sonic are introduced and described in this chapter. First, a set of use cases is described in Section 3.2 to highlight the benefits of the Sonic approach. Second, requirements for an open and decentralized OSNF are derived in Section 3.3, which are used for the concept and design of the proposed OSN ecosystem. Section 3.4 analyzes functionality of today's most popular OSN services to derive a common set of core features, which are to be supported by the OSNF. Section 3.5 describes the concept and design of the identification of users, followed by a description of the concept and design of the OSNF architecture in Section 3.6, including roles, components, and the general data model. Finally, Section 3.7 describes the protocol for facilitating seamless communication and interoperability between OSN platforms in the OSNF.

3.1 Definitions

In the last decade, the OSN community has created its own lingo to describe activities and assets in OSN services. To prevent ambiguous interpretation of terms, this thesis uses the following terms:

Definition 2: User
In the scope of this thesis, a user is a natural person who uses an OSN service. Each user has a registered user account for an OSN service, in which his OSN profile is stored and managed, which again is identified by a globally unique identifier (GlobalID).

Definition 3: Identity
By registering a user account in an OSN service, a user creates an online identity representing himself, also referred to as online persona. As of the definition of the ISO/EIC specified in [169], the term *identity* describes a set of attributes related to an entity. In the scope of this thesis, a user (entity) maintains an OSN profile (set of attributes) and is uniquely identifiable via a globally unique identifier, the GlobalID (see Section 3.5).

Definition 4: OSN Profile
An OSN profile is the online representation of a user, including all associated data. The OSN profile comprises data such as the social profile page, sent and received messages, status updates, images, or the friends list, but also extends to user account information such as username, password, and cryptographic keys.

Definition 5: Social Profile
A social profile or social profile page is the social homepage of a user, on which he can describe himself. Social profiles usually comprise a profile picture, the user's name or nickname, and other details such as the date of birth, address, or relationship status. Being a representation of the user, the idea of a social profile page is to be accessible for other users depending on the specified access policies.

Definition 6: OSN Service
An OSN service is defined as a specific implementation of OSN. The term hence refers to the general functionality and user interface provided by the implementation, usually providing a specific and unique user experience that distinguishes it from other OSN services. Examples for OSN services are Facebook, Google+, or Diaspora and Friendica in general. While larger OSN services, such as Facebook or Google+, are usually distributed over a larger number of datacenters, the service is provided to the user as a single, monolithic service with a unified, streamlined user experience. Monolithic OSN services such as Facebook are hence still single, centralized OSN services.

Definition 7: OSN Platform
An OSN platform is a specific instance of an OSN service. OSN platforms refer to a single individual installation of an OSN service, which exists independently from other OSN platforms. Examples for OSN platforms are Facebook, Google+, or a single Diaspora pod or Friendica instance.

Definition 8: OSN Client
An OSN client is any application to access the functionality of an OSN platform on behalf of a user. This includes a OSN platform's web page to be used and displayed in a web browser, as well as dedicated smartphone and desktop applications, which access the OSN platform's functionality via special APIs. Finally, OSN clients can also be third party applications that use an OSN platform's API.

3.2 Use Cases

To explain the benefits of the Sonic approach, five use cases are described that point out the shortcomings of closed, proprietary OSN platforms and point out, how an open and heterogeneous OSNF can help to solve them. In the following use case scenarios, four individuals *Alice*, *Bob*, *Charlie*, and *Dave* are considered, who want to keep in touch with each other using an OSN service. As depicted in Figure 3.1, *Alice* and *Bob* already signed up with an OSN P_A, while *Charlie* is using a separate OSN platform P_B because most of his other friends use it too. Both platforms P_A and P_B are popular and easy to use, but use their customers' personal data for targeted advertisements. *Dave* has always been a little hesitant when it comes to using OSN services in general, as he is very concerned about his privacy. Hence, he has not registered with any OSN platform yet and tends to keep in touch with his friends and relatives via email and phone to avoid giving his personal data away.

3.2.1 Use Case 1: Signing Up

As *Alice* and *Bob* are using the OSN platform P_A and *Charlie* uses a separate OSN platform P_B, *Dave* is forced to either sign up with both platforms P_A and P_B, or not be able to connect to either *Alice* and *Bob* on platform *A*, or *Charlie* on platform P_B. None of the options is desirable, as they either require him to give his personal information to two platform providers, or not connect to some of his friends. *Dave* would prefer a separate OSN platform P_C, which is known for not sharing their customers' data with advertisers. In the given scenario, *Dave* is forced to sign up with platform P_A, allowing him to connect to his friends *Alice* and *Bob*, but not *Charlie*, whose profile is hosted on platform P_B.

In the Sonic OSNF, it is irrelevant on which platform a user registers, as connections to other users can be created across platform borders. Here, *Dave* could have signed up with his preferred OSN platform P_C, while connecting to all of his friends on both of the platforms P_A and P_B. Sonic promotes seamless connectivity between social profiles on all compatible OSN platforms,

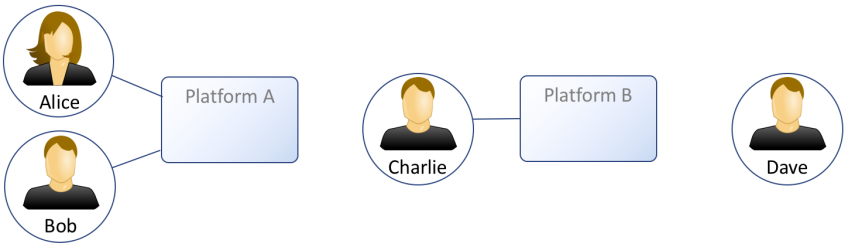


Figure 3.1: Initial situation for the described use cases: Users *Alice* and *Bob* are using OSN platform P_A, user *Charlie* is using OSN platform P_B. User *Dave* is not registered with any OSN platform.

thus rendering borders between different platforms irrelevant and transparent to users. For users, connecting to a friend whose social profile is hosted on a different OSN platform works exactly the same as if it was hosted on the same platform.

3.2.2 Use Case 2: Multiple Social Profiles

To alleviate the situation from Use Case 3.2.1 where *Charlie* cannot stay in contact with his friends *Alice*, *Bob*, and *Dave*, *Charlie* also registers with the OSN platform P_A and connects to his friends. Now, all four users have a social profile on platform P_A, but as *Charlie* wants to stay in contact with his other friends and associates on platform P_B, he keeps his user account and social profile there. Hence, he now has to manage two separate social profiles in two separate OSNs. In case *Charlie* wants to share some content with his friends in both OSN platforms, he has to publish it twice. This results in some overhead for him as he wants to only have one online persona.

In the Sonic OSNF, a user only manages one social profile and may connect to other users on any OSN platform in the OSN ecosystem. As users are connected across OSN platform borders, creating and managing more than one profile is possible, but not necessary. In case a user wants to have more than one social profile, which are separated from each other, different identities can be used. This might be desirable for different contexts of usage, for example for having a personal profile and one for a professional, work-related context.

3.2.3 Use Case 3: Inconsistencies with Posting and Commenting

To prevent inconsistencies between his two separate profiles, *Dave* installs an integrator application that connects to both of his social profiles on the platforms P_A and P_B. The integrator application fetches all news from all connected social profiles, and displays them in one view. Vice versa, posts of *Dave* are automatically published by the integrator application in all connected profiles. If a friend of *Dave*, for example *Alice*, now happens to comment on a posting of *Dave*, this comment will be automatically shown to him in his integrated view, but will not be published in any other connected social profiles. In case *Dave* replies to the comment with another comment, his reply will be published in all his connected social profiles by the integrator. However, as *Alice's* comment has

only been published in the OSN she is using, the list of comments in all of *Dave's* other OSN profiles are missing her comments and hence may lose their meaning, assuming the content of the comments is referencing one another.

In the Sonic OSNF, the comments of all users are received by the user's OSN platform that created it in the first place. As all comments are stored in one location, inconsistent representations of the content is prevented. Hence, users from all connected OSN platforms will receive the same content.

3.2.4 Use Case 4: Event Management

Alice wants to organize a meeting with her friends and manage the meeting using her OSN platform. She creates the event and starts inviting people. Unfortunately, she can only invite people from the same OSN platform. As of this, she has to create a separate event in all connected OSNs in which she wants to invite people. As a consequence, the same event needs to be maintained in multiple platforms, while guest lists and discussion boards cannot be synchronized. Hence, guests invited on platform P_A cannot see guests and posts from other platforms and vice versa.

In the Sonic OSNF, event support can be implemented using a feature extension. Here, the event organizer's OSN platform manages the event data, which can be accessed by all attendees.

3.2.5 Use Case 5: Data Portability

In order to stay in touch with his friends, *Charlie* has set up a second OSN profile on platform P_A in addition to his main profile on platform P_B. As more and more friends of *Charlie* setup a social profile at platform P_A, his original profile at platform P_B becomes less and less relevant. *Charlie* hence decides to abandon his profile at platform P_B and start using his new profile at platform P_A exclusively. Yet, *Charlie's* profile at platform P_B stores information such as photo albums, exchanged messages, and status updates he wants to keep, as they were created and curated over a long period. To move his profile data to platform P_A, *Charlie* has to extract all data manually, including texts, images, and friend connections. At platform P_B, he has to manually insert the data into his new OSN profile. While this is possible for some content, other content cannot be inserted. For example, it is possible for *Charlie* to create a new photo album and upload the images to platform P_A, but messages he exchanged with his friends cannot be 'inserted' into his new profile. The content therefore is lost. Furthermore, links to content of other profiles as well as links to his profile data are broken, as *Charlie's* user identifier changed due to abandoning his old OSN profile at platform P_B.

In the Sonic OSNF, *Charlie* can simply create a new OSN profile on a new platform and migrate all data automatically to the new profile location. Data is formatted using a common standard, so all data is kept and can be used in in the new OSN profile at platform P_A. Furthermore, *Charlie's* user identifier remains unchanged, so links from and to his OSN profile are kept intact.

3.3 Requirements

To address the shortcomings and drawbacks of current centralized and decentralized OSN platforms as described in Section 1, the following architectural significant requirements (ASR) [170] have been derived and identified as prerequisites for an open and heterogeneous OSN ecosystem [3]. In software architecture design, an architectural significant requirement is a requirement that determines and shapes an architecture [170]. The following non-functional requirements therefore describe specifications for the overall architecture of the OSNF and its components and thus define the foundation for the OSNF and its components.

Requirement R1: Non-Intrusive Design

Sonic follows the idea of building a federation of existing OSN platforms to create an open and heterogeneous OSN ecosystem. Assuming the general willingness of operators of existing OSN platforms to open their services to such a federation, requirements to alter core parts of an existing OSN platform implementation will most likely repel platform operators or at least hinder gaining their acceptance of the approach. Hence, the overhead of implementing the required protocols and functionalities into an existing OSN platform implementation should be kept to a minimum with as little required adaptations as possible. Most importantly, implementing the proposed solution must allow OSN platforms to provide an unchanged user experience to their customers, which is in many cases used as a trademark of the OSN.

Requirement R2: Platform Independent Social Personas

Social profiles can be interpreted as online personas, through which users interact with websites and each other. With the web becoming more and more interconnected and social, these kind of social identities have become an integral part of how content is discovered, consumed, and shared. As of today, these social identities and profiles are bound to the OSN platform they were created in. Having access to a person's social persona hence allows to extract and derive highly sensitive personal information about it's owner. An example for this are Facebook identities, where visits to arbitrary websites are tracked by Facebook through Like buttons [171]. Social personas being tethered to the OSN platforms ultimately binds users to the OSN platform they signed up with. Hence, user identities as well as the social profiles, including all associated data, must be independent of the OSN service and platform they were created on.

Requirement R3: Decentralized and Federated Architecture

To address the issue of walled gardens and lock-in effects in OSN platforms, users need to be able to chose between a variety of interconnected OSN platforms and operators, thus not limiting users to a single platform based on their choice. A decentralized, federated architecture allows users to chose a provider they trust and moreover select OSN platform implementations that match their personal needs and preferences, or even implement and host their own OSN platform. This contributes to a highly diverse OSN ecosystem. The architecture of Sonic must

hence provide loose coupling of OSN platforms without a central component that is able to control communication in the federation. Furthermore, the architecture must allow scaling to a large number of users.

Requirement R4: Distributed Control and Management

Centralized OSN providers are often confronted with the accusation of misusing their power that originates from being in full control of some or all central components and functionalities, which are crucial for an OSN service's operation. This is a result of intentionally creating lock-in effects through usage of proprietary protocols and data formats as well as using the users' personal information to create revenue, for example through targeted advertisement. As of this, introducing a central entity in an OSN federation should be avoided as it would allow operators of this entity to control the entire federation. The proposed solution should not be dependent on any central entity that could interrupt or impair the functioning of entire federation or parts of it.

Requirement R5: Users Control their Data

OSN operators are often criticized for using their customers' personal profile data for targeted advertisement or selling their customers' profiles to companies. Users of such platforms are mostly not informed about what data is actually used for which purposes or to which third parties information is disclosed while not allowing them to control which data is used for what or accessed by whom. By offering users the choice of which OSN platform and operator to entrust their personal data to and who gains access to which parts of a social profile, users are allowed to exert full control over where their personal data is stored and who has access to what parts of it.

Requirement R6: Seamless Communication and Interoperability

One of the major drawbacks of today's OSN landscape is that users cannot communicate freely with users of different OSN platforms and services. Being mediums for social interaction and communication, OSN platforms should allow seamless communication between and interperability with each other [35] [5]. Platform borders should be entirely transparent to users so that it is rendered irrelevant whether a connected user's social profile is hosted on the same OSN platform or not.

Requirement R7: Open Protocols and Data Formats

Not only is *"a lack of interoperability among products and services [...] bad for competition and innovation"* [60], but also does the lack of common standards, protocols, and data formats create barriers between OSN platforms that hinder users to communicate freely. Hence, open protocols and data formats are needed to facilitate seamless communication and interoperability between OSN platforms [35]. This builds a foundation for an open OSN ecosystem, in which data and messages can be exchanged seamlessly. The proposed solution must hence introduce transparent interoperability and connectivity between OSN platforms. At the same time OSN platform implementations must be able to integrate the solution with minimal overhead while remaining independent in

their design choices regarding to business logic and user interfaces at the same time.

Requirement R8: Migration of OSN Profiles

Migration describes data portability of OSN profiles, being the possibility to move entire user accounts from one OSN platform to another without losing any kind of data or connection to other users or social profiles [4]. Support for data portability in combination with interoperability in OSN networks, as described in Requirement R6, ultimately erases lock-in effects. As stated by Graef [35], migration allows users of any OSN platform to migrate to a competitors platform at any time in case they become dissatisfied with the services of their current platform or when a competing service offers a more appealing quality of experience. This would give rise to a shift of power from OSN platform operators to their customers, as an unsatisfied user will not keep using a service if alternatives exist that satisfy his needs better. Hence, this would contribute to more competition among OSN platform operators, most likely resulting in more innovation of existing services.

Requirement R9: Singular Social Profiles

Users of OSN services should not be forced to maintain and manage multiple social profiles when attempting to stay connected to other users on multiple different OSN platforms. Doing so aggravates the impact on data privacy related issues, as personal profile information is not only stored at one, but on multiple OSN platforms. Furthermore, maintaining multiple social profiles may result in inconsistencies between the different social profiles. While in some occasions such differences may be desirable for the user, having to keep multiple social profiles updated is a cumbersome task and often results in outdated information on social profiles. OSN integrators, such as SocialDeskApp[1], are realized via applications or plug-ins and require a separate user account and social profile in each connected OSN platform. The integrator then synchronizes the data with the connected OSN platforms, resulting in an unnecessary replication of all user information across multiple platforms.

Requirement R10: Global User Identification

In an open and decentralized OSN ecosystem that allows not only seamless and transparent interoperability across platform borders, but also migration of entire user accounts, user identification must allow continuously resolving user identifiers to the current location of their user accounts and social profiles. Moreover, user identifiers should remain stable and unchanged even after migrating a profile. As current user identifiers are either built for single platform purposes and hence depend directly on the respective OSN platform implementation, or depend on the domain name of the platform that assigned the identifier, new means of identification for an open and decentralized OSN ecosystem need to be implemented [7]. The proposed solution hence needs to support globally unique, domain-independent identifiers, which can be resolved to the respective user's profile location.

[1] SocialDeskApp: `http://socialdeskapp.com/`. Accessed: 25.5.2017

Requirement R11: Extensibility

OSN services evolve and adapt to trends and new technologies to provide a state-of-the-art user experience for their customers. With changing functionality, static OSN solutions that fail to adapt to changes in customer demands will most likely be unable to succeed and will ultimately be superseded by more flexible systems or solutions. Built-in support for evolving functionality is hence a feature of great importance for an architecture of an OSNF to allow OSN services to continuously extend and improve their services.

3.4 A Taxonomy of Featuresets of Online Social Networks

Following the idea of an open and heterogeneous OSN ecosystem, in which a federation of loosely coupled OSN platforms allows users to host their social profile at any OSN platform they choose without being disconnected from their friends or acquaintances using other OSN platforms, means to exchange information between different OSN platform implementations in a common fashion are of paramount importance. As of today, a high number of OSN services exist that differ in scope and functionality [24], making a mapping of functionality between them a seemingly complicated task. Still, even though today's OSN platforms differ to great extents in appearance, user interfaces, architecture, functionalities, or API model, one can identify recurring 'default' features that are semantically equivalent or at least similar. For a common standard to be able to connect platforms, different implementations of features need to be mapped to each other in a standardized fashion. Unfortunately to this date, no extensive survey exists that analyzes the functionality of OSN services that facilitates communication and general information exchange between user profiles. In this section, a survey of OSN functionality is presented that analyzes the features of different OSN services and derives a common taxonomy of OSN functionality. Using the OSN feature taxonomy, functionality of varying OSN platforms are made comparable, thus allowing for a detailed comparison of featuresets of different OSN platforms. From the results of this survey, a common set of default features, which is commonly supported by today's most popular OSN platforms, is derived. This core of OSN functionality, the *core OSN featureset*, is then later used in the design and architecture of the Sonic ecosystem in Sections 3.6 and 3.7, thus ensuring that the presented solution is compatible with commonly supported features of today's OSN platforms.

3.4.1 Related Work

In the pursuit of analysis of social services and platforms, researchers derived several definitions of the term "Online Social Network". Most definitions include a more or less abstract definition of OSN functionality, but lack a clear description and analysis of the individual supported features.

As defined by Boyd, *"social network sites are based around profiles, a form of individual [or group] home page, which offers a description of each member"*. Boyd added that social profiles comprise a list of friends and content such as text, images, video, and comments [172]. In [24], Boyd and Ellison give another, rather abstract, definition of OSNs, in which they derive a general featureset of social networking sites comprising the ability of users to create *semi-public profiles*, model *connections to other users*, and *traverse this list of connections*. The work further lists standard functionality being *profiles*, *friends*, *comments*, and *private messaging*. Boyd and Ellison refined their definition in [119], stating that an OSN is a *"[...] networked communication platform [with] uniquely identifiable profiles, [...] connections, and [...] streams of user-generated content [...]"*.

Basic functionalities of OSN platforms have also been defined by Heidemann et al., giving an abstract definition of core functionalities of OSN services [127]. The classification lists *personalized user profiles*, topics of *interest*, e.g. in form of special groups one can subscribe to, and *personal contacts*, being lists of friends or acquaintances.

A more detailed description was given by Richter and Koch [125], who argue that OSN services offer six basic functionalities for their users, which are specializations of two fundamental categories of OSN functionality, *keeping in touch* and *identity management*. The six basic functionalities named by Richter and Koch are *identity management*, *expert search*, *context awareness*, *contact management*, *network awareness*, and *exchange*. Identity management in this context describes the construction of a social profile of a user that can be viewed by others. The social profile is used for presenting or staging oneself for a particular audience or task, while configuring access permissions regarding to what users are allowed to access which part of the social profile information. Expert search comprises functionality that allows a user to both search for information by certain criteria, such as name or location, as well as proactively receiving contact recommendations from the OSN service. Context awareness describes common context with other users, indicating what connects two individuals or a group. This could be a common employer one has worked for in the past, the city one is living in, or interest in a certain kind of music. Contact management describes the management of connections to other users, for example the list of friends or acquaintances, but also extends to tagging people and managing access restrictions to one's social profile. Network awareness describes support for indirect communication, for example in form of news feeds. This allows users to be aware of posts or events of their contacts. Finally, exchange describes means to communicate directly with other users, for example in form of text messages.

A similar classification of general features of social networks was defined by Rohani and Hock in [173]. Their classification lists seven general features, some of them comprising several functionalities. The list specified in [173] comprises a *personal profile*, basically being a homepage with various information about the user, and a *personal bill board* of content a user published. Furthermore, users can model connections and relationships to other users in a *friendship network*, while staying in touch via the exchange of messages or IP-based phone calls (*communication with online connections*). The classification also describes the functionality of a *thinking room*, being a user's ability to interact with content on the OSN. This comprises the ability to leave comments for content, voting for content for example by leaving reviews in form of a one to five star rating, publicly marking content as a favorite, or flagging it as inappropriate. *Forums*, which are described as functionality to form user groups for certain topics, allow users to publish content to a specific group of users. In Rohani and Hock's definition, only members can publish content in a forum, yet everyone can view it. Finally, the functionality of *e-newsletters* describes a recommendation of content to be published by the user, where the recommendation is generated by the OSN service.

Kneidinger grouped OSN functionality into three categories, being tools for *communication*, *entertainment*, and *presentation*, where common functionality of OSN services is characterized as having a *profile* for self-representation, adding *friends* to one's network, *observe and comment on activities* of friends, and using *communication* chat tools [174].

Schneider et al. [120] described features of OSN services. The list of features comprises managing *profiles* including content in form of *text*, *images*, and *video*, maintaining a *list of friends*, joining and participating in *groups and networks*, write on other user's *walls*, *communicate* via internal messaging services, and use *third-party applications*.

Paul et al. adapted the definition given by Boyd and Ellison in [24]. They defined an OSN as a platform that allows users to build public profiles, explicitly model connections to other users, share information and content publicly or addressed to specific users, and allow connectivity for social applications for interaction and collaboration. Furthermore, Paul et al. specified a list of *networking functions*, comprising a profile, relationship management, applications, a newsfeed for posting pictures, videos, or hyperlinks, tagging users, liking or disliking content, commenting, messaging, and groups [106]. In [111], they derived basic and extended functionality of OSN services from their earlier definition. In their definition, basic functionality comprises *profile management*, *relationship handling*, and *interaction*, where interaction can be direct or indirect. Here, direct interaction defines a messaging system (1:1 communication) and indirect interaction defines sharing of content with multiple users (1:n). Extended functionality comprises an *API* for third party applications to connect, *search* functionality for users to search for content or other users, a *recommender system* that allows users to be recommended as friends, and a *social network connector* that connects different OSN services, for example in form of integrator plugins.

Kietzmann et al. defined seven building blocks of social media, being identity, conversations, sharing, presence, relationships, reputation, and groups [175]. *Identity*, being the core functionality, comprises disclosure of personal information in a social profile, while *relationships* cover users establishing a network of friends by adding each other in lists. *Reputation* allows to evaluate a user's trustworthiness or social standing in the OSN service, for example with *endorsements*, *likes*, or *ratings*. *Sharing* describes the ability of users to create and publish content in a form of feed, *conversations* are defined by direct message exchange with other users, and *presence* allows users to be informed about other users' current status or whereabouts. Finally, *groups* allow users to bundle the sheer amount of content published on OSN services by topic.

For creating an ontology of OSN functionality for the SocIoS project, Tserpes et al. derived a list of common OSN functionality. The resulting vocabulary is adapted from the data models of OpenSocial and comprise *person*, *media item*, *activity*, *event*, *message*, *rating*, and *group* [176].

In their definition of OSN services, Kim et al. [123] listed functionality of OSN services being *people*, *communities*, *photos*, *videos*, *bookmarks of web pages*, *user profiles*, *user's activity updates*, *comments*, *sharing*, and *voting*.

Table 3.1: Comparison of definitions of OSN features given in literature.

Definitions	Profile	Friends	Feed	Comment	Tag	Like & Dislike	Image & Video	Galleries	Messaging	Groups	Events	Vote	Review	Presence	Endorse	Search	Recommendation	3rd Party Apps
Boyd & Ellison [24][119]	●	●	●	●	○	○	●	○	●	○	○	○	○	○	○	○	○	○
Heidemann [127]	●	●	○	○	○	○	○	○	○	●	○	○	○	○	○	○	○	○
Schneider et al. [120]	●	●	●	○	○	○	●	○	●	●	○	○	○	○	○	○	○	●
Richter and Koch [125]	●	●	●	○	●	○	○	○	●	○	○	○	○	○	○	●	○	○
Rohani and Hock [173]	●	●	●	●	○	●	○	○	●	●	○	●	●	○	○	○	○	○
Kneidinger [174]	●	●	●	●	○	○	○	○	●	○	○	○	○	○	○	○	○	○
Paul et al. [106][111]	●	●	●	●	●	●	●	●	●	●	○	○	○	○	○	●	●	●
Kietzmann et al. [175]	●	●	●	○	○	●	○	○	●	●	○	○	●	●	●	○	○	○
Tserpes et al. [176]	●	○	○	○	○	○	●	○	●	●	●	○	●	○	○	○	○	○
Kim et al. [123]	●	●	●	●	○	○	●	○	●	●	○	●	○	○	○	○	○	○
Europ. Commission [62]	●	●	●	●	○	●	●	○	●	○	○	○	○	○	○	○	●	○

When deciding upon whether Facebook and WhatsApp should be allowed to merge, the European Commission defined a list of functionalities of OSN services. According to *Case No. COMP/M.7217 - Facebook/WhatsApp* [62], essential functionalities of OSN services comprise a public or semi-public profile and a list of friends or contacts. Furthermore, messaging, comments, recommending friends, and posting content such as pictures, videos, or links are recognized as important functions of OSN services, giving users the ability *"to indicate their interests, activities or life events, create photo albums and express opinions on other users' postings (for example, by commenting or 'liking')"*. Finally, the document published by European Commission states that not all of these features have to be implemented by a service to be qualified as an OSN.

As shown in Table 3.1, existing definitions of OSN functionality differ to great extents. Moreover, functionality such as liking, tagging, or managing photos are not covered by most definitions, even though being commonplace in almost every existing OSN service as of today. Finally, many of today's popular OSN features, such as event management or support for pages, are not covered by most of the existing definitions. To determine which functionality is supported by OSN services, a more comprehensive definition is needed.

3.4.2 OSN Features

While functionality of OSN services changed and evolved since SixDegrees.com was launched in 1997 [24], the main functionality remained mostly unchanged. According to the various definitions of OSN functionality discussed in Section 3.4.1, OSN services feature user profiles, allow users to establish links between each other, publish content, and exchange private messages.

In order to define a common standard for seamless interaction of various diverse OSN platforms, supported functionalities and features of different platforms need to be mapped to one another. Even though some OSN platforms differ to great extents in appearance, user interfaces, architecture, functionalities, or API models, one can identify recurring default features that are semantically equivalent or at least similar. Hence, a common set of OSN functionality, the *core featureset*, can be derived.

In the scope of allowing different OSN services to seamlessly connect to each other, only functionality related to interaction with profiles or information of other social profiles is relevant. As OSN services implement a high variety of functions that just affects user interface, app integration, or user experience, the following definition is used to clarify that only functionality related to interaction between OSN profiles is considered.

Definition 9: OSN Feature
An *OSN feature* in an OSN service is functionality that enables a user to directly interact with content of another user's OSN profile by means of creating, reading, updating, or deleting data.

For example, a social profile page is accessible for other users and is therefore considered to be an OSN feature. While at the same time, photo editing, games, recommendations, search, or displaying notifications to the user are not considered to be OSN features in the sense of this definition. The combination of all OSN features of an OSN platform describes its functionality, defined as the OSN featureset.

Definition 10: OSN Featureset
The combination of all OSN features implemented by an OSN platform is defined as its *OSN featureset*.

Finally, the OSN core featureset represents what OSN features are implemented and supported by the majority of the most popular OSN platforms.

Definition 11: OSN Core Featureset
The combination of all OSN features present in the majority of major OSN platforms define the *OSN core featureset*.

3.4.3 Analyzed OSN Services

For the analysis of OSN featuresets[2], today's most popular OSN services are considered [178]. As popularity of OSN services varies geographically and changes over time [177], OSN services with only regional importance are considered as well (see Figure 3.2). The Social Media Update 2016 published by the Pew Research Center lists the most popular OSN platforms in the USA being Facebook, Instagram, Pinterest, LinkedIn, and Twitter. According to the study, Facebook is used by 79% of online US American adults, Instagram by 32%, Pinterest by 31%, LinkedIn by 29%, and Twitter by 24% [38]. A study by Ofcom analyzed popularity of OSN services by country [16]. As shown in Figure 3.3, Facebook is the most popular OSN service in all surveyed countries, while the popularity of Twitter, LinkedIn, Google+, and Pinterest varies. Based on these findings, the OSN services Facebook, Google+, Twitter, Linkedin, Instagram, and Pinterest are analyzed. Furthermore, the popular German career-network Xing is considered. In Russia and several former soviet states such as Belarus, Kazakhstan, Estonia, Kyrgyzstan, Moldova, Ukraine and Latvia, the OSN service VKontakte is the most popular site according to Alexa Internet Ranking [179]. For China, the very popular OSN service RenRen is considered being one of the "*most popular, most open and best-financed social network sites*" in China [180]. Finally, to also consider distributed OSN services, Diaspora, Friendica, and the relatively new distributed OSN service Mastodon are analyzed as well. All three

[2]For the survey, the operational websites and – if available – developer platforms of the surveyed OSN services as of August 2017 were investigated. As OSN service operators constantly implement new features to improve product quality and user experience, the results of this survey may need to be updated in the future.

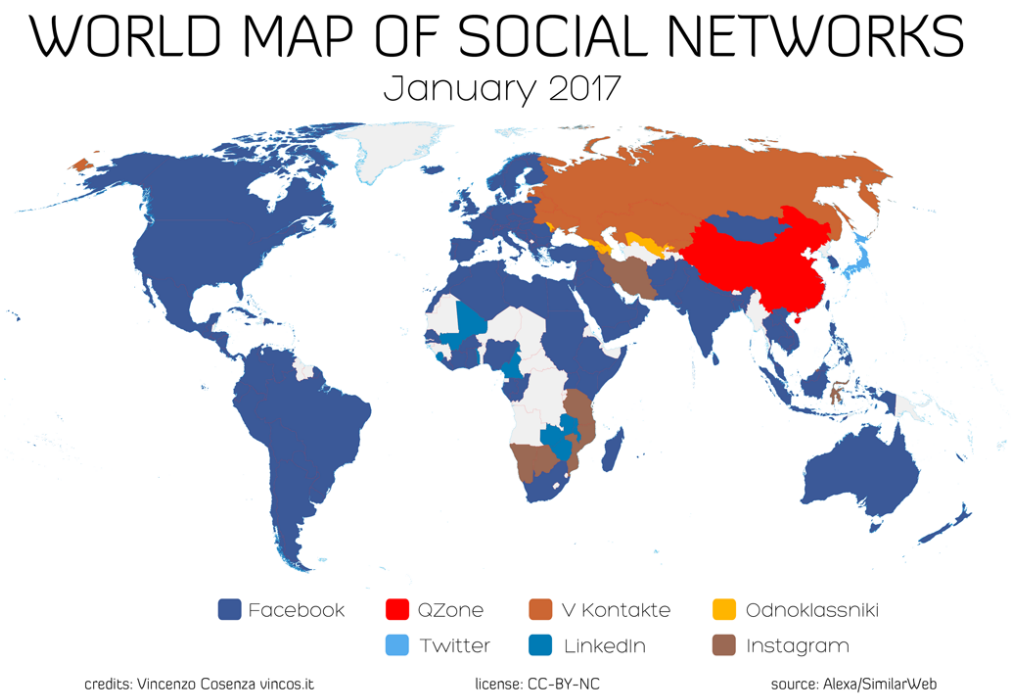


Figure 3.2: Most popular OSN services by country [177]. Image source: `http://vincos.it`

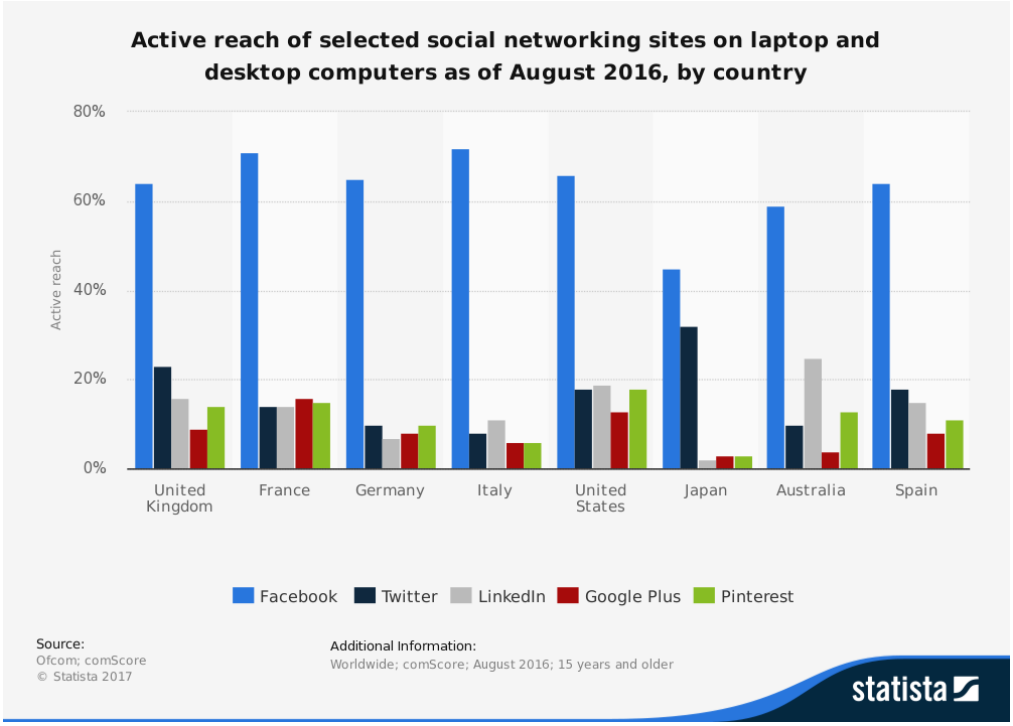


Figure 3.3: Reach of OSN platforms by country in August 2016 [16].
Image source: `https://www.statista.com`

OSN services are built on a federated open source architecture that allows users to entrust their social profile and personal information to a trusted provider or even host their own OSN server node. Diaspora, Friendica, and Mastodon gained a lot of attention in the last years due to their focus on decentralization and a privacy conserving architecture [97], yet couldn't attract a significant user base due to various reasons as analyzed by Koll et al. in [101].

Messenger services, such as WhatsApp or Skype, are intentionally not considered in this study, as they lack core social networking functionalities [62]. Most importantly, messenger services usually do *"not enable users to create detailed profiles containing a number of data fields [...], time-lines or news feeds, or to post information, explore other users' networks [...] or carry out many of the other features that form part of the social networking user experience"* [62].

Facebook

Facebook is built around the features of the user's social profile page, a feed of content published by other users, and messaging functionality. The *profile page* features information such as the user's name, a profile picture, a cover photo to be used as a background image, information about educational background and work, places one has lived, contact information and basic details such as gender or date of birth, relationship status information, list of family members, personal details, and life events. Furthermore, users can add a set of featured photos, a textual biography, or nicknames. While most of this information is optional, the parameter name is required. Facebook tries to enforce users to use their full real

name instead of abbreviations or fake names[3]. Secondly, Facebook features a list of *friends* with whom a user is connected. The friend list is populated by sending or accepting friend requests. The friendship relation in Facebook is bidirectional, meaning that once a friend request is accepted, both parties are added to each other's friend list. The friend list can be viewed by other users unless access permissions are defined to deny accessing this content. Once a user has been added to one's friend list, one automatically starts following this user. *Following* a user or a page is a subscription to content published by this user or page, while not being added to the friend list of the followed person. In contrast to the friend relation, following is unidirectional.

Furthermore, users can post *status updates* such as messages, images, or videos and *share* status updates previously published by other users. Each social profile page in Facebook features a *feed* for each user referred to as 'timeline'. This timeline shows all status updates of this user in chronological order, hence representing a history of a user's (public) activity on Facebook. Posts from the timelines of all friends are shown in a feed on Facebook's homepage with the intention to give the user an overview of what is happening in the (digital) lives of one's friends. To manage images, users can create *photo albums*, in which *photos* are published. Also, *videos* and *live videos* can be published to one's feed. Albums as well as individual photos or videos can be edited, for example by adding a title or description, location, or tagging other users. To further allow users to broadcast what is currently happening, for example at an event or place, Facebook supports *live video* streaming. Live videos can be up to 30 minutes long and are shown in the user's or page's feed. When the streaming is stopped, the video is added as a regular video to the collection of videos of the user or page.

To directly communicate with other users, Facebook features *conversations* between two or more communication partners. Conversations comprise textual *messages*, which can also contain other content such as images or files. Moreover Facebook allows IP *voice calls* and *video chats* and introduced a feature to *transfer money* [181]. Users can *poke* other users, a feature without a fixed meaning or purpose. Sending a poke to a user simply results in the user being notified about the fact that a poke has been sent to him and by whom. Users can react to other user's published contents in a variety of ways. The probably most popular one is to *like* content, where the number of likes is accumulated and displayed next to the content as an indicator of its popularity. As of 2016, Facebook introduced *reactions* that allow users to specify how they feel about content in more detail. Using this feature, users can chose between *like*, *love*, *wow*, *haha*, *sad*, *angry*, and *thankful* [182]. Furthermore, users can comment on content, which allows to have (public) discussions about a topic.

To represent businesses, places, companies, brands, products, companies, artists, bands, or just interests, Facebook features *pages*. Similar to a regular user's profile page, pages have a name, a feed, a profile picture, and a background image and can be used by the respective owner to get in touch with people by for example announcing news regarding to a brand or handle customer feedback for a product. Pages are created by regular users and bound to them, where

[3]What names are allowed on Facebook? `https://www.facebook.com/help/112146705538576`. Accessed: 5. May 2017

the creator of a page assumes the role of the page's administrator. Hence, as pages do not represent persons, they cannot be added as friends, yet pages can be followed, liked, and shared on one's timeline. Furthermore, one can send messages to a page, which are then to be answered by the administrator(s) of the page. Finally, pages can be *reviewed* by users by giving one to five stars and an optional textual explanation of their rating. Facebook also features information about locations without a page being created, called *places*. Users can like places and *check in*, in the sense of publicly announcing that they are at this place. As places are similar to a page, a place and a page can be merged to represent for example a restaurant. As a recent feature, Facebook's marketplace enables users to create and publish *offers* for items or services they would like to sell [183].

To organize *events*, such as a party, meeting, or general public event, Facebook allows users and pages to host *events* for a certain topic. The event's administrator can decide whether an event should be public or private, and can invite people to attend. Invited people are listed as attending, interested, maybe, declined, or no reply to allow others to see who is attending or not. Events have an internal feed, videos and live videos, albums with photos, and comments. Users can also organize themselves in *groups* for certain topics. A group is created by a user, who assumes the role of the group's administrator, and who can invite other group members. Groups allow members to access an internal feed, documents, events, albums and photos, and videos, which are not accessible by non-members. Support for *documents* and *files* allows users to create, view, and edit textual documents together or upload and download arbitrary files in a group.

Google+

In contrast to Facebook and other OSN services that focus on self-representation of users in form of social profile pages, Google+ is mainly organized around posts of content [46][184][185]. While the main page of Google+ shows a feed of posts from users that one follows, every profile in Google+ has a *feed* showing the posts of the respective profile owner. Posts can be textual messages, images, videos, or a user's current location, and can include a list of *tagged* users, indicating that they are linked to the content in some way. To express an opinion or answer, users can write textual *comments* on posts, which comprise a textual message and optionally a picture or hyperlink. Furthermore, a post can contain a *poll* with predefined answers, of which viewers can choose one from, where all answers to a poll are recorded and shown to viewers after they answered the poll. Posts can further be *shared* by users, thus publishing them on one's profile feed. Similar to Facebook's feature *like*, Google+ implements a *+1* feature. By clicking a +1 button, users express that they like or value content posted on Google+. The number of submitted votes is then shown as an indicator of the content's popularity on Google+.

A user's *profile* page in Google+ shows the user name, a profile picture, a background image, and a list of collections, communities, and posts created by the user. Furthermore, the profile optionally comprises information such as a textual 'tagline', gender, date of birth, personal contact details, workplace,

or education. What part of a profile's information is visible to others can be configured by the profile owner. Only the profile picture, background image, and user name will always be visible to others. In Google+, users may *follow* other users, thus forming unidirectional relationships instead of bidirectional ones. In comparison to other OSN services where a friend request has to be accepted first, following a profile is instantaneous and does not need to be approved. People one follows can be organized in circles, which are logical groups or categories, where users can belong to multiple circles simultaneously. This way, one's contacts can be easily organized, while adding users to circles has no effect on those user accounts and is also not announced to them.

To organize content, Google+ supports *collections*. Collections bundle content such as posts or images, allowing users to create channels for specific topics. Users can subscribe to collections, resulting in content posted to a collection one is subscribed to showing up in one's news feed. A collection is always owned and managed by the user who created it, allowing only him to add posts or images to the collection. Users are able to publish status updates with photos or videos, which are then implicitly organized in *albums*. While photos, videos, and albums can be liked, commented on, and shared, specifying a title, description, or location is not possible. Users can also organize in groups called *communities*. Compared to collections, a community has a list of member, who have to be invited by one of the community's moderators. Community members can post content within the community, which is accessible only for other community members.

Finally, Google+ supports *event* management. Events have a title, a date, a starting time, and a list of invited users, with additional optional parameters such as end time, location, website, transit information, and textual description. Furthermore, each event has a theme, which comprises a background image to be used when the event is displayed. Invited users can choose to accept the invitation by choosing between the options *yes*, *maybe*, and *no*, and can specify a number of guests they would like to bring in case this was allowed by the event's creator. Finally, users can post to an event similar as to a community, thus posting images or comments.

VKontakte

The Russian OSN platform VKontakte, being Russian for "in contact", is often described as the "Russian Facebook" due to its impressive popularity in Russia, Ukraine, Belarus and Kazakhstan [186]. Similar to Facebook, users in VKontakte can create a *social profile* with information such as one's name, gender, interests, or date of birth [187]. Users can add other users as *friends* or simply *follow* each other. Similar to Facebook, befriending a user involves sending a friend request that has to be accepted and is bidirectional, while following a user merely subscribes to content published by the followed user and is unidirectional. VKontakte features a *feed* for every user, on which users can publish *posts*, which can be *shared* by other users. Content can be *liked* and *commented on* by users.

Users can upload *photos* and *videos* and organize them in *albums*, where photos, videos, and albums all can have an optional title and textual description.

Streaming of *live videos* is also supported by the service, where the streamed video is shown in the user's profile. Users can send *gifts* in form of icons to other users, which have to be bought. Gifts received by a user are optionally shown on the user's profile page. VKontakte integrates *music* support, where users can upload and collect songs and organize them in *playlists*, while being accessible for other users and played via an integrated audio player. Users can also create, edit, and publish textual *documents* and *notes*, and up- and download *files*. VKontakte supports *communities*, which can represent a normal group, a *page*, or an *event*. Communities have a feed, photos, videos, and links and further specify a name, a description, a cover image, and a subject that specifies the community's purpose. Communities allow managers to send community messages, which are shown to all members as part of the community page. Regular communities can be open, closed, or private, where in closed or private communities the contents can only be accessed by community members. Communities that resemble *pages* are always public and can be followed by users. Finally, *events* can be open or closed and have a location, starting and ending date, and contact information. Invited users can accept or decline an invitation, or accept the invitation tentatively. All invited users are listed as members of the event. Users in VKontakte can sell items on a *market*. To privately sell a product, a user creates an item by specifying title, description, and an asking price, and adds a photo of the item. Furthermore, a category of the item being sold needs to be specified as well as the location where the item is being sold. Users interested in buying the item can then contact the seller via the platform.

RenRen

The Chinese OSN service RenRen, meaning "everyone's network", "*organizes users into membership-based networks representing schools, companies, and geographic locations*" [131]. The service, which was formerly named Xiaonei and allegedly copied its design and business model from Facebook [180], allows users to create a *social profile* and add information such as name, date of birth, or a profile picture. Users can furthermore set a *status* message, indicating what they are currently doing, which can be accessed by other users. Users, who accessed another user's profile will be listed as visitors, allowing the owner to see who visited their profile. To create connections to other users on the network, users can add each other to a *friend list*. The friend list is publicly accessible for other users. Users can post *status updates* in *blogs* and post *images* and *videos*, where images can be organized in *photo albums*. Users can *comment* on, *like*, and *share* posted content, and furthermore communicate privately by sending *messages* to each other. RenRen supports *places*, representing locations similar to Facebook's pages. A place has a name, an address, and geocoordinates. Users can then *check in* to places, announcing that they are currently visiting the place. Separately, RenRen makes use of *locations*, which represent regional areas. A location is defined by specifying country, province, city, county, a name of the nearest street, an address, and geocoordinates. Users can access a location's feed by specifying a latitude, longitude, and a radius to retrieve content posted with coordinates in the specified area.

Twitter

Twitter is a microblogging service that allows users to publicly post status updates called *Tweets* of a maximum length of 140 characters[4]. Tweets can be used to express opinions, broadcast news, or even announce events, and embed images and videos via a URL. All tweets posted by a user form a user's *feed*. Twitter uses a system of keywords called *hashtags* to tag messages and allow users to browse available tweets by topic. Users can *follow* each other, forming unidirectional links that require no permission by the followed user to be created. To prevent users from following or accessing one's tweets, accounts can be operated as private accounts, which require the account owner to accept follow requests and only show information of the account to accepted followers. Tweets of a user are then displayed to all his followers. Users can *retweet* tweets of other users, reposting the tweet in their own timeline with an indication that it has been originally published by another user. To communicate privately, Twitter supports sending *direct messages* between users, also including conversations with multiple users[5]. Finally, Twitter supports *lists* that work similar to groups. Lists aggregate tweets of one or multiple users, called members, where only members can publish tweets on the list. Lists can be either private or public, where content published on a private list is only visible to it's members, while public lists can be followed by other users.

Linkedin

Linkedin is a professional OSN service focusing on modeling relations between business partners and professionals. Users can create a *social profile*, which is focused on the representation of the career of the user and is subdivided in the categories experience and education, skills, accomplishments, and interests. Furthermore, users can specify their current employer and position. Users can add acquaintances or coworkers as *connections*, which have to be accepted and are established as bidirectional links. Linkedin allows users to post *status updates* in a *feed*, where posts can be simple textual messages or comprise *articles* or *images*, where articles are textual documents that can be edited and downloaded. Users can react to content by *liking*, *commenting*, and *sharing* the post, image or article. Users can furthermore *endorse* users for skills they claimed to possess on their profiles, indicating that one agrees or vouches for the user to actually possess the skill in question. To represent organizations or companies, Linkedin supports *companies*, which work like a combination of groups and pages at Facebook. Companies in Linkedin allow a company or organization to publish a description and updates in form of a feed. Furthermore, users can join a company, becoming listed as employees. Finally, Linkedin allows users to use a *messaging* system, which enables users to send private messages to one or multiple individuals.

[4]Twitter started testing tweets with an extended length limit of 280 characters in September 2017 [188]

[5]Twitter direct messages: `https://about.twitter.com/directmessages`. Accessed: 25. May 2017

Xing

Xing is a career-oriented OSN platform modeling professional relationships. Users can create a *profile* comprising information resembling a CV such as skills, current and previous employments, education, received certificates, or languages spoken. Finally, users can express what they are searching for career wise by specifying a position or project they want to work in. Users can create *connections* to other users, indicating that they worked together or are related in a professional context. To stay in contact with connected users, private messages can be sent, where two or more users can be added to a *conversation*. Xing features a *feed*, in which *status updates* are posted, which can be simple textual messages and can optionally comprise a link to an external website. Posted status updates can be marked as *interesting*, *commented* on, and *recommended*, resulting in the item being shared in one's feed. Companies and organizations can create *pages* in Xing to represent themselves, where users can *follow* the page and write reviews, where multiple aspects of a company or organization are rated, including "internal communication", or "equality". To allow users to organize themselves to discuss specific topics, Xing supports *groups*. Groups can be either closed or public, and specify a topic and a textual description. Groups have a feed, in which members can post status updates that are accessible only to group members. Finally, Xing features *events*, where events have a title, description, category, and star and ending date. Events can be either public or private, where invited users can see and register for attendance.

Diaspora

Diaspora[6] allows users to connect by *following* each other. While following a user establishes a subscription to this user's content, the followed user explicitly has to "start sharing" with him, where followers are grouped in "aspects" similar to Google+'s circles. Content in Diaspora is then published in a *stream* and visible for users of selected aspects. While *posts* are regular textual messages, they can comprise other types of content as well. Here, Diaspora supports *images*, where uploaded files are added to a global list and cannot be organized in albums. Furthermore, users can start *votes*, where the posting user specifies a question and a set of predefined answers, from which other users can then choose. The results of the poll are then displayed in the post. To react to posts, users can *like* a post and *reshare* it, where liking of content is restricted to the original posts, while comments and other content cannot be liked. Finally, users can *comment* on posts, where comments are simple textual messages and cannot contain rich content such as images or videos. Diaspora also features a *profile*, where name, gender, date of birth, and location can be specified. Furthermore, the profile can have a profile picture, a textual description, and a list of keywords the user is interested in. To communicate with other users directly, Diaspora supports *conversations*, where support for conversations was added in the latest release 0.6.

[6] Version 0.6 of Diaspora has been analyzed, which has been released on August 26th 2016: `https://blog.diasporafoundation.org/33-diaspora-version-0-6-0-0-released`. Accessed: 11. June 2017

Friendica

The distributed OSN service Friendica[7] mainly focuses on Microblogging and allows users to build a *social profile*, which can comprise information such as name, profile picture, gender, date of birth, contact information, keywords, or a textual description. Users can add each other as *friends* and exchange *messages*. Due to the underlying implementation of the federation, accessing another user's profile, will redirect users to the host of the viewed profile, resulting in a inconsistent look and feel of the network. Users can post *status updates*, which are published in a *stream* that is accessible by other instances of Friendica as well. Posts can be *liked*, *disliked*, and *starred*, allowing a more detailed response to published content. Furthermore, posts can be *shared*, *commented* on, and *tagged*, where tags can be either for other users or keywords in form of hashtags. Friendica allows users to upload *photos* and *videos*, where images can be organized in *albums*, and upload *files* that can then be accessed and downloaded by other users. Finally, the platform supports the creation of *events*, where users can specify a title, description, start and end date, and location. Furthermore, events can be shared with other users, who then can reply by accepting or rejecting the invitation, or respond tentatively.

Mastodon

Mastodon is a decentralized microblogging service with functionality similar to Twitter [140]. Built with the intention to provide equal or better functionality than the centralized original, it aims at allowing users to host their own servers and decentralize control of the social web. Mastodon[8] allows users to post *status* messages in a *timeline*, where *hashtags* can be used and other users can be *mentioned*. For direct communication, Mastodon supports restricting the audience of individual status posts to specified users, allowing private *conversations*. The service features a simple *profile*, allowing users to *follow* each other. Similar to Twitter, status posts can be *favorited* and *replied* to. Furthermore, Mastodon supports uploading and managing of *image* and *video* files. Finally, Mastodon offers a fine-grained management of privacy settings and access control that allows users to restrict access to individual parts or the entire user profile.

Instagram

Instagram focuses on *images* posted by users in a *feed*. Users can *follow* each other, resulting in a subscription to the followed user's feed. Images posted on the feed can be *liked* and *commented on* by other users. Instagram also features a *profile* page, which allows a user to specify basic information, being a name, biography, profile picture, gender, title, location, and contact information such as an email address, website, and phone number. A unique feature of Instagram

[7] Version 3.5.2 of Friendica has been analyzed, which has been released on June 6th 2017: `http://friendi.ca/2017/06/06/friendica-3-5-2-released/`. Accessed: 11. June 2017

[8] Version 1.5.1 of Mastodon has been analyzed, which has been released on August 6th 2017: `https://github.com/tootsuite/mastodon`. Accessed: 7. August 2017

is the possibility of users to choose and change their username, which is used as the profile's identifier. Finally, users can send direct *messages* to each other, where communication between multiple users at once is not supported.

Pinterest

Pinterest evolves around the *pinning* of images, which are then posted in a *feed*. Pinned images can have a textual description and a link leading to a website associated with the pinned image. Unlike other OSN services, Pinterest is a "social curating service" and does not focus on connections between users, but rather connections between users and content [189]. Consequently, users can *follow* each other as well as *boards*, where boards are collections of pins, for example by category. Following a user or board creates a subscription to pins posted on the followed profiles or boards. Boards can be created with a public or secret setting, where users need to be invited to become a member to view and pin images to and from a secret board. Furthermore, pinned images can be *saved*, essentially sharing them in another board. Pinterest features a simplistic user *profile* that features the username and a profile picture. The profile furthermore shows a feed of pins posted by the user and the boards he created. The main page of Pinterest shows a feed of pinned images chosen by the service based on what a user likes. Users can *comment* on pins, where comments consist of simple textual messages. Furthermore, users can announce that they "tried" a pin, indicating they *reviewed* the pin's content and approve it in the sense that they successfully followed a recipe or followed the advice contained. The total number of approvals is shown next to the pinned image. The approving user can optionally write a textual comment and upload further images to give more detailed *feedback*.

3.4.4 OSN Feature Taxonomy

To make functionalities of different OSN services comparable, a taxonomy for OSN features is required. While the various implementations of a certain OSN feature may differ between services, they still describe similar or even equivalent functionality. For example, a Facebook *like* is fully equivalent to a *+1* in Google+, while support for *photos* differs in details between the two OSN platforms. To study and compare the support of OSN features in different OSN services, a taxonomy for OSN features has been derived.

Social Profile

A *social profile* is a customizable page in an OSN service that allows users to express themselves and describe what is going on in one's life. The social profile stores information in key–value pairs such as the user's name, profile picture, a cover photo to be used as a background image, information about educational background and work, places one has lived, contact information and basic details such as gender or date of birth, relationship status information, list of family members, personal details, and life events. Furthermore, users can add a set

of featured photos, a textual biography, or nicknames, where the type of data users can add to a social profile page differs between OSN services. While social profiles are implemented in all surveyed OSN services, the OSN feature differs in the information stored within a profile. While Facebook or Google+ for example allow users to specify a variety of information, Instagram and Pinterest support only a minimalistic profile. Other special OSN services such as Linkedin or Xing focus more on representing one's career path, essentially turning the social profile into a web-based curriculum vitae. The issue of varying information available in social profiles is also addressed by Open Social as described in Chapter 3.7, which specifies an extensive list of optional parameters that could be added to a social profile, with only a user identifier and a user name being mandatory.

Link

A *link* is a connection between individuals in an OSN service, indicating friendship, acquaintance, or merely interest in a person the person's context [190]. Allowing users to connect to each other is supported by all surveyed OSN services, hence being one of the most essential OSN features. To add a user to one's list and establish a link between two individuals, a request is sent by one of the users to the other user, where existing links to other users are collected in a link roster, also referred to as friends or followers list [190]. Most OSN implementations require the requested user to explicitly accept the request before the user is actually added to the link roster. Once accepted, links can either be uni- or bidirectional. While unidirectional links indicate that a user *A* is connected to a user *B* but not the other way around, bidirectional links assume that if user *A* is connected to user *B*, *B* is also connected to user *A*. All links established in an OSN service form the social graph, representing information about how individuals are connected to each other. Links between users of OSN services are often also used for access control, where users in the link roster are allowed to access certain parts of a user's social profile, while others are not. Similar to links between users indicating friendship or being acquainted, *follow*-relationships indicate interest in content published by another user or page, which is then either proactively pushed to the follower's OSN profile, or reactively fetched on demand. Following someone hence equals subscribing to content the followed entity posts to its stream, where follow-relations are usually unidirectional. All surveyed OSN services support either uni- or bidirectional connections with some services supporting both at the same time. As bidirectional connections can be composed of two unidirectional links between two users, links in this taxonomy are unidirectional to support both paradigms of connections between users.

Conversation

Conversations bundle textual communication in form of text messages between two or multiple communication partners. Messages can be sent by any participant of a specific conversation and are received by all other participants. The term conversation covers simple messaging restricted to two communication partners as well as group communication between multiple communication partners. Messages in a conversation usually have a status parameter indicating its local status, such as received, read, or deleted. Using this status, the recipient can determine which messages he has already read, deleted, or which are still unread. Conversation implementations can also comprise a remote status that indicates whether a message has been received, read, or deleted by a recipient. Moreover, messages can often contain content other than text, such as images, videos, voice recordings, links, location information, or general file attachments.

Poke

A *poke* is a feature without a fixed meaning or purpose. Most prominently implemented in Facebook, a poke can be sent to another user resulting in a notification that this user was "poked" by the initiator of the poke. The feature also exists in some other services, for example in an XMPP extension called *attention* [191], or as the feature "gruscheln" in StudiVZ [129] and is sometimes also referred to as *nudge* or *buzz*. The feature was even turned into a standalone service called Yo[9], whose creators describe it as "*the simplest notification platform" where you can send a "simple ping that let's you know something happened and the predefined context fills in the data that is not provided in the notification*" [192].

Like

To allow users to express that they like content such as status updates, photos, profiles, or messages, most OSN platforms provide a *like* feature. Users can send a like, usually by clicking a specific like button next to the content. The total number of likes is then summed up and displayed, indicating how popular the content is. While most OSN platforms such as Facebook, VKontakte, LinkedIn, or Twitter implement the feature using the term "like", other OSN platform implement the same functionality using different terms. For example, Google+ allows users to send a "+1", Xing uses the German term "interessant" (engl.: "interesting"), and Diaspora uses the term "love". The feature is one of the most popular ones in OSN services, where only Pinterest, as the only analyzed OSN implementation without support for the like feature, removed it's support early 2017[10].

[9] Yo app: `https://www.justyo.co/`. Accessed: 20.6. 2017

[10] As of early 2017, Pinterest removed a previously existing like feature and added the save functionality instead [193].

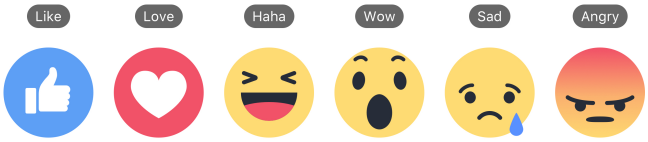


Figure 3.4: Visualization of reactions in Facebook. Reactions *none* and *thankful* have no graphical representation. Image source: `https://en.facebookbrand.com/assets`

Reaction

As the like feature does not allow users to express more complex reactions to content, *reactions* offer feedback in a variety of predefined expressions. Reactions work very similar to the likes, but allow the user to chose from a variety of meanings. For example, Facebook's implementation of the feature [194] allows users to chose between *none*, *like*, *love*, *wow*, *haha*, *sad*, *angry*, and *thankful* as depicted in Figure 3.4. The number of reactions of every type are then listed – similar to the number of regular likes – next to the content in that context they were posted to indicate how many people reacted to this and in what way.

Collection

Collections allow users to publish content such as images or videos bundled by topic, event, or category. A collection has a title and can additionally have a textual description, describing a location or a date. The collection's owner can add or remove content to a collection.

Image

Users use *images* such as photos or graphics to express ideas, share experiences, and visualize achievements. The image feature allows users to post and manage pictures or photos in their OSN profile, where an image can optionally be added to a collection or posted in a stream. Images either store the actual image data in an encoded format or a URL of an image file. Optionally, an image has a title, description, and additional image metadata such as the image's resolution, encoding algorithm, width and height, date of creation, copyright information, or location information.

Video

Videos work similar to images and can be posted and managed by users in their OSN profiles, for example by posting it to a stream or adding it to a collection. Besides either storing the video data in an encoded format or a URL to the video file, videos can optionally have a title, description, and metainformation such as information about resolution, length, width and height, video codec, date of creation, copyright information, or location at which the video was recorded.

Live Video

Live video allows users to share a video stream of what is happening instantly. While the user records the video using for example a mobile phone, the video is streamed to the OSN service and can be seen by other users. After the live video feed is ended, the video is added as a regular video to the user's collection of videos.

Comment

To express a more detailed opinion or reaction to content in an OSN, most OSN services allow users to write *comments*. In a general sense, comments are textual feedback to content such as photos, status updates, or even other comments, and are attached to the content that is commented on.

Voice Call

Mostly common in IM services, voice call functionality allows users to call other users in a VoIP manner [195]. The feature has become very popular as a replacement for international phone calls, as only the fees for Internet access apply.

Video Call

Similar to voice calls, video calls allow users to call other users with a video feed. Extended support for this feature even supports video conferencing.

Stream & Activity

OSN services use a *stream* of *activities*, where users can inform their network of friends and acquaintances about news and events in their life by posting about something they did, saw, or experienced. An activity usually comprises a textual message, image, video, link, or any other type of content, and furthermore stores other information such as the activity's author, time being published, or tagged users. To further specify the type of content enclosed, a parameter type is used in many implementations, where the standard Activity Streams 2.0 [102] has been widely adopted to describe activities and streams. The Activity Streams 2.0 Vocabulary differentiates between twelve distinct types of status update content types (objects), such as *document*, *event*, *image*, *place*, or *video*, and further specifies twenty-eight different types of actions to be expressed, such as *announce*, *create*, *listen*, *offer*, or *travel* [196]. Activities posted by a user are usually published in his stream, a list of all status updates by that user. The stream, also referred to as "timeline" or "feed", hence comprises a journal of that user's activity within the OSN service. Given the required access permission, status updates can also be posted on another user's stream, or on the private stream of a group, page, or event. Status updates of linked users and pages are normally shown in a main stream, functioning as a "*personalized stream of stories, recommendations and news from the people, news sources, artists and businesses you've*

connected with" [197]. Users can also *share* status updates that have previously been posted by other users. Sharing a status update creates a copy of the original post and re-publishes it in a user's stream. The post is marked as a share to indicate that the content has been posted by someone else before and lists the name of the original author.

Tag

To mention a user in a status update, image, or other content, a *tag* can be created that is attached to the content the tagged person is mentioned in. This way, the participation in or presence of users at in for example an event can be indicated. When tagged in content, users are listed when the content is displayed, for example in the description of an image in a photo gallery. Some implementations such as Friendica extend the concept of tags by additionally allowing to tag content of other users with hashtags being textual keywords.

Event Management

Event management functionality allows users to organize events using OSN services. An event usually has a title, a date, a starting time, and a list of participants, and other optional parameters such as a description, ending time, location, or a background picture. The event's host assumes the role of administrator and can hence change the configuration of the event. When invited to an event, a user can choose to decline, accept, or accept tentatively, where the information about who is invited and who accepted the invitation or not can be seen by all other invited participants. To communicate with other participants and announce news, an event has a stream, where participants can post status updates. Content published in the context of an event is only visible and accessible to the invited participants. Depending on the support of the OSN service, an event may support further OSN functionality such photo albums or documents, where the respective information is only accessible to the event's participants.

Vote

Voting defines the ability to create a poll for a certain topic with a number of configurable options. Depending on the type of the poll, users can then select one or multiple of the options. The results can be displayed, showing which option got the most votes. For example, a group of friends planning to go to the cinema could decide which movie they want to see using a poll.

File

Files can be published in a social network for other user to download and specify a title and a file type. Files are usually in a binary format, such as pdf documents or zip files.

Document

Documents are files that can be edited by users directly, allowing collaboration on articles, spreadsheets, or lists. Depending on document type and access permissions, documents can be either just read or even edited and overwritten by users.

Review

A *Review* describes functionality to express a comparative assessment of the quality, standard, or performance of content. A review allows a user to express his opinion in form of a rating by giving up to five stars for content, where content is usually a page, venue, location, brand, or similar. A one-star-rating would equal the worst, five stars equal the highest possible opinion. Optionally, a review can include a textual comment to allow users to explain the rating they have given.

Group

Groups are logically separated areas in an OSN in which content can be published that is only accessible within the group. To distinguish who is able to access the group's contents and who can not, a group has a list of members, who exclusively are able to access the group's contents and publish new content in the group. Groups can be open or closed, defining whether everybody can join the group or approval of the group's administrator is required to do so. Similar to chat rooms in IRC, groups are mainly used for collaboration such as to discuss topics, events, or other activities. Groups have a name, a description, and in most cases a background picture. A group has an administrator, who can change the configuration of the group, including to change the group's topic, invite users, or configure the group's access permissions. Furthermore, a group has a list of members who can access the group. Each group has an activity stream, where content posted to the group is shown. Depending on the support of the OSN service, groups may support other OSN functionality such photo albums, documents, or events.

Page

A *page* is a website within an OSN service that represents an entity that is not a regular user. Pages are used to represent businesses, places, companies, brands, products, artists, bands, movies, or just interests. Similar to regular social profile pages, pages feature a name, a description, and a profile picture. Each page has one ore more administrators, who can create and publish content in the context of the page and also change the page's configuration.

Check-In

Originally introduced by services such as Foursquare[11] or Gowalla[12], a *check-in* is a public announcement of a user that he is currently at a certain place, usually a venue such as club, restaurant, or a public place. Check-ins can be used to determine how many people are at a certain place, or if a certain person is even currently there. The feature raised concerns regarding to privacy issues, as anyone could see when you were not at home[13].

Music & Playlist

Music support allows users to upload audio files. Audio files have a title, an artist or band name, and album information. Furthermore, they can have additional metainformation such as the audio codec or duration. Audio files can be added to playlists, which arrange multiple audio files in a specific order, where users can specify a playlist's title and description. Audio files and playlists can be viewed by other users, where usually an integrated audio player allows users to play the selected audio file or playlist.

Gift

Gifts are icons that can be bought and gifted to users, who can then display them in their social profile. The amount and type of the received gifts is thought to indicate how respected or loved a user is by his friends and connections. Gifts usually have to be bought using real money or a special kind of currency introduced by the OSN service.

Offer

Offers describe physical goods or items a user is willing to sell. Each offer describes the item to be sold by specifying a category, selling price, title, description, and physical location. Furthermore, pictures of the item can be added to give potential buyers a better understanding of the item for sale. Offers can be viewed by other users, who can get in contact with the seller in case they are interested.

Endorsement

To confirm a claim someone made on his profile, users can *endorse* that claim to publicly give the claim more credibility. The total number of endorsements of a certain claim is then displayed in the user's profile. Endorsements are mostly used in career networks, where users claim to have a certain set of skills, and coworkers and acquaintances can endorse users for the skills they claim to possess.

[11] Foursquare: `http://foursquare.com`. Accessed: 12. April 2017
[12] Gowalla: `http://blog.gowalla.com/`. Accessed: 12. April 2017
[13] PleaseRobMe.com: `http://pleaserobme.com/why`. Accessed: 10. June 2017

3.4.5 Sonic Core Featureset

From the analysis of the OSN features supported by the twelve surveyed OSN services as listed in Table 3.2, a core featureset of OSN functionality can be derived. While the OSN features social profile, link, stream & activity, and comment are supported by all surveyed OSN services, likes and conversations are supported by 11, and images and tags by 9 out of 12. Groups, videos, and collections are also popular OSN features, yet are only supported by 6 of the surveyed networks, followed by events and pages (supported by 5 services each). All other OSN features are supported by 4 or less networks, being check-ins, live videos, voting, files, documents, offers, and reviews (supported by 2 services), as well as reactions, music & playlists, poking, gifts, voice calls, video calls, money transfers, and endorsements (supported by 1 service each).

As the results of this survey depend on the choice of OSN platforms and may hence differ in a similar comparison using a different set of surveyed OSN services, only OSN features with a high popularity are considered as features of the core featureset. Furthermore, as OSN services continuously evolve and implement new features, the percentage of OSN services supporting a certain feature may change over time. For the Sonic core featureset, a threshold of two thirds was chosen for an OSN feature to be considered in the core featureset, resulting in the following definition of the Sonic core featureset:

Definition 12: Sonic Core Featureset
The Sonic core featureset comprises OSN features supported by a two-thirds majority of OSN services, being profile, link, conversation, stream & activities, image, like, comment, and tag.

Based on the results of the analysis of OSN features supported by twelve popular OSN services, a more detailed definition of an Online Social Network can be given that extends the original definition given by Boyd and Ellison in [24]:

Definition 13: Online Social Network
An Online Social Network, short OSN, is a web-based service platform that allows users to (i) create and access social profile pages, (ii) link to each other and traverse those links, (iii) exchange messages, (iv) post activities in a stream, (v) tag each other in posts, (vi) comment on published content, (vii) like content, and (viii) publish images. OSN services can furthermore implement additional OSN features for a unique portfolio and user experience.

Table 3.2: Comparison of the support of OSN features of different OSN services.

Feature	Facebook	Google+	Friendica	Diaspora	VKontakte	RenRen	Xing	LinkedIn	Twitter	Mastodon	Instagram	Pinterest
Social Profile	●	●	●	●	●	●	●	●	●	●	●	●
Page	●	○	○	○	●	●	●	●	○	○	○	○
Link	●	●	●	●	●	●	●	●	●	●	●	●
Conversation	●	●	●	●	●	●	●	●	●	●	●	○
Voice Call	●	○	○	○	○	○	○	○	○	○	○	○
Video Call	●	○	○	○	○	○	○	○	○	○	○	○
Stream & Activity	●	●	●	●	●	●	●	●	●	●	◐[a]	●
Like	●	●	●	◐[b]	●	●	●	●	●	●	●	○
Reaction	●	○	◐[c]	○	○	○	○	○	○	○	○	○
Comment	●	●	●	●	●	●	●	●	●	●	●	●
Tag	●	●	◐[d]	●	●	●	○	●	●	●	○	○
Collection	●	●	●	○	●	●	○	○	○	○	○	●
Image	●	●	●	●	●	●	○	○	○	●	●	●
Video	●	○	●	○	●	●	○	○	○	●	●	○
Live Video	●	○	○	○	○	○	○	○	○	○	○	○
File	●	○	●	○	●	○	○	○	○	○	○	○
Document	●	○	○	○	●	○	○	●	○	○	○	○
Music & Playlist	○	○	○	○	●	○	○	○	○	○	○	○
Event	●	●	●	○	○	○	●	○	○	○	○	○
Group	●	●	○	○	●	●	●	○	●	○	○	○
Poke	●	○	○	○	○	○	○	○	○	○	○	○
Review	●	○	○	○	○	○	○	○	○	○	○	◐[e]
Vote	●	○	○	●	○	○	○	○	○	○	○	○
Check-in	●	○	○	○	○	●	○	○	○	○	○	○
Gift	○	○	○	○	●	○	○	○	○	○	○	○
Offer	●	○	○	○	●	○	○	○	○	○	○	○
Money Transfer	●	○	○	○	○	○	○	○	○	○	○	○
Endorsement	○	○	○	○	○	○	○	●	○	○	○	○

● Supported ◐ Partly supported ○ Not supported

a) Instagram supports a stream of images or videos, but does not support publishing other content.
b) Diaspora only supports liking status updates, but not comments or other content.
c) Friendica supports reacting to content by liking, disliking, and starring.
d) Friendica supports adding textual tags to content.
e) Pinterest only allows to write textual reviews. Giving a rating is not supported.

3.5 User Identification

A decentralized, heterogeneous OSN ecosystem, in which migration of entire user accounts between OSN platforms is supported, needs to provide means to discover OSN profiles and enclosed data where established links between social profiles and associated content are kept intact even when an OSN profile is migrated to a new OSN platform and is hence not accessible anymore at the former location. As of today, user accounts and content is mostly identified and addressed via URIs [198], which comprise the domain name of the OSN platform they are created and stored in. In such a scenario, if an OSN profile that is identified in this manner is migrated to a new OSN platform, identifiers for the profile itself as well as for all enclosed content invariably remain pointing to the former OSN platform's domain and hence become invalid. To maintain a profile's accessibility and addressability, identifiers of all content and the user profile itself would need to be changed to the new OSN platform's domain, resulting in links in other OSN profiles pointing to the migrated profile to break. Examples for this are friends lists pointing to the migrating user's profile, or comments written by other users that reference content that's stored in the migrated OSN profile. Furthermore, linking to content published in a social profile from other web services and applications, for example in form of a website or blog, is not feasible if links should be kept intact even after a profile has been moved. If identifiers change when migrating an OSN profile to a new location, links to the migrated profile become invalid, rendering links in other user's friends lists, as well as all content created by or referencing the migrated profile or content in it, unreachable. To alleviate this issue, a separate mechanism would be required to update all references in all content in all profiles on all OSN platforms, including every content object ever created that links to this identifier. Friendica, which uses email-style identifiers, proposes a rudimentary approach for profile migration where a user manually extracts his social profile dataset and re-imports it at another Friendica node. After this manual migration process, linked user profiles are informed about the migration, where the user identifier is updated to reflect the new profile location[14]. While this approach allows a user to move his profile manually, links to content are broken as the user identifier for the moved OSN profile is changed. For example, content containing hyperlinks to a user's profile page or other content he published in his OSN profile are not updated, resulting in broken links.

To avoid links being interrupted, identifiers should remain unchanged even after an OSN profile is moved to a OSN platform with another domain name. Hence, using domain-agnostic identifiers will allow user profiles to remain completely independent of the OSN service they were originally created in, resulting in platform independent social personas as of Requirement R2 as defined in Chapter 3.3. As the domain-part of domain-bound identifiers guarantees global uniqueness, domain-agnostic identifiers must introduce other means to prevent ambiguity of identifiers following Requirement R10. Additionally, identifiers must be created and managed by users themselves, as

[14]Moving accounts in Friendica: `https://github.com/friendica/friendica/blob/develop/doc/Move-Account.md`. Accessed: 23.6.2017

Requirement R4 disallows any form of central authority that could exert control over the entire OSNF or parts of it. This is due to the fact that in a system where identifiers are resolved to locate the actual social profile linked to it, a central authority could block the resolution of identifiers easily and hence indirectly exert control over communication in the entire OSNF by inhibiting the lookup of OSN profiles and content. Even when control is distributed between multiple authorities, where each authority would be in control of creation and management of a distinct set of identities, each authority would remain in full control of the identities it manages and could hence block access of any affected user to the OSNF, for example when a customer migrated to a competitor's OSN platform. To prevent a possible misuse of power by a central or semi-central authority, identifiers in Sonic are created and resolved independently of any authority that could voluntarily interfere with the identifier's usage. As in such an architecture no central authority is able to assert a user's identity claim, users need to be able to prove ownership of their identifiers to other users and OSN platforms so that impersonation of users is prevented. Finally, to allow identifiers in Sonic to be resolvable to the actual OSN profile's location without the help of a central authority, Sonic distributes control between all participating servers via a distributed hash table (DHT). For this purpose, Sonic introduces the Global Social Lookup System (GSLS), a distributed directory service built on peer to peer (P2P) technology using DHT technology.

3.5.1 Related Work

Services that manage multiple users or objects require a measure of identification to distinguish between individual users or objects, where the concept of *identity* is defined to be a set of attributes related to an entity [169]. For this purpose, an identifier (ID) is assigned to each entity, where an identifier is a name that usually is a sequence of letters, numbers, or symbols, with the usual intent of being unique in a certain domain. This assures that each user or object can be uniquely addressed via its assigned identifier, and two equal entities can always be distinguished from each other. Traditionally, a service assigns identifiers to each user and data object, which is unique within the domain of the service but not necessarily on a global scale. As a well known example, the Linux operating system identifies each user via a unique user name and a serial number (*userid, uid*). The uid is used by the system to identify users, while the actual user name is mostly used for authentication and displaying purposes. OSN services also identify users by a – not uncommonly numerical – user identifier, which has to be unique within the domain of this service or application. As long, alphanumerical identifiers are hard to remember and distinguish for humans, most services introduce a human readable username or *handle*, which in most services also acts as an identifier for the same profile. Hence, usernames also need to be unique and are hence not suited for representing regular names. Furthermore, most OSN services allow their users to pick a displayable, human readable name, also referred to as *display name*, which is not necessarily unique and is used as a normal name when displaying information about a user. For example, Facebook uses two identifiers for users, a user name and a numeric ID. While both the user

name as well as the numeric ID are unique in the domain of Facebook and can hence be used to address user profiles, a regular name, which is not necessarily unique, is displayed in the profile. This name is thought to represent the user's actual name and Facebook urges users to use their actual first and last names[15]. For example, Mark Zuckerberg's user name is *zuck*, his ID is 4, and his regular name is *Mark Zuckerberg*. To retrieve his social profile, both identifiers can be used: `https://www.facebook.com/zuck` as well as `https://www.facebook.com/4`.

If identifiers are only issued and resolved within a single domain or service, identifying and localizing the individual entities is simple as creation of all identifiers can be easily controlled by the service operator. The drawback of this approach is that the identifier can not be used directly to identify entities outside of the issuing domain. An example of this are user names in UNIX-based systems, where user names are unique on each system, but cannot be used to identify a user on another server. In most services and systems that require identifiers to be used and resolved across domain borders, identifiers are composed of a local identifier, which is unique in the domain in which it is issued, and the service's domain name, which globally and uniquely identifies the service domain. This simplifies the process of determining whether an identifier is already in use by another entity, as querying (all) other domains for already issued identifiers is not necessary. This allows situations in which an identifier for a user *Alice* can exist in multiple separate domains at the same time without any conflict, as only the identifier's domain-part needs to be globally unique. This kind of composed identifiers is used by web-based services and applications, where the domain's full qualified domain name (FQDN) is used as the identifier's domain-part. Here, first the identifier's domain part is resolved via the Domain Name Service (DNS) [199], while resolving the identifier's local part is delegated to the respective domain. Composed identifiers are used by most web-based applications and services, for example in the form of URIs [198], email addresses [200], or jabber-IDs (JID) implemented by XMPP in the format `local-part@domain-part` [201].

Unified Resource Identifiers (URI) or International Resource Identifiers (IRI) [198] are also used to uniquely identify entities such as documents or persons. Basically, URIs comprise a *scheme*, which is usually registered with the Internet Assigned Numbers Authority (IANA)[16], an *authority part*, describing a domain or IP address, a path further specifying the targeted resource in hierarchical form, and an optional query- and fragment-part, where the query can be used to describe key-value-pairs, and the fragment to specify a fragment-id. Using URIs, the identified entities can be classified and categorized, for example by specifying paths such as `http://company.com/employees/berlin/alice`. By utilizing URIs as identifiers for entities in a service or app, users and services can easily resolve an identifier to a resource or document, which then may provide further information about the linked entity.

WebID [202] uses URIs as identifiers for individuals or entities, which can be resolved to a profile document via HTTP or HTTPS. Profile documents are

[15]Facebook naming regulations: `https://www.facebook.com/help/112146705538576`. Accessed: 23.8.2017

[16]Internet Assigned Numbers Authority (IANA): `https://www.iana.org/`. Accessed: 12.8.2017

provided in RDF [203] and provide semantic information about the described individual or entity. WebID was extended in 2009 by FOAF+SSL to a decentralized and secure authentication protocol [204]. The standard, which was later renamed to WebID-TLS [205], published an entity's cryptographic public key in the WebID document, which could then be used to establish an encrypted communication channel with the owner of the matching cryptographic private key. The standard uses FOAF social graph information [104] specified in the WebID documents to verify how a requesting entity is related to the identified individual or entity. This way, the standard even allows for authentication of users without the need for a central authority, as users build a decentralized web of trust using FOAF. Yet, with WebID, each user is required to own and operate a domain and functioning website for a WebID to be resolvable.

WebFinger is an HTTP-based protocol for discovery of information about entities being individuals or things, where entities are identified via email-style URIs using a special URI scheme `acct://` [103]. WebFinger resources will answer a query for a specific entity with a JSON Resource Descriptor (JRD) specifying the targeted entity with a subject, as well as aliases, properties, and links. WebFinger is used as the discovery protocol for OpenID Connect [206], and has been implemented by DOSN services such as Diaspora or Friendica.

In 2003, the OASIS group introduced eXtensible Resource Identifiers (XRI) as an identifier scheme for abstract identifiers [207]. XRIs are designed to be domain-, location-, and platform-independent and can be resolved to an eXtensible Resource Descriptor Sequence (XRDS) document via HTTP(S) [208] in case they comprise a resolvable domain name. Work on the XRI 2.0 specification was discontinued in 2008 by the XRI Technical Committee at OASIS.

OpenID is a decentralized authentication framework promoted by the OpenID Foundation[17] that allows users to be authenticated at any web service supporting this standard [209]. The standard allows users to choose an arbitrary OpenID Provider (OP) and register an identity, which a user then can use to register and login with arbitrary other services supporting the standard. OpenID employs URLs as identifiers, where the OpenID 2.0 specification also supports XRIs [207] to be used, resulting in identifiers being unalterably bound to the domain of the provider. In OpenID, users connect to a service referred to as Relaying Party (RP), identifying themselves with an OpenID identifier. The RP then contacts the OP and retrieves an Extensible Resource Description Sequence (XRDS, formerly Yadis [208]) document describing the capabilities and provided functionality of the OpenID provider, where a shared secret is established between RP and OP. The user's user agent is then redirected to the OP to authenticate using his credentials and is redirected back to the RP after successful authentication with the shared secret. The RP then verifies the shared secret and accepts the user agent as being successfully authenticated. The standard has been combined together with OAuth into OpenID Connect [206], extending the concept to authorizing a service (RP) to access a resource owned by the user while being authenticated similar to OpenID.

[17] OpenID Foundation: `http://openid.net/`. Accessed: 12.8.2017

As of today, almost every web-based service utilizes domain-bound identifiers, as they can be issued easily without the need to check for possible collisions. As the DNS is used to resolve the domain-part of the identifier to the issuing service domain, the identifiers can also be used to identify entities in other service domains. Yet, migrating a domain-bound identifier to a domain distinct from its issuing domain introduces additional complications. Firstly, the local-part of the identifier may be used in the new service domain by another entity and hence result in a collision so exchanging the domain-part of the identifier is generally not possible. Secondly, if the identifier's domain-part is kept, the former service domain would remain responsible for resolving the identifier, giving the domain's operator full control of disabling identifiers issued in his domain. Solutions such as WebID-TLS are able to circumvent the problem, yet require every user to register and maintain a domain name - a task that could overburden the average user of OSN services. To overcome these issues, a single issuing domain can be used for identifiers.

In consequence, if entities need to be identified independently of a fixed service domain, domain-bound identifiers cannot be applied. In scenarios, where identifiers are not issued by a central authority but in a non-orchestrated, distributed fashion, random values can be used as identifiers, where measures need to be implemented to reduce the likelihood of a collision. Cryptographic hash functions can be used to derive a seemingly random bit-sequence from a combination of random input, for example created by a random number generator (RNG), or deterministic input values. The resulting output is then, depending on the quality of the used hash function, unlikely to be created twice and may therefore be used to identify entities. This paradigm is used for example by Universally Unique Identifiers (UUID), also known as Globally Unique Identifiers (GUID) [210]. UUIDs are 128 bit identifiers created by using a machines MAC address and a timestamp (version 1 and 2), MD5 or SHA1 hashes of the machine's namespace identifier (version 3 and 5), or a random number created by a Random Number Generator (RNG) (version 4). The uniqueness of UUIDs is hence based on the assumption that generating the same identifier twice is very unlikely and creation of the identifiers abides to the standard, meaning that MAC addresses are not forged with the intent of creating a collision. The UUID standard does not describe means to resolve an identifier to a location or object, and no central registry of issued UUIDs exists.

Other identifier schemas followed similar approaches. Twitter Snowflake [211] is an identifier schema based on hashing a timestamp, a preconfigured machine number, and a sequence number. Twitter Snowflake was built for fast and distributed id generation without the need for the machines generating the ids to coordinate with each other. Snowflake was discontinued in 2010, but other implementations of the approach exist, for example PHP Cruftflake [212] or Boundary Flake [213]. Boundary Flake, which follows a similar approach as Twitter Snowflake, is a *"decentralized, k-ordered id generation service"* [213] that hashes a machine's MAC address, a UNIX timestamp, and a 12 bit sequence number to create a 128-bit identifier.

In comparison to identifiers composed of a local and a domain-part, identifiers based on randomness or machine-dependent information such as MAC addresses can be generated in a distributed fashion, that is without a central entity controlling the process. Anyhow, verification of an entity's identity might be problematic, as any entity can assume any ID. To circumvent this, distributed entity's need to be resolvable in a trusted and secure manner.

Directory Services

In order to locate a resource identified by an identifier, the identifier is usually resolved to a data record or a document that specifies further information about the user or entity that is linked with the identifier. Such data records are usually stored and published in a directory service that authorized clients and services can access to resolve an identifier.

Directory services organize data records, referred to as *entries*, in a hierarchical structure. Each entry is addressable via a distinguished name (DN), which serves as an identifier that is not necessarily unique. Entries are organized in a hierarchical manner, where each entry stores a reference to its parent entry, resulting in a tree-like data structure. This allows each entry to be uniquely identified and addressable via a unique path from the tree's root entry to the entry in question. A concatenation of all entries on this path results in a unique identifier, the relative distinguished name (RDN). Directory services allow entries to be shifted between branches or levels in the tree-like structure. As of this, an entry's RDN is not guaranteed to remain unchanged. A widely adopted and popular standard for directory services is the Lightweight Directory Access Protocol (LDAP) [214] [215] [216], which is based on the ITUT standard X.500 [217]. LDAP is often used in corporate networks for employee or email directories.

Built as a service to resolve URIs in the web, the Domain Name System (DNS) [199] [218] has been designed as a decentralized directory service that organizes entries in a hierarchical fashion. The DNS allows users and services to resolve human readable domain names into IP addresses, from which requested documents or resources can then be requested directly, hence mapping identifiers to a network location. The DNS architecture uses a worldwide network of hierarchically organized DNS servers with DNS root servers functioning as the main authorities. The global domain space is categorized into zones, where authority for a zone can be delegated to servers. Each zone can comprise one or more domains, where information about resolvable domain names is stored in resource records (RR). Each RR specifies data required to discover the location of the resource to be accessed, being the host's IP address. The DNS root zone is maintained by the Internet Corporation for Assigned Names and Numbers[18] (ICANN), which delegates authority for zones specifying domains below top level domains (TLDs) to other servers and organizations. Information about TLDs is managed by DNS root servers operated by ICANN. Resolving a URL to the respective web server's IP address requires the client (DNS resolver) to recursively contact DNS servers starting at a root server, where a server will

[18] ICANN Website: `https://www.icann.org/`. Accessed: 23.8.2017

refer the client to the next DNS server by specifying its location. As this process is rather time consuming, caching strategies have been implemented to keep information about often contacted domains and servers in a local database. While LDAP and the DNS are built on a network of servers providing the respective service, both systems are build on a hierarchical design and ultimately require a central organization or company to maintain and exert control.

Peer-to-peer (P2P) architectures introduce several advantages compared to traditional client-server approaches [219]. Data stored and managed in a P2P network can be accessed by every participant in the network (sharing), while control over the network in terms of data, resources, and services is decentralized without a central authority (decentralization). In a P2P network, every participant is both client and server at the same time, allowing each node to specify the amount of resources it wants to make available to the network (self management). Due to the lack of a central authority orchestrating communication and message flow in the network, participating nodes need to organize themselves in terms of new nodes joining the network (self organization). Furthermore, unexpected failures and problems need to be automatically solved, for example when a node unexpectedly fails and the data and information managed by this node is not available anymore (self healing). As control of the system is distributed over all participating nodes, the architecture lacks a single point of failure and is hence resilient against attacks such as DDoS [220]. While an attacker could still attack a single node, functioning of the overall system is not impaired due to the self healing and organizing mechanisms of the architecture. Still, special attacks on P2P networks exists that exploit specific features of the attacked network structure [221].

While earlier generations of P2P networks, such as Gnutella [222] or KaZaA [223], connected nodes in a unstructured way, modern P2P architectures such as CAN [224], Chord [225], Pastry [226], or Kademlia [227] create structured overlays referred to as Distributed Hash Tables (DHT) to organize nodes. DHTs separate a large address space into smaller sections, where responsibility for each section is assumed by one of the participating nodes. Each node is assigned a unique address (key), which functions as the node's logical location in the DHT. To maintain connectivity in the network, each node further stores and manages a list of links to neighboring nodes as well as links to a small number of randomly selected, distant nodes called routing table. These links are utilized as shortcuts when traversing the network, effectively reducing the average path length in the network. This creates an overlay network disregarding physical connections or proximity of the participating nodes. To store information in a DHT, each data object to be managed by the P2P network is assigned a key, which is usually created by hashing the object's contents. The node responsible for the section of the DHT's address space, the object's key is located in, will then store the object or information regarding its retrieval. For retrieval of information, DHTs implement key-based routing (KBR) strategies where requests for a specified key are automatically forwarded within the network to the node responsible for handling the key. As the structure of a P2P network overlay usually exhibits a power-law distribution [228] [229], it is possible for

a query to traverse the entire network in just $O(logN)$ steps by following the links stored by the individual nodes. The Kademlia protocol [227] bases on a reactive KBR approach, which automatically manages and stabilizes a node's routing table. The protocol utilizes parallel, asynchronous lookups to reduce delays and furthermore compensate for failed nodes. This way, Kademlia-based systems are able to provide performant, robust, and resilient services that scale to high numbers of servers [230]. In consequence, scalable, robust, and efficient applications can be built based on DHT-based P2P networks [231]. For example, DHT-based alternative designs for the DNS were proposed by Ramasubramanian and Sirer [232], and Cox et al. [233]. The proposed solutions were able to provide a similar performance compared to the original architecture, but showed a far better resilience against orchestrated attacks [234].

3.5.2 Global User Identification

In order to provide unambiguous, yet decentrally issued and managed identification in the Sonic OSNF, users and platforms are identified via globally unique, domain-agnostic identifiers called GlobalID. GlobalIDs are designed to be domain and platform independent and remain invariant even when a user's OSN profile is moved to a new OSN platform and domain. Following this approach, a user's OSN profile can be identified and addressed regardless of the server and domain on which it is actually hosted on. With invariant, platform independent identifiers, a seamless migration of OSN profiles is made possible, where connectivity between different OSN profiles is maintained - even when the location of a user's OSN profile is changed frequently [4]. As GlobalIDs are issued without the support of a central authority, collisions with existing GlobalIDs need to be prevented. Similar to approaches used in UUIDs or Twitter Snowflake, GlobalIDs are hence created using cryptographic hash functions, rendering the creation of two identical GlobalIDs highly unlikely. To prevent potential attackers from illicitly claiming a user's identity, GlobalIDs are derived from an RSA key pair's public key, referred to as *PersonalKeyPair*. This allows the rightful owner of a GlobalID to prove ownership of the identity using the matching private key, which remains in the possession of it's owner.

Specifically, GlobalID's are derived from an `PKCS#8`-formatted RSA public key and a salt of 4 characters length using the key derivation function `PBKDF#2` with settings SHA256, 10000 iterations, and 256bit output length. The result is converted to base36, creating an alphanumeric string of upper case characters and digits (A-Z, 0-9) as described in Listing 3.1 in order to shorten the length of an identifier. Lower case characters are not used for GlobalIDs to reduce possible ambiguities, such as 1 and 1, and make the identifier visually distinguishable from regular hash values, which are commonly encoded using base64, including lower-case characters. The process of creating a GlobalID is depicted in Figure 3.5 and creates identifiers with a length of 50 to 52 characters. An example of a GlobalID is `2UZCAI2GM45T160MDN44OIQ8GKN5GGCKO96LC9ZOQCAEVAURA8`. As GlobalIDs are designed to be domain-agnostic by omitting any domain-bound party, they cannot be resolved via the DNS to a service providing information about a OSN profile's actual location. For this reason, all GlobalIDs are registered in a globally

distributed directory service, the Global Social Lookup System (GSLS), which provides an interface for applications and services to resolve a GlobalID to the actual location of the associated OSN profile. Information about the OSN profile's location, as well as other information required for verification of authenticity and integrity are stored in a dataset maintained by the GSLS called *Social Record*.

```
UALPHA      = %41-5A                ; (A-Z)
DIGIT       = %x30-39               ; (0-9)

GLOBALID    = 50*52(UALPHA / DIGIT)
```

Listing 3.1: GlobalID format in ABNF [235]

As GlobalIDs are encoded hash values, they are not human-readable or easily memorable and should hence not be used for displaying purposes. In fact, the respective human-readable display name specified in each Social Record should be used when displaying information about a user in the Sonic OSNF. In social networking, a link between two users is usually created by a user sending a friend or follow request to the targeted user profile. This is usually the case after the targeted user profile has been found as a result of a directed search request or because a recommendation system recommended the targeted user profile based on certain criteria. Also, the targeted user profile might be discovered by browsing a common friend's friend roster, a list of members of a group, or viewing content created by the targeted user. In all these cases, the targeted user profile is usually displayed by showing its human readable display name and optionally the profile picture. Furthermore, additional information, such as the city of residency or country can be displayed. Hence, handling the GlobalID identifiers is not necessary for users. While the displayable information is not necessarily unique and may lead to confusion, GlobalIDs are used by the service internally to uniquely identify each user.

The Social Record Dataset

The GlobalID and associated information is published in a dataset called Social Record, which comprises information that is required in order to resolve a GlobalID to the respective profile's current location in form of a URL. Besides the profile's actual location, the Social Record dataset comprises information such as the public key of the PersonalKeyPair and the salt, which are used to construct the GlobalID; the GlobalID itself as well as the GlobalID of the OSN platform hosting the OSN profile; the public key of the AccountKeyPair, which is used to verify integrity and authenticity of request and response messages as well as

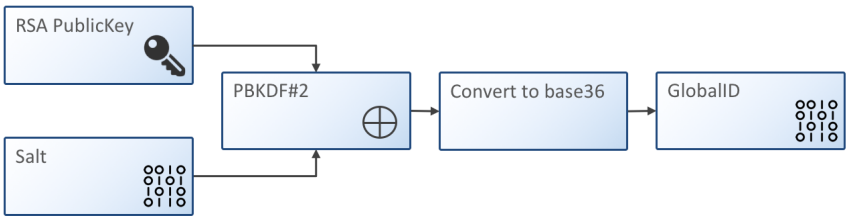


Figure 3.5: Creation of GlobalIDs.

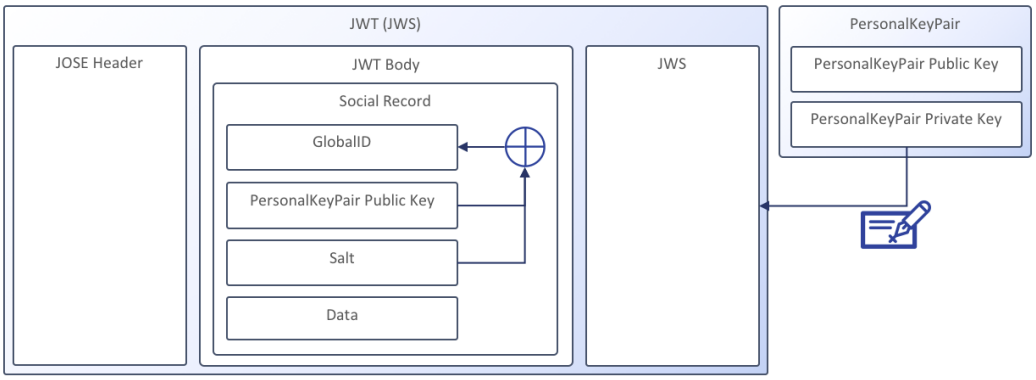


Figure 3.6: Social Record format

content objects created by the respective user; a list of revoked AccountKeyPairs; a timestamp in XSD-datetime format indicating the date and time when the dataset was edited; the user's username for displaying purposes; and a parameter `active`, which is used for announcing the state of the Social Record, for example during an ongoing profile migration as described in Section 3.5.4. The individual parameters of the Social Record dataset are described in Table 3.3.

To make a Social Record's integrity verifiable, each Social Record dataset is digitally signed using the owner's PersonalKeyPair. This allows OSN platforms and users to verify that the dataset's contents have not been altered by an unauthorized party. A Social Record's signature is a JSON Web Signature (JWS), encoding the dataset as a JSON Web Tokens (JWT) [236] using the `RS512` JSON Web Algorithm (JWA) to digitally sign the payload using the dataset owner's PersonalKeyPair. The Social Record data itself is a private claim named `socialRecord` and has to be a serialized, base64-encoded JSON object. As the digital signature of the JWT is created using the user's RSA private key of the PersonalKeyPair, so the signature can be verified by everyone using the respective public key that is included in the signed dataset. The resulting JWT is a base64-encoded string [237] that can be easily transfered and validated. Figure 3.6 depicts the format of the Social Record encoded as a signed JWT. The Social Record object itself is encoded as a JSON object and stored in the body part of the JWT. Here, the enclosed RSA public key of the user's PersonalKeyPair is used in combination with the salt to derive the GlobalID via PBKDF#2. The JWT's JOSE header specifies the used signature algorithm, while the digital signature itself is created with the matching RSA private key of the user's PersonalKeyPair.

In case that the user's AccountKeyPair has been compromised and should be revoked, a key revocation certificate can be created. Following the Certificate Revocation List (CRL) specifications of X.509 [238], the created key revocation certificate comprises the encoded revoked public key, date and time of the revocation, a numerical indication of the reason for the revocation, and a digital signature. All revocation certificates are published in the Social Record, while the outdated AccountKeyPair is replaced with a new one. As the entire Social Record dataset including the list of revocation certificates is signed by the user's PersonalKeyPair, the revocation and exchange of a AccountKeyPair can be verified by any third party.

Table 3.3: SocialRecord format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`socialrecord`" specifying the object type.
type	String	Required. Type of the Social Record. Can be `user` or `platform`.
globalID	GlobalID	Required. The identifier for the user profile.
platformGID	GlobalID	Required. GlobalID of the associated OSN platform.
displayName	String	Required. Human-readable username for on screen display.
salt	String	Required. Cryptographic salt of 4 byte length.
accountPublicKey	String	Required. Base64-encoded RSA public key of the user's AccountKeyPair.
personalPublicKey	String	Required. Base64-encoded RSA public key of the user's PersonalKeyPair.
datetime	XSD-Datetime	Required. Date and time of the creation or last change of the Social Record.
keyRevocation List	KeyRevocation Certificate	Required. List of revoked account key pairs.
active	int	Required. Flag that describes the current status of the Social Record.

Table 3.4: KeyRevocationCertificate format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`keyrevocationcertificate`" specifying the object type.
key	String	Required. The revoked public key of an AccountKeyPair.
datetime	XSD-Datetime	Required. Date and time of the creation of the certificate.
reason	int	Required. Numeric value for specifying the reason of revocation as specified by X.509 [238].
signature	String	Required. Base64-encoded signature created with the user's PersonalKeyPair.

Security Considerations

As GlobalIDs are generated in a distributed fashion without a trusted central authority preventing an intended or unintended creation of a collision, uniqueness of GlobalIDs needs to be ensured by other means. As per design no central trusted authority exists that could assert identities in a controlled fashion, the difficulty of creating an already existing GlobalID again is the only guarantee that identifiers are unique and collisions cannot occur. GlobalIDs are created by hashing an RSA public key using SHA256. Hence, an attacker would

either need to recreate the key pair or create a collision with a combination of a key pair and a salt that yields a known GlobalID. The probability of creating a collision for a specific or arbitrary GlobalID is therefore limited to a minimum by design, where the likelihood of unintentionally created collisions for a GlobalID are insignificantly low given the assumption that a secure RSA key generator is used. Hence, in the remainder of this section only intentional attempts to create a collision for a GlobalID are considered as attack scenarios. For every attack scenario, the information available for an attacker is the data stored in a Social Record dataset, specifically the PersonalKeyPair's public key, the 4-byte salt, the resulting GlobalID, and the digital signature that has been created for the JWT encoding.

An attacker's goal could be to entirely take over a user's social identity in order to be able to act on his or her behalf in the OSNF. This would require the attacker to either update the Social Record or overwrite the entire Social Record dataset in the GSLS with a valid alternative. As Social Record datasets are digitally signed, unauthorized changes of parts of a Social Record would result in invalid signatures. Attack scenarios on a user's identity hence require an attacker to create a new valid signature for a forged Social Record.

Deriving an RSA private key from mere knowledge of the public key is considered not to be feasible [239]. An attacker could therefore alternatively attempt to exchange the PersonalKeyPair's public key in the Social Record with an arbitrary new key pair he created for this purpose. Yet, simply replacing the PersonalKeyPair's public key itself would create a new, distinct identity in the GSLS, as a changed key would result in an invalid Social Record dataset. Updating the GlobalID to prevent the dataset from being rendered invalid would result in a changed GlobalID. This is due to the fact that the GlobalID is directly derived from the PersonalKeyPair's public key, while being used as the lookup key in the GSLS. In consequence, a changed key and GlobalID would not overwrite or change an existing Social Record dataset in the GSLS, but merely create a separate, valid identity. The attack would hence be deflected.

Another attack scenario is prevented by the use of a cryptographic salt value. Without the use of a salt, an attacker could build a huge database of all GlobalIDs and associated keys in the GSLS in order to attempt to break any GlobalID in the database via brute force. The attacker would then attempt to create a collision for any GlobalID in the database by creating a large number of RSA key pairs until a created key pair matches any one of the GlobalIDs in the database. This attack could be further improved by creating lists of key pairs following the idea of rainbow tables. While this attack scenario would not allow to target a specific GlobalID, it's probability of success increases with the total number of attacked Social Record datasets. Anyhow, this attack essentially attempts to re-create a 4096-bit RSA private key by brute force which has been proven to be practically impossible [239]. Anyhow, as generating an RSA key pair is the most time consuming task in creating a GlobalID, an attacker might choose one key and just alter the salt in oder to find a collision. To limit the possibility of this attack to succeed, the length of the salt has been fixed to 4 bytes, allowing only 2^{32} possible salt values, effectively eliminating the chance of creating a collision

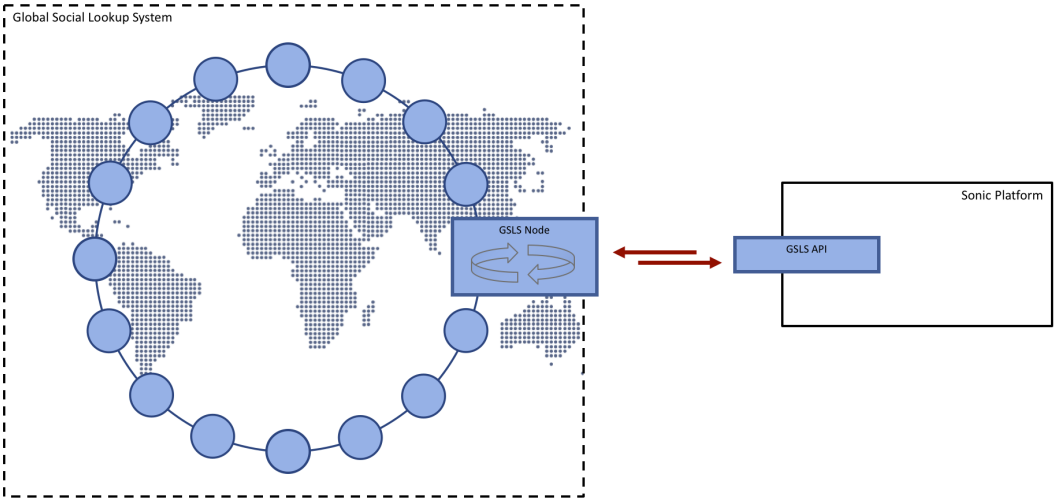


Figure 3.7: Global Social Lookup System architecture

through manipulation of the salt. Using the birthday bound [240], an attacker would need to create 4.8×10^{37} key pairs and salts for a 1% chance of a collision, thus rendering an attack practically impossible. The introduction of an individual cryptographic salt value into the process of how GlobalIDs are derived from a key further hardens the process, as the salt is different for each simultaneously attacked GlobalID.

Finally, an attacker could attempt to steal the PersonalKeyPair from a user's device by acquiring either physical or logical access to it, or use social engineering to obtain a copy of the key pair voluntarily provided by the key pair's owner himself. Due to the computational overhead required to successfully forge a valid Social Record dataset, stealing the key or persuading the owner into handing it over to the attacker would probably be the attack vector with the highest success rate. Anyhow, Sonic cannot provide security mechanisms to prevent such attack scenarios, as they solely depend on the user and the user's device.

3.5.3 Global Social Lookup System

Following the idea of a fully decentralized OSN ecosystem that does not depend on any entity or service controlled by a single corporation or group, the GSLS was designed as a directory service built on DHT technology. Similar to the DNS, any participant in the Sonic ecosystem is able to host a GSLS server that is automatically integrated into the DHT, forming a dynamic, heavily distributed directory service. The GSLS operates as a global directory service with a REST-based interface for read and write operations as described in Table 3.5. As data in the GSLS is public and may be overwritten by unauthorized entities, the data is digitally signed using the user's PersonalKeyPair. As the GlobalID is derived directly from the enclosed public key and the salt, unauthorized changes in the payload would result in either an invalid digital signature or – in case the key pair is exchanged – an altered GlobalID.

GSLS API

For resolving GlobalIDs of users and platforms as well as for creating and updating Social Records in the GSLS, the service features a REST-like API as described in Table 3.5. Social Records for specific GlobalIDs can be retrieved via HTTP GET by specifying the respective GlobalID. As Social Records are publicly accessible, the API does not need to implement access policies. If a Social Record exists in the DHT for the specified GlobalID, it is sent to the requesting client encoded as a JWT. The client receiving the signed JWT can then extract the Social Record and verify the digital signature of the JWT as well as the integrity of the Social Record itself.

When a new Social Record is to be created and written to the GSLS, the user identified by the GlobalID creates a valid Social Record dataset. The Social Record is then encoded as a JWT, which is signed using the user's PersonalKeyPair. The signed JWT is then sent to the GSLS via HTTP POST, where the receiving node verifies both the digital signature and the Social Record's integrity. If the enclosed Social Record dataset and the JWT's signature are valid, the GSLS node will store JWT in the DHT using the GlobalID as it's key for retrieval. In case a Social Record is already stored for this GlobalID, the signature is not valid, or the Social Record dataset's contents are not valid, the GSLS server will return an error.

Similar to writing a new Social Record to the GSLS, updating an existing Social Record that is already stored in the DHT is implemented via HTTP PUT. Similar to creating a new Social Record, a user creates a valid Social Record dataset with the updated information, which is then encoded as a JWT and signed using the user's PersonalKeyPair. When a GSLS node receives an updated Social Record, it checks whether dataset and signature are valid and if the GlobalID has already been registered. The JWT is then stored in the DHT, effectively overwriting the old version of the JWT.

Finally, the GSLS allows to request a status report for diagnostic purposes via HTTP GET, where the requested node's version information as well as it's connectivity status in the DHT network is returned.

Table 3.5: GSLS API

Method	Path	Parameter
GET	/	N/A
GET	/:globalID	N/A
POST	/	Social Record as JWT
PUT	/:globalID	Social Record as JWT

3.5.4 Profile Migration

Identifying users via domain-agnostic, globally unique identifiers allows addressing OSN profiles independently of the service and domain they are stored in. As GlobalIDs are resolved to profile locations via the GSLS, a change of a profile's location only needs to be updated in the respective Social Record dataset. Initiating a connection to a target OSN profile requires the sender to resolve the targets GlobalID via the GSLS. From the result of the request, the sender is able to extract and verify the actual profile location in form of a URL, to which messages then can be sent directly. If the profile location is changed in the Social Record dataset, all communication attempts are automatically redirected to the targeted OSN profile's new profile location. As described in Section 3.6.4, Sonic URLs are used for links to OSN profiles and content in the Sonic OSNF, which comprise the respective owner's GlobalID instead of it's actual URL. When a request is sent to a social profile following the Sonic protocol as described in Section 3.7, the GlobalID is automatically substituted with the OSN platform's URL. Following this paradigm for resolving links to OSN profiles, links to a user's OSN profile and content may remain unaltered and valid after a profile is moved to a new OSN platform hosted under a different domain name.

This enables migration of OSN profiles between OSN platforms and services with a minimal overhead, while keeping links between OSN profiles intact at all times. Migrating a user's OSN profile from a source OSN platform P_A to a target OSN platform P_B is initiated by the respective profile owner by creating an empty stub account at the target OSN platform. Following the Sonic protocol for migration as described in Section 3.7, first the Social Record is updated where the value of the parameter `active` is set to `2` to indicate that the OSN profile is currently being migrated to a new location. This enables OSN platforms and clients sending a request to the OSN profile that is being migrated to detect an ongoing migration. As the OSN profile contents are moved to the target OSN platform in the migration process, non-idempotent requests that create, update, or delete content in the OSN profile being migrated could result in loss of data or inconsistencies. Hence, only idempotent requests for reading content should be allowed and answered during an ongoing migration, while all other requests should be denied by the OSN platform with an error message indicating that a profile migration is in process and the request should be sent again later. The next step of the migration process comprises transmitting all profile content to the target platform P_B, where data is encapsulated according to the data formats described by the Sonic protocol. In case no error occurred during the entire process, the profile migration is concluded through an update of the Social Record where the profile's location is updated to point to the target OSN platform P_B and the value of the parameter `active` is reseted to `1`. Requests targeting the OSN profile are now automatically directed to the new profile location at the target OSN platform. To finalize a successful migration, all profile data is deleted from the source OSN platform P_A, while in case of a failed migration, all content is deleted from the target platform and the profile's location remains unchanged in the Social Record. The individual steps and details of the migration process are described in Section 3.7.

3.6 Architecture

Today's most common type of architecture for web services in general and for OSN services in particular is a centralized one. While the majority of OSN services is built in a centralized way, components of these services are often in themselves distributed [111], caused by an increasingly complex internal setup [48]. The architecture of Facebook, for example, *"consists of multiple datacenter sites and a backbone connecting these sites, [...] where each datacenter contains multiple clusters"* [241] [242]. As the dataset comprising the entire social graph of Facebook is simply too large to be handled by a single instance, "The Associations and Objects" (TAO), a read-optimized graph data store is used to *"distribute [the dataset of the social graph] as a single geographically distributed instance"* [243]. Yet, while being technically highly distributed geographically and logically, services such as Facebook only expose a single or few domains and APIs to their customers and thus create a perception of the service as one monolithic entity. As noted by Paul et al., this allows centralized OSN services to provide a holistic, single service for its users as well as third party services and applications, where control over all components is held and managed by one company [111]. Yet, as described in Chapter 2.1.2, this form of decentralization is simply of technical nature instead of an authorial one. According to the definition of Paul et al. given in [111], a DOSN service is *fully decentralized* when all basic functionality of a DOSN service does not rely on a centralized component and *"distinct and independent authorities run and maintain [all] technical resources"*.

For the design and implementation of web services, several requirements have been defined by the World Wide Web Consortium (W3C) [244], including interoperability, integration with the World Wide Web, security, scalability, and extensibility. Similarly, the FIArch group identified 5 design principles for the design and implementation of novel Internet services and applications in their report on "Future Internet Design Principles" [245]. Theses principles comprise heterogeneity, scalability, robustness, loose coupling, and locality. Here, the principle of heterogeneity describes that different devices, programs, or services are connected in an open architecture, while scalability is defined as the ability of a service to support large numbers of nodes and users. Robustness dictates that a service needs to be able to deal with any kind of error that occurs, while the principle of locality proposes to store data where it is needed instead of remote locations in order to provide fast response times and less overhead. Finally, loose coupling allows simple testing and maintenance procedures, and a minimized unwanted interference with other systems and components.

Existing DOSN approaches, such as Diaspora, Peerson, or Friendica, aim to create fully distributed OSN services and are built in a way where functionality as well as control over data is distributed over a larger number of servers or devices. In these architectures, functionality is distributed over a large network of servers, while data is kept and maintained by the respective server's administrator or device owner. All servers or devices in such a DOSN service cooperate and communicate using a common protocol, using certain standards and data formats to exchange information and data. Yet, as pointed out in

Chapter 1.5 and analyzed in Chapter 2.3, while this allows users registered on different servers to freely communicate with other users of the same DOSN service, users are still mostly confined in the DOSN service they signed up with and cannot migrate in most cases to an alternative OSN service if needed.

Following the vision of a truly open and decentralized OSNF, as described in Chapter 1.7, in which multiple OSN services are connected via loose coupling to form a heterogeneous federation, an OSN ecosystem requires an underlying standard for how OSN platforms communicate and how information and data objects are exchanged. In order to allow arbitrary OSN services to connect to each other, an architecture needs to satisfy the architectural significant requirements for an OSNF as defined in Section 3.1. These requirements comprise a non-intrusive design (Requirement R1), support for decentralization and federation of participating services (Requirement R3), distributed control and management of data and social profiles (Requirement R4 and R5), seamless and transparent communication and interoperability (Requirement R6), open protocols and data formats (Requirement R7), as well as support for migration of user profiles (Requirement R8).

As described in Chapter 1.5, Koll et al. defined a list of challenges distributed OSN services would have to face if they want to prevail on the market [101]. The requirements stated include independence of any external resource providers, being efficient in terms of communication and storage overhead, providing scalability for a large user base at a reasonable performance, being resilient against threats that could destabilize the network, as well as providing means for users to exert full control over their data. While Koll et al. acknowledge that not all stated requirements can be fully satisfied at the same time, they conclude that a distributed, federated approach with OSN functionality and data hosted on the users' servers or home gateways would provide a promising way to combine both performance of the individual nodes and independence from any central entity. The feasibility of this approach has been proven by Salve et al., who analyzed data availability in distributed data stores [246].

While hosting one's OSN profile on a private server already exists in F-OSN services such as Friendica or Diaspora, Sonic aims to create a federated environment, which is able to interconnect all kinds of OSN services. The architecture of Sonic is therefore designed to allow arbitrary existing OSN services to join the OSNF by integrating the required components to connect to other OSN platforms in the federation. A set of common data formats as described in Chapter 3.7 ensures that content can be exchanged between OSN platforms using the Sonic protocol. Following this approach, existing OSN platforms with large numbers of users can directly communicate with small self-hosted OSN platform implementations. This way, individual users who want to host their own OSN profile in their home network or on a private server, are able to connect to the OSNF equally as large OSN services with millions of users.

The Sonic architecture therefore introduces a set of components that implement connectivity in the OSNF, which can be integrated into an existing OSN implementation with a minimal overhead. This way, existing OSN services are able to implement the Sonic architecture and thus connect to the OSNF

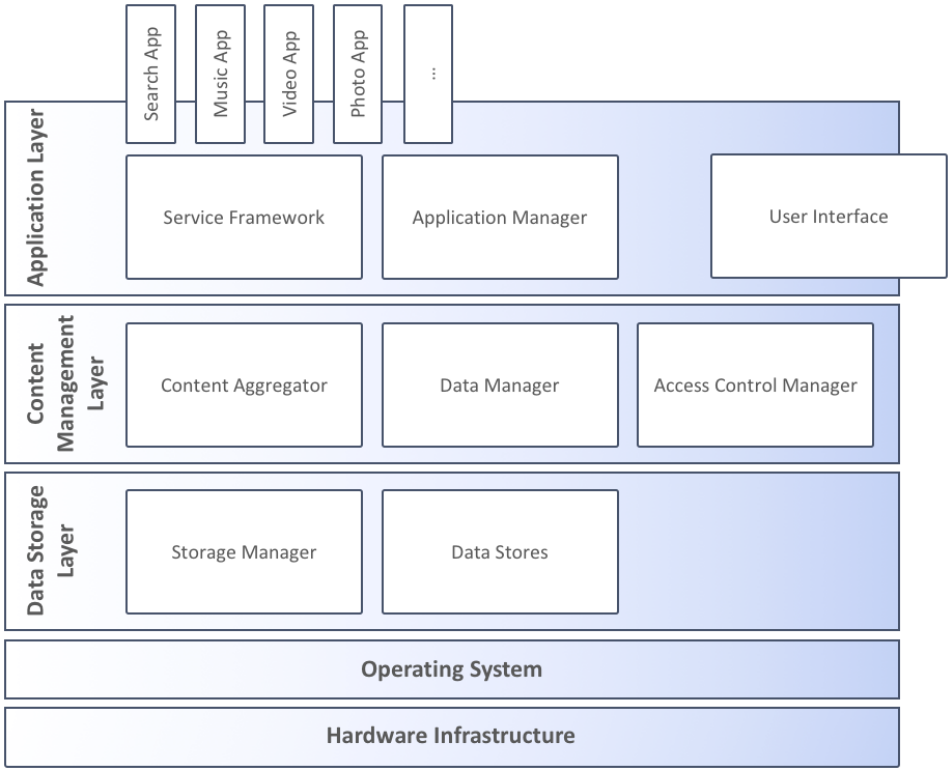


Figure 3.8: Architecture of DOSN services as described by Pallis et al. [122]

without the need to change their unique user experience, user interface, or underlying architecture.

This section presents the Sonic architecture, its components as well as concepts of how features of Sonic are implemented. Section 3.6.1 elaborates on related work related to the architectures of social web services, while Section 3.6.2 describes the architecture of the Sonic OSNF, including the role model, content model, and access control mechanisms.

3.6.1 Related work

Following the definition of Boyd and Ellison given in [24], OSN services are web services in general, which are defined by the W3C as *"[...] a software system designed to support interoperable machine-to-machine interaction over a network"* [247]. According to the definition, the purpose of a web service is to provide a specific kind of functionality on behalf of its owner, referred to as the *provider entity*. A web service is then consumed by a *requester entity*, being a person or organization that wants to access and use the web service's functionality. Here, communication is implemented via messages exchanged between requester and provider entity, where both parties *"must first agree on both the semantics and the mechanics of the message exchange"* in order for the messages to be correctly interpreted by the receiving side [247].

While a trend can be observed according to which OSN services are becoming increasingly complex and less structured [48], several general architectures for DOSN services have been proposed that simplify and generalize OSN

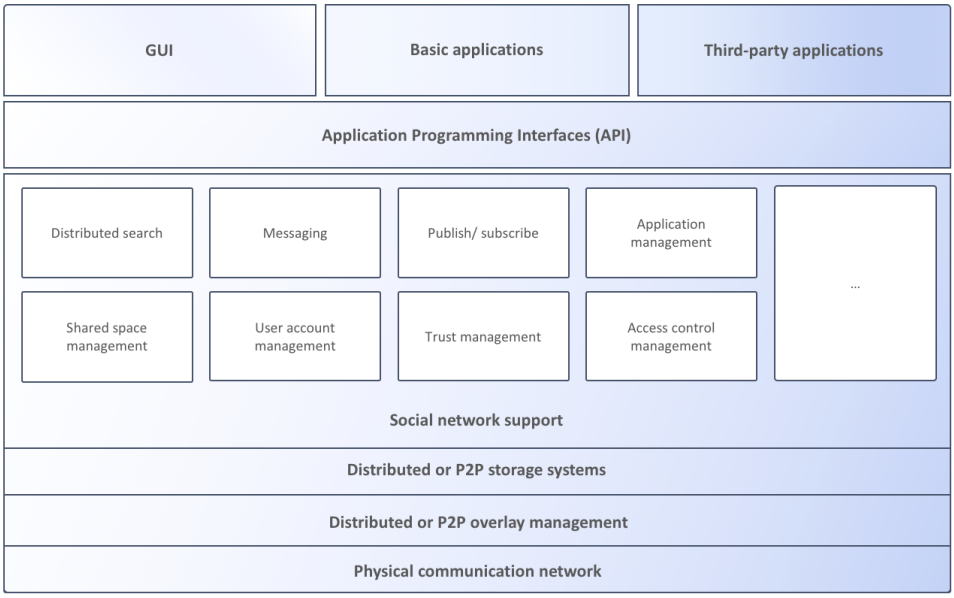


Figure 3.9: Architecture of DOSN services as described by Datta et al. [88]

architectures, which are loosely based on the n-tier architecture pattern as described by Richards in [248].

In [122], Pallis et al. describe a reference architecture of OSN services comprised of three main layers as depicted in Figure 3.8. On top of the basic layers that describe *hardware infrastructure* and the *operating system*, the *data storage layer* manages persistence of data objects of an OSN service. The data storage layer comprises two components, of which the *storage manager* controls storage of information while the *data store* component comprises the technical means to store information, for example in form of a database. The second layer, the *content management layer* comprises three distinct components, being a *content aggregator*, a *data manager*, and an *access control manager*. While the *content aggregator* acquires and distributes social content, the *data manager* facilitates maintenance and retrieval of the social content graph. In addition, the *access control manager* controls access to the contents of OSN profiles according to access control policies. On top, the *application layer* provides services and functionality to access, consume, or create data. In their architecture definition, all functionality such as search, news feeds, media collections, or mobile access to the service are encapsulated in separate services, which communicate with the underlying content management layer in order to access and manipulate information stored in the social content graph [122].

Datta et al. defined a reference architecture for DOSN services in [88] comprising six layers as depicted in Figure 3.9. While the lowest layer describes the *physical communication network*, the *distributed or P2P overlay management layer* provides functionality for connectivity of peer nodes in a P2P overlay or servers in a federation. The third layer implements functionality for querying, writing, and updating content in the distributed service. All core OSN functionality is implemented in the *social networking layer*, including search functionality, messaging, user account management, or access control management. On top of

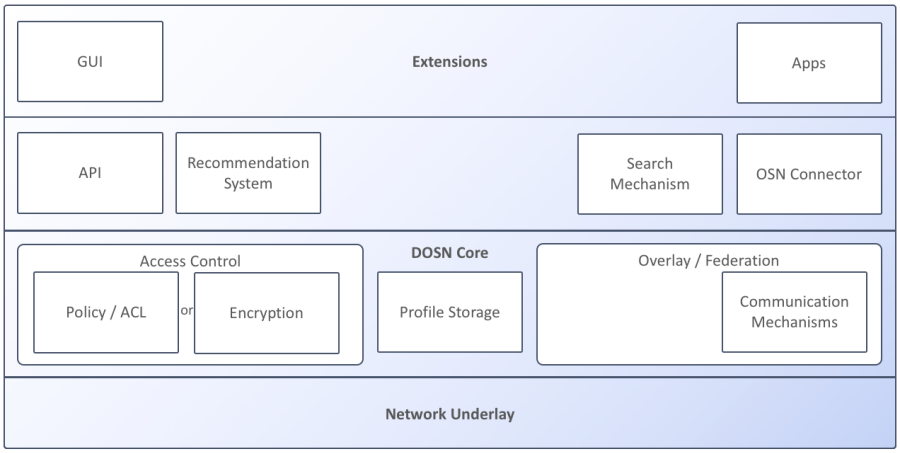


Figure 3.10: Architecture of DOSN services as described by Paul et al. [111]

the social networking layer, a DOSN service implements a set of APIs for external services and applications, preferably conforming to common open standards. These APIs are used on the top layer by the OSN service's GUIs in form of apps or websites, as well as basic OSN applications and third party applications [88].

Paul et. al describe a general architecture of DOSN services, which they derived from their survey of DOSN services in [111]. As depicted in Figure 3.10, the architecture model describes three layers, of which the lowest layer represents the communication network. The *DOSN core* layer provides all essential functionality of the DOSN, including access control management, profile storage, and communication mechanisms in a federation or overlay network. The access control components may be implemented in three different ways, being access control policies or encryption schemes, or a combination of both. The component responsible for communication organizes the exchange of messages between the different nodes or servers in the DOSN service, including protocols for direct user interaction as well as support for technical information exchange. Finally, the profile storage component manages the persistence of all OSN profiles including all associated data. The top layer is again divided into two sublayers, of which the first sublayer provides APIs for connectivity of third party applications and services, and additionally an OSN connector component that implements connectivity to other OSN services in form of plug-ins. Furthermore, the layer implements search and recommendation functionality that allow users to search and discover content and user profiles. The upper sublayer implements GUIs for the user of the DOSN service to access and use as well as applications that communicate with the DOSN service via the provided APIs [111].

3.6.2 Sonic Architecture

The architecture of the Sonic OSNF is an open and heterogeneous federation of loosely coupled OSN platforms, where all OSN platforms are connected via the Sonic protocol. This protocol specifies a common API as well as data formats as described in Section 3.7 and thus facilitates interoperability in the OSNF. In this open federation, a user can register at an arbitrary OSN platform, set up an OSN profile, and publish content. Content and information is then accessible in the entire OSNF via the federation of OSN platforms. To access their accounts and content in the OSNF users employ OSN client applications, such as websites or smartphone applications, in order to access the OSN platform they registered with. Content from other OSN profiles is then fetched and accumulated from all OSN platforms in the OSNF, allowing each OSN platform to provide content from all sources in the OSNF to their users. As the idea of the Sonic OSNF is to facilitate seamless interoperability between existing OSN services, the architecture design of the Sonic OSN platform is designed to allow existing services to implement necessary components of the architecture without being forced to change the platform's overall look and feel or QoE.

The architecture of the OSNF describes three distinct entities, being OSN platforms, OSN clients, and GSLS servers. OSN platforms, as defined in Chapter 3, are specific instances of an OSN service which implement the functionality of an OSN service and furthermore store and maintain OSN profiles of one or multiple users. Users in the Sonic OSNF register with a specific OSN platform, where multiple users may register with and use the same OSN platform simultaneously. For accessing a OSN platform's service, OSN platforms provide OSN clients, for example in form of a web page or mobile application, that allow users to access and use the OSN platform's functionality and manage user accounts. Here, OSN clients may access the respective OSN platform's functionality via a proprietary and platform-specific API provided by the OSN platform for this specific use. Based on the Sonic protocol as described in Chapter 3.7, OSN platforms are connected to each other in a loosely coupled fashion. This allows OSN platforms to directly communicate with each other and exchange information about users. Finally, the architecture of the Sonic OSNF comprises a network of GSLS servers as described in Chapter 3.5, which provide information about a user's identity in the OSNF. Figure 3.11 depicts the interworking of the components of the Sonic OSNF.

Sonic OSN Architecture

The Sonic architecture adopts the general idea of a layered DOSN architecture as proposed by Datta et al. in [88], Pallis et al. in [122], and Paul et al. in [111]. For this, the Sonic architecture adopts layers for presentation, business logic, and data storage, and furthermore adds a federation layer that manages connectivity in the OSNF as well as an identity layer that implements functionality for user and object identification. The resulting five-tier architecture, as depicted in Figure 3.12, is loosely based on the n-tier architecture pattern as described by Richards in [248], in which each layer encapsulates functionality for a specific

domain and communicates with components implemented in other layers if necessary.

The *Presentation Layer* implements communication functionality with OSN clients and third-party services. For allowing users to access the service and interact with its functionality, the presentation layer manages communication with OSN clients, which render the graphical user interface (GUI) of the OSN service. The most common forms of user interfaces for OSN services are websites that ace accessible via HTTP to be displayed in a web browser on the client's device or mobile applications. Furthermore, the presentation layer may feature additional APIs to communicate with other services, for example to integrate OSN elements in third-party web services or to allow external applications to access the functionality of the OSN service. The design of a Sonic OSN platform does not mandate which additional types of interfaces a OSN platform provides. This is to allow any form of service delivery to the end user, especially allowing existing OSN services to integrate the Sonic OSN service architecture into the existing architecture without changing existing core service features or the QoE experienced by the user.

The *OSN Layer* implements the core functionality of the OSN service. Most importantly, the OSN layer implements the business logic of the OSN platform that is required for the service to function, including functionality for acting on requests received from the federation layer, or performing an action requested by the user accessing the OSN platform's GUI via an OSN client. This includes the program logic for handling all supported OSN features as described in Chapter

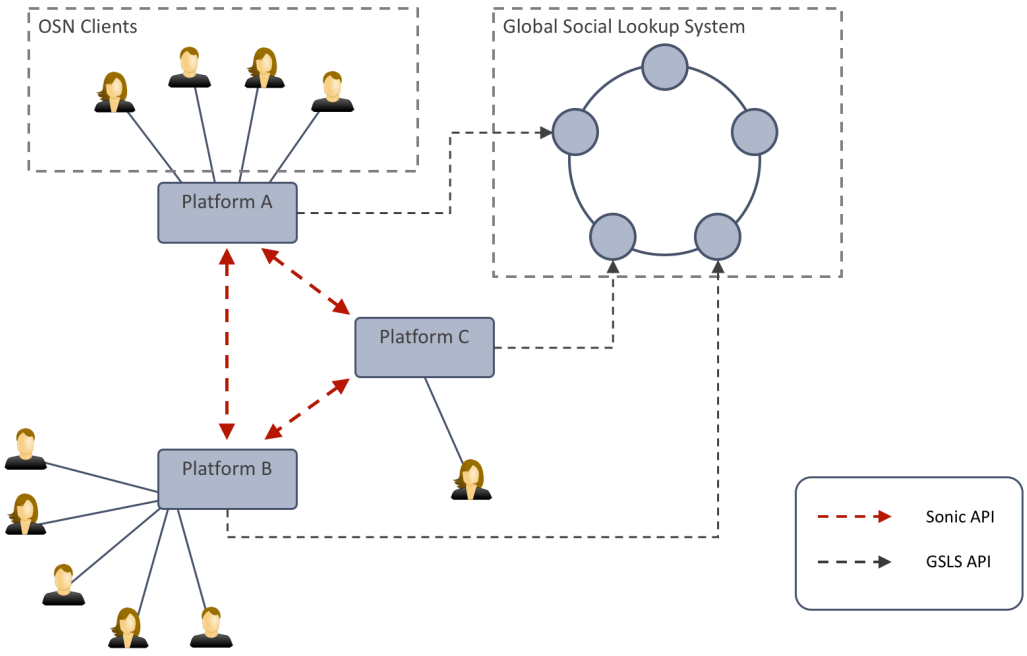


Figure 3.11: Architecture of the Sonic OSNF. Users utilize OSN client applications to connect to OSN platforms, on which their OSN profiles are stored. All OSN platforms are connected in a loosely coupled fashion based on the Sonic protocol. Management of identities in the Sonic OSNF is managed by the distributed GSLS, to which OSN platforms connect via the GSLS API.

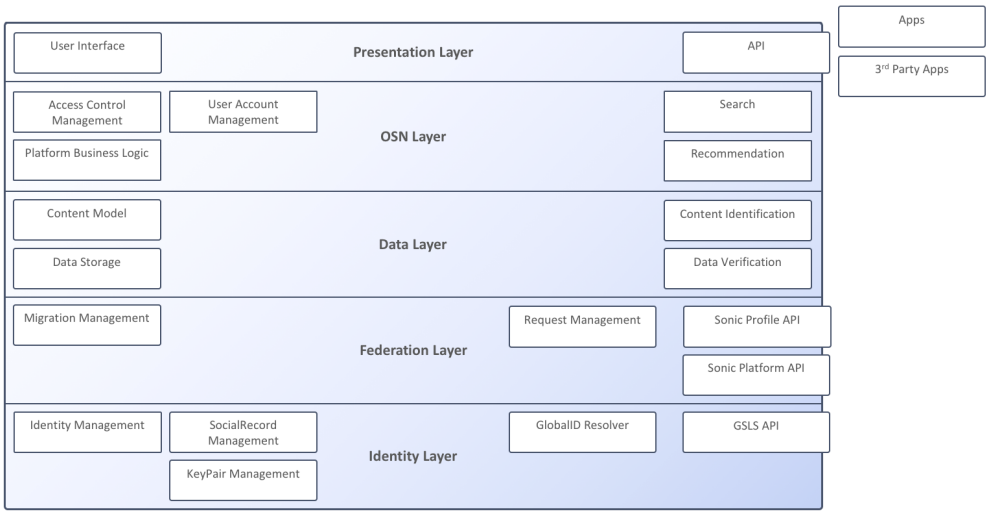


Figure 3.12: 5-Layer model of the Sonic architecture

3.4, such as status updates, comments, profile pages, or likes. Furthermore, the OSN layer provides functionality for user account management, which allows users to configure and manage their user accounts on an OSN platform server, for example by configuring privacy settings or access control rules. Besides this, the OSN layer implements functionality for user account management as well as access control functionality, allowing users to specify policies of who can access what parts of their OSN profiles. The access control component implements functionality to verify an incoming requests' permission to access the user's content. To allow OSN platforms in the Sonic OSNF to implement various forms of access control as described in Section 3.6.5, Sonic does not marshal how access control mechanisms are implemented, allowing OSN implementations to keep using existing solutions. Finally, the OSN layer implements search and optional recommendation mechanisms [6], allowing users to actively search for users and content, as well as receive recommendations from the OSN platform.

The *Data Layer* implements handling of data objects including persistence of data. This includes functionality for serialization and deserialization of content objects, as well as functionality for verification and validation of content received from remote, potentially untrusted OSN platforms. Furthermore, the data layer optionally implements functionality to transcode Sonic content objects into the OSN platform's native data formats and Sonic-formatted content objects into their respective native formats. As depicted in Figure 3.13, this allows existing OSN implementations to keep using their own data formats for content objects and information, while being able to exchange content with other OSN platforms in the OSNF in a common data format.

Interoperability with the Sonic OSNF is implemented in the *Federation Layer*, which implements the functionality required to connect an OSN platform to other OSN platforms in the Sonic OSNF. Most importantly, the federation layer implements the Sonic Profile API and Sonic Platform API as described in Chapter 3.7, to which requests from remote OSN platforms can be directed. Incoming requests are verified by the request management component implemented in the federation layer, which inspects the integrity of incoming request messages

via the enclosed digital signatures as well as the general formatting of messages. Valid requests are then parsed by the data layer and forwarded to the business logic implemented in the OSN layer for processing. Finally, the federation layer implements functionality to build, format, and sign outgoing request messages in order to allow OSN platforms in the OSNF to communicate with each other. Migration functionality is implemented in the migration component, which orchestrates the migration of an entire OSN profile from one OSN platform to another. Here, the migration component implements functionality to encapsulate all data of an OSN profile in a migration data format as described in Chapter 3.7, which allows to recreate the entire OSN profile at a different OSN platform after the information has been transferred.

The *Identity Layer* encapsulates functionality for the identity management, including communication with the GSLS. Specifically, the identity layer implements functionality to resolve GlobalIDs to the associated Social Record datasets via the GSLS API, as well as functionality for caching and verification of the datasets. By resolving GlobalIDs, an OSN platform is not only able to locate remote OSN profiles of users and verify their identity, but also to translate Sonic URLs to regular URLs, which can then be used to identify and locate content objects in the OSNF as described in Section 3.6.4. As requests and data objects in the Sonic OSNF are digitally signed in order to ensure integrity and authenticity of data objects and requests, the identity layer implements functionality to handle a user's key pairs for signing and for verification of content and requests messages as part of managing SocialRecord datasets of users.

The proposed architecture model for OSN platforms in the OSNF has been designed to be easily integrable into existing OSN implementations as well as new projects. While the general architecture builds on architectural models described by Datta et al. [88], Pallis et al. [122], and Paul et al. [111], functionality for federation support and identification have been introduced in the federation and identity layer. By implementing the functionality covered by the respective components, OSN services can provide interoperability and data portability in the OSNF. To further support an easy implementation of the required components, a PHP-based SDK has been developed that implements the required functionality. Implementation and structure of the Sonic SDK are described in Chapter 4.

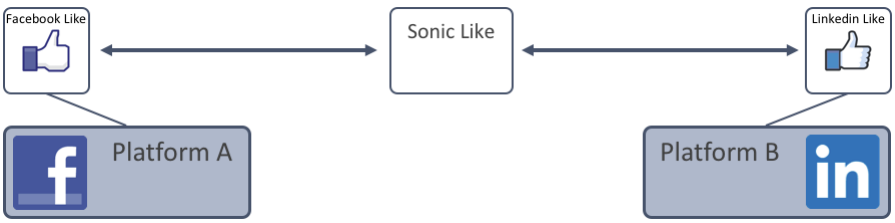


Figure 3.13: Transcoding of a like object in between two OSN platforms using proprietary data formats. While Platform *A* internally uses its own data format of a like, likes are disseminated to other OSN platforms in the appropriate Sonic data format as described in Chapter 3.7. If required, other OSN platforms can then transcode the received likes into their own representation.

3.6.3 Relationship Model

OSN services implement various forms of link relationships between users. The most common form are bidirectional links indicating a mutual friendship or acquaintance between two individuals as described in Chapter 3.4. Establishing such a link usually requires that one user sends a *friend request* to another user, who can then accept, reject, or simply ignore the request. If the request is accepted, both users are added to the respective other's list of friends. This friend list can then be published in their OSN profiles. While bidirectional links are used as connections between users by most traditional OSN services such as Facebook, Linkedin, or VKontakte, other OSN services implement unidirectional links between users. Unidirectional links are mostly used to indicate that a user is 'following' or 'interested in' another user's profile and content, and hence often implies a subscription to content created by the followed user. For example, Instagram, Twitter, or Diaspora implement a unidirectional link model, where two separate, opposing unidirectional links can be created to construct a bidirectional connection. Unidirectional links are often allowed to be created without the followed user approving a request. This approach either relies on the idea that 'everything is public', as for example implemented by Twitter[19], or assume that a user's privacy settings or access control policies dictate what content is made available to which followers.

In order to provide a design for relationship management that is compatible with all OSN platforms, links in Sonic are designed as unidirectional connections. Each connection needs to be requested by a user and confirmed by the user targeted by the link request. To prevent users from creating forged link rosters and falsely claim that they are connected to other users, links are represented by digitally signed data objects, where the user targeted by a link request creates the signature of the created link object. This prevents users from illegibly claiming that another user, for example a celebrity, is a friend. Functionality for the creation of and access to remote link collections is provided by the Sonic protocol as described in Section 3.7.

3.6.4 Content Model

To allow users to keep full control over their data and control who accesses what part of their OSN profiles according to Requirement R5, Sonic proposes a decentralized data storage. Content management in Sonic stores content created and owned by a user in his own OSN profile, where every user can freely choose the OSN platform the OSN profile and all comprised content is stored on. Content of OSN profiles is stored in open formats defined by Sonic and addressable via UOIDs, rendering OSN profiles independent from the OSN platform they are hosted on. This allows OSN profiles to be migrated to other OSN platforms without links to and from the migrated content being interrupted. In order to allow content in the Sonic OSNF to be managed and delivered in a common fashion, all content is encapsulated in content objects of distinct types,

[19] While all tweets in Twitter are public by default, user profiles can be configured to be 'private' so that creating follow relationships and accessing a user's tweets requires approval by a profile owner.

where data for each OSN feature of the Sonic core featureset is represented by the matching data format. Content is addressable via Sonic URLs, which are domain-independent. As all types of the Sonic core featureset are represented in a common, human-readable data format, any OSN platform is able to use and process the enclosed information.

Sonic URLs

To uniquely address content stored in OSN profiles, Sonic URLs based on the respective user's GlobalID are used. Sonic URLs used in the Sonic OSNF replace the authority-part of a regular URL [198] with the GlobalID of the respective user. Following this paradigm, OSN profiles and all associated content can be uniquely addressed regardless of the OSN platform the content is stored in and even remain valid after a profile is moved to a new OSN platform and domain. When resolving a Sonic URL, the GlobalID in the authority-part is resolved to the associated Social Record dataset using the GSLS. Following, the GlobalID in the Sonic URL's authority-part is replaced with the value of the parameter `profileLocation` provided by the Social Record dataset as depicted in Figure 3.14. The value can either represent a domain name, including a specific sub-domain and port, or an IP address. The resulting URL is then used to locate and address the profile or content at its current location. This way, URLs can remain unchanged even if a profile is moved to a new OSN platform with a different domain.

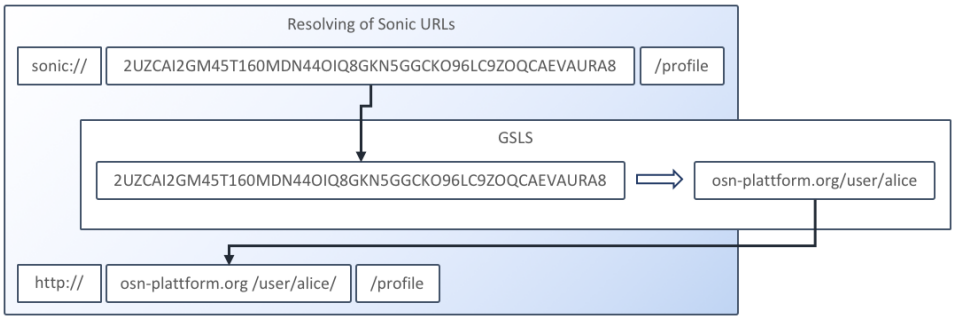


Figure 3.14: Resolving Sonic URLs in Sonic

Unique Object Identifiers (UOID)

To uniquely identify content across multiple federated servers while at the same time allowing OSN profiles and associated content to be migrated between OSN platforms, Unique Object Identifiers (UOIDs) are introduced. UOIDs are constructed of a global-part and a local-part, where the global-part is a GlobalID as defined in Section 3.5 and the local-part being an alphanumeric identifier that does not need to be unique on a global scale, yet is required to uniquely identify the content object in the context of the respective OSN profile. When creating a new UOID for a content object in the OSNF, the GlobalID of the user creating the content is used as the global-part of the UOID, while the local-part has to be unique for the OSN profile of the

respective user. An example of a UOID comprising a 12-char local-part is `2UZCAI2GM45T160MDN44OIQ8GKN5GGCKO96LC9ZOQCAEVAURA8:a32bf569f2ad`, where the structure of a UOID is described in Listing 3.2.

```
UALPHA       = %41-5A                ; (A-Z)
LALPHA       = %x61-7A               ; (a-z)
DIGIT        = %x30-39               ; (0-9)
GLOBAL-PART  = 50*52(UALPHA / DIGIT)
SEP          = ":"
LOCAL-PART   = 1*16(ALPHA / DIGIT)

UOID    = GLOBAL-PART SEP LOCAL-PART
```

Listing 3.2: Format of UOIDs in ABNF [235]

Content Model

The Sonic architecture implements a content model that stores content objects within the OSN profiles of their owners, from where they can be retrieved on demand. As Sonic explicitly abstains from publish-subscribe-models that would automatically replicate content to all subscribers, unnecessary replication of content objects is prevented. This allows owners of content to exert full control over the distribution of content.

In OSN services, several use cases exist, in which content is created to be used and displayed within the social profile of another user. Examples for this are status updates posted in another user's stream or a comment one creates on a friend's picture. Storing this type of content objects in the respective author's OSN profile would introduce a massive communication overhead, as displaying any content would require an OSN platform to query all remote locations and retrieve the respective associated content, such as comments, status updates, or likes. The design of Sonic's content model addresses this problem by storing content that was created in reference to other content directly at the location of the referenced content, thus satisfying Requirement R5. This way, communication overhead is prevented. Here, Sonic specifies two distinct types of content types, being local and remote content.

```
{
  "@context":      "http://sonic-project.net/", "@type": "comment",
  "objectID":      "4YIS70QYSITGCVQYYBSKY3P8I6W3J4QCNNF5V4KOH8ZHSPQ9C9:4857d0f26230c8c9",
  "targetID":      "4YIS70QYSITGCVQYYBSKY3P8I6W3J4QCNNF5V4KOH8ZHSPQ9C9:79a1f01e74c208eb",
  "comment":       "Test comment",
  "author":        "4YIS70QYSITGCVQYYBSKY3P8I6W3J4QCNNF5V4KOH8ZHSPQ9C9",
  "datePublished": "2017-12-05T14:21:22+01:00",
  "signature": {
    "@context":   "http://sonic-project.net/", "@type": "signature",
    "targetID":   "4YIS70QYSITGCVQYYBSKY3P8I6W3J4QCNNF5V4KOH8ZHSPQ9C9:4857d0f26230c8c9",
    "creatorGID": "4YIS70QYSITGCVQYYBSKY3P8I6W3J4QCNNF5V4KOH8ZHSPQ9C9",
    "timeSigned": "2017-12-05T14:21:22+01:00",
    "random":     "b6bf9f23e368ef89",
    "signature":  "-----BEGIN SIGNATURE-----M9eHleAKh9znXy8sX4hz09FlhJfaMgfyPQx6vVEN5pgIA
    A1jOSGmWarFIY7UcchsZd3ZWvKh3L4k5bTBX1BLOqCXn25GGL+QXLEKdJ75mhcYiNDTk+nZ6yp3sBES7v18CD
    /ds7g6575wDxZbyAm1Ltc5OoYPjIItghsz1u4jHcTF5RnP30Kp20hrkyQkxpIzC1ROqWhsoAwtt9s93t1z47f
    nNEDtkBzXct9R8g0bY0YYorP4uaWZ+HWXOdRQRJ+llpmChmXCI4cef4AmI7MjAX53ipr1tEwQ+akAaUJAIU4a
    zAdeCeAtNGKaekRRfM1ASLJsiKHtlLXdhYQHN7pKFw==-----END SIGNATURE-----" }
}
```

Listing 3.3: Example comment content object

- **Local Content** Content objects in Sonic are stored within the OSN profile of its creator and are referred to as local content. The content's author therefore has full control over the content, including access control, editing, or deleting data.
- **Remote Content** Content, which is associated with content stored within the OSN profile of another user, is stored within the OSN profile of the targeted content. As this type of content is possibly stored in a remote OSN profile, it is referred to as remote content. The owner of the OSN profile the content is being stored in gains full control over storage and distribution of all remote content objects, including deletion. As the content's creator loses direct control over the content's distribution and dissemination, remote content objects are secured against unauthorized change and edits via a digital signature created by the content author's AccountKeyPair. This allows anyone accessing the content to verify its integrity, so that unauthorized changes can be detected.

To uniquely identify a content object in Sonic, every content object specifies a parameter `objectID` specifying the object's UOID, as well as the GlobalID of the content's author. Furthermore, content objects can reference other content, for example a comment on an image or status update. In this case, the referenced content object is specified via a parameter `targetID`, allowing OSN platforms to identify all content that is associated with a given content object. Finally, remote content needs to be signed by its author and hence specifies a digital signature. The basic parameters of content objects in Sonic can be summarized as follows:

- **Object ID:** Every content object in Sonic can be identified by its globally unique UOID, which is composed of the author's GlobalID and a locally unique identifier. This allows content objects to be uniquely identified across different OSN profiles and platforms.
- **Target ID:** If content is related to other content objects, the relation is described by specifying the UOID of the related object as `targetID`. For example, when commenting on an image, the object describing the comment will specify the image object by specifying the image object's UOID as its `targetID`.
- **Author:** The author of a content object is the user who created it and is the only user who is allowed to edit its contents. In case of remote content objects, a digital signature ensures that the object's contents are not altered by an unauthorized user.
- **Signature:** As remote content can be stored in remote OSN profiles depending on the targeted content, remote content objects comprise a digital signature.

Content Ownership

As content in Sonic is stored by default within its author's OSN profile, the author also assumes the role of the content's owner, giving him full control over the distribution of his own content. If content is accessed by another user, Sonic makes a distinction between local and remote content objects. While local content is always stored at the author's OSN platform within his own social profile, remote content references another content object and is stored within the same social profile as the referenced content. This can result in situations where content is stored in a different social profile and platform. For example, a comment from a user *Alice* on a picture posted by another user *Bob* is stored as an attachment to the picture of *Bob* within *Bob's* account. In this scenario, even though *Alice* is the Author of the comment object, *Bob* assumes ownership of it and is therefore responsible for storage and delivery of the comment object of *Alice*. By storing remote content alongside the content it references, the overhead required to fetch all associated content objects is reduced, as all content is stored in one location instead of being distributed across multiple social profiles and OSN platforms.

Roles

All data in Sonic is encapsulated in content objects along with additional meta information, comprising information about its owner, creator, or type. When handling content objects, actors will assume one of the roles *Author*, *Owner*, *Viewer*, or *Provider*. This section explains the concept of The concept of encapsulating content in content objects is described in Section 3.6.4. Figure 3.15 depicts the roles employed in the Sonic OSNF.

- **Author** When creating content objects in Sonic, the creating user automatically assumes the role of the content object's author. Being the author of a content object authorizes a user to edit the content in any way. In order to indicate who created a content object, information about authorship is stored in every content object by specifying the author's GlobalID. To prevent content from being altered by someone other than the author, a digital signature is used to guarantee the content's integrity for some types of content.
- **Owner** Content objects are usually stored within the social profile of the user who created it. The user in who's OSN profile the content object is actually stored, gains ownership of the content, where the owner of content has full control over storage and access control of the content object. As certain content objects are stored within the social profile of a user who is not the content's author, some content objects are digitally signed by the author to prevent the owner from altering the content object's contents.
- **Viewer** A viewer is any user accessing (consuming) a content object of any user. Depending on the access permissions set by the content's owner, different viewers will be provided with different content. For example, *Alice*, a close friend of *Bob*, might access personal profile information such as a

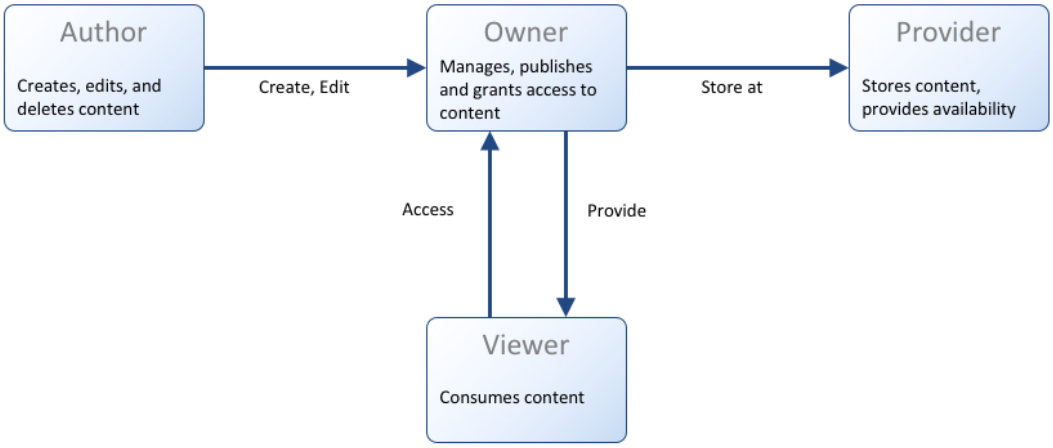


Figure 3.15: Sonic role model

private phone number, which is not accessible to other users. A viewer cannot edit content objects he accesses but is able to react, for example by commenting, to the content he accesses, depending on the type of content and the access permissions specified by its owner.

- **Provider** A provider runs a OSN platform, on which one or more social profiles are hosted. As the OSN platform provides the interfaces for accessing the content objects of the social profiles located on the platform, a provider is responsible for facilitating access to the data and handling protocol requests and responses. A provider is usually not author or owner of the content objects he handles. Providers must follow the directives specified by the content's owner and must not deviate from them. Providers must not change content objects.

3.6.5 Access Control Model

As described in Section 3.6.4, content objects in the OSNF are always stored within the OSN profile of the respective content's owner. This guarantees that users remain in full control of their data, as it is stored within their own OSN profile. In comparison to other DOSN architectures, for example solutions that are based on the Salmon protocol, content is not replicated to all subscribers. This allows users to exert more direct control over who is granted access to individual data records. Still, remote content objects are stored within the OSN profile it references and is hence possibly stored within the OSN profile of another user on another OSN platform. In this case, the remote OSN profile's owner assumes ownership of the remote content object, allowing him to specify access control policies for the content object.

As described in Chapter 1, the idea of the Sonic OSNF is to create a heterogeneous and open federation of OSN services, in which novel and existing OSN services are connected in a loosely coupled fashion. Existing OSN services can join the OSNF by implementing the architectural components specified by Sonic in order to be able to interoperate within the OSNF. This includes existing OSN service implementation with an already implemented access control model and policy definition mechanisms that allow users to specify and manage their privacy settings. Sonic aims to introduce as little overhead for integration into existing OSN implementations as possible and hence does not mandate how access control is implemented or exerted by an OSN platform. Nevertheless, Sonic specifies a format for describing access control rules, which can be directly applied to content and functionality specified by the Sonic protocol. The access control model of Sonic is hence defined as an optional model and can be substituted with other mechanisms for access control.

A survey of existing access control mechanisms implemented in existing DOSN services has been conducted by Pathak et al. in [249]. The survey found that most DOSN services implement a form of role-based access control scheme, allowing users to specify access control rules with varying granularity, while some of the surveyed DOSN services only support very basic access control mechanisms. The authors conclude that *"users should be able to express rich policies in terms of their social context, links, groups, and domain, which should be enforced before granting access to their data"* [249].

As specified by the NIST, role-based access control (RBAC) *"provides a valuable level of abstraction to promote security administration [...]"* and hence provides a flexible and extensible framework for access control in DOSN services [250]. The basic model of RBAC specifies permissions to access certain resources based on a set of defined roles. Users are then explicitly assigned to these roles and automatically acquire the permissions that were assigned to all members of this role, where both permission-role as well as user-role assignments can be many-to-many. To allow users to specify permissions for all users including users not explicitly assigned to a role, permissions can be specified for a role "*", to which all users belong. This allows users to specify base directives that apply in case no more specific permissions have been defined.

For access control in Sonic it has to be noted that the Sonic protocol in itself already specifies how content of a specific resource can be accessed. For example as described in Chapter 3.7, content of the resource PROFILE can only be read, but never created, updated, or deleted by anyone else other than the owner himself. Other resources, for example COMMENT or ACTIVITY, furthermore allow remote users to create, update, or delete content. While content may only be updated or deleted by the respective content's author, the content's owner may deny any legit requests to any form of content stored in his own OSN profile.

For regulating access to content of one's OSN profile, the Sonic access control model implements a two-tier access control mechanism based on the flat RBAC model [250] that allows users to specify fine granular policies. Following the Create-Read-Update-Delete (CRUD)-pattern popular in database management systems [251] or REST-ful web services [252], Sonic allows to specify access based on the HTTP method of an incoming request, being HTTP GET, HTTP POST, HTTP PUT, and HTTP DELETE.

The first tier of the Sonic access control model specifies permissions for accessing resources in form of access control rules, where each rule specifies whether to allow or deny access for every access method for a specific role. Access control rules are formatted as AccessControlRule objects as described in Table 3.6. The enforcement of the specified policies is implemented as follows: First, the mechanism determines all roles a user is assigned to. Second, all access control rules specified for any of the roles the user is assigned to are considered. If any of the specified rules allows access for the access method of the request, access is granted. Otherwise, access is denied, which is communicated to the requesting user via a HTTP 403 "permission denied" status code [253]. This allows scenarios in which a user *Alice* is both assigned to the roles *friend* and *acquaintance* for accessing another user *Bob's* OSN profile. Here, write access to the *Bob's* ACTIVITY resource can be denied for acquaintances, while it is allowed for friends. *Alice* hence is granted writing access to the resource, as she is assigned the role friend.

While tier one access control facilitates the definition of fine-granular access control rules on resource level, tier two access control allows to further specify content-specific access restrictions. Here, a user can specify access restriction rules for individual content objects, which then overwrite a decision of the tier one access control mechanism. This enables a user *Bob* to specify that his friend *Alice* with the assigned roles *friend*, *acquaintance*, and * may not access a specific status update object, even though she is generally allowed to access the ACTIVITY resource. Access restriction rules are formatted as AccessControlRestrictionRule objects as described in Table 3.7.

Table 3.6: AccessControlRule format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`AccessControlRule`" specifying the object type.
objectID	UOID	Required. The UOID identifying the AccessControlRule object.
owner	GlobalID	Required. The GlobalID of the access control rule's owner.
permission	String	Required. Specifies what kind of action the requesting entity is asking to perform following the Create-Read-Update-Delete (CRUD)-pattern. `C` maps to HTTP POST, `R` to HTTP GET, `U` to HTTP UPDATE, and `D` to HTTP DELETE.
role	UOID	Required. The UOID of the role to which the access control rule applies.
resource	String	Required. The name of the resource targeted. For example `PROFILE`.

Table 3.7: AccessControlRestrictionRule format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`AccessControlRestrictionRule`" specifying the object type.
objectID	UOID	Required. The UOID identifying the AccessControlRule object.
owner	GlobalID	Required. The GlobalID of the access control rule's owner.
restriction	String	Required. Specifies what kind of action the requesting entity is asking to perform following the Create-Read-Update-Delete (CRUD)-pattern. `C` maps to HTTP POST, `R` to HTTP GET, `U` to HTTP UPDATE, and `D` to HTTP DELETE.
role	UOID	Required. The UOID of the role to which the AccessControlRule applies.
targetID	UOID	Required. The UOID of the resource targeted.

Table 3.8: AccessControlRole format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`AccessControlRole`" specifying the object type.
objectID	UOID	Required. The UOID identifying the AccessControlRole object.
owner	GlobalID	Required. The GlobalID of the object owner.
name	String	Required. Unique name of the role.
members	Array<GlobalID>	Required. Array of GlobalIDs assigned to this role.

Table 3.9 specifies an example set of rules. The example assumes that two separate roles R_1 and R_2 are assigned to a user *Alice*, who wants to update a specific comment object she created earlier identified by $UOID_1$. The first rule specifies that no one is allowed to access the resource comment. The second rule specifies that users assigned to the role R_1 may access the resource via HTTP GET, but may not create, update, or delete content. Finally, the third rule specifies that users assigned to role R_2 may have full access to the resource. The tier one access control evaluation would hence grant *Alice* access to update the targeted comment object under the assumption that she is the legitimate author of the content object.

For tier two, a separate restriction rule is specified that disallows access for users assigned to role R_2 for the object identified by $UOID_1$. Alice is hence denied access to this specific comment object, even though full access is granted for all other content of the resource comment. The example shows that with the proposed access control model, users can specify fine-grained access control policies, where access can be controlled on a fine-grained level in tier one, and on a object-specific level in tier two.

Table 3.9: Access control example

Tier 1 Rules

role	**resource**	**permission**
*	COMMENT	
R_1	COMMENT	R
R_2	COMMENT	CRUD

Tier 2 Restriction Rules

role	**target**	**restriction**
R_2	$UOID_1$	UD

3.7 The Sonic Protocol

In order to facilitate seamless communication and interoperability in a heterogeneous federation of OSN service implementations, a common communication standard is required. Agreeing on a common way of communicating with each other allows OSN platforms to connect in an loosely coupled, federated fashion, where users are free to choose an OSN platform according to their personal preferences without being cut off from the rest of the social web.

A definition of a common protocol must be easy to introduce into existing OSN platform implementations as of Requirement R1 without the requirement for existing implementations to change their business logic, user interface, or overall service too much. Furthermore, the protocol must implement a decentralized, federated network of communicating servers (Requirement R3) that does not rely on any centralized component for control or management (Requirement R4). Most importantly, the solution must provide seamless communication and interoperability, rendering platform borders entirely transparent to users (Requirement R6). Finally, the protocol must support data portability of entire OSN profiles (Requirement R8) and provide extensibility (Requirement R11). Based on the requirements for an OSNF as defined in Section 3.3, a holistic protocol describing a set of APIs and data formats, the Sonic protocol, is introduced.

Existing platforms that implement the Sonic protocol must agree on providing the APIs defined by Sonic and receive content in the described data formats. Furthermore, OSN platforms are required to provide access to content according to the protocol, such as a user's profile page or stream, where data again is provided using the described data formats. Sonic does not stipulate how data is stored, handled, or displayed to the user, as such a requirement would demand existing OSN implementations to change their internal logic and user interface to great extends and lessen the acceptance of the approach.

This section is organized as follows: First, existing communication protocols and data formats are described Section 3.7.1, followed by an overview of proposed OSN communication standards and implemented APIs. Section 3.7.2 then describes the details of the Sonic protocol, including an extensive description of the APIs and data formats.

3.7.1 Related work

To allow servers in a loosely coupled federation to communicate with each other, all communication partners need to agree on a commonly used protocol and provide APIs as endpoints for message exchange [254]. Daigneau [255] describes three different styles of web service APIs, being RPC-based, message-based, and resource-based. Independently of the style of client-server communication, Daigneau points out that certain design principles should be followed, such as encapsulation or autonomy. Encapsulation should be used to hide complexity and implementation details from clients interfacing with an API in order to

prevent an implementation from becoming too tightly coupled with the API, while autonomy guarantees independence from other services.

For clients to remotely execute functionality of a web service, Remote Procedure Call (RPC) based APIs are used. RPC-based APIs usually expose a single or few endpoints, to which all RPC requests are sent. Request messages in RPC-based APIs uniquely identify the targeted function of the service to be executed [256]. To pass required or optional parameters to the service, requests specify a set of parameters that map directly to the parameters of the targeted function. Upon reception of the RPC request, the service maps the request to the targeted procedure, executes it, and returns a response message containing the procedure's return values. As clients address and access a service's functionality and data directly using RPC-based APIs, architectures built in this fashion are tightly coupled. Hence, changes in the design and the interface of the service require the client to be updated accordingly. Message-based APIs allow clients to send request messages containing information that describe a task to be executed. In comparison to RPCs, a service handler determines how to process and answer the request, so that functionality and data representation are kept hidden from the client. Message-based APIs are often used in SOAP [257], while in most cases messages are encoded using XML [258]. This way, message-based APIs are more decoupled from the service implementation and less tightly coupled. Resource-based APIs allow clients to access and manipulate a service's resources while being insulated from the underlying implementation and data representation of the service. Resources in the sense of resource-based APIs can describe anything from files to individual records in a database table. The pattern became popular with the Representational State Transfer (REST) [252], which promotes an HTTP-based framework for accessing web-based resources.

REST relies heavily on the existing functionality of HTTP [259] as well as other fundamental technologies of the web. Clients employ HTTP methods to perform Create-Read-Update-Delete (CRUD) operations for resources, which are uniquely identified by a URI. As stated by Tilkov et al., REST is based on five basic principles, being unambiguous identification of resources via URIs, Links and Hypermedia, standard HTTP methods, flexible representation of resources, and stateless communication, where the main benefits of RESTful APIs are loose coupling, interoperability, reusability, and scalability [260].

Best practices for designing Web APIs that are easy to learn, use, and maintain were described by Mulloy in [261]. The guide points out flexibility and freedom of REST as an architectural style for web APIs, promoting a *pragmatic REST* approach. In his guide, Mulloy emphasizes that use of naming and a clear structure of resources is important, while use of concrete names should be favored over use of verbs. Mulloy also points out the importance of error handling, where the occurrence of errors should be considered for every resource. Following the idea of utilizing HTTP status codes [253] to report errors, Mulloy suggests to use only a small subset of defined HTTP status codes, namely 200, 201, 304, 401, 403, 404, 400, and 500, and optionally provide more detailed information in a separate status code and detailed human-readable status message. Furthermore, Mulloy proposes that web APIs should be versioned to

allow clients to target a specific version of an API, where the addressed version of a request or response should be transferred as a dedicated HTTP header.

Recommendations for designing Web APIs were also proposed by Espinha et al. in [262] including providing a stable API that does not change too often to allow developers to migrate to new versions, while providing backwards compatibility with older API versions, as unexpected change of proprietary web APIs is acknowledged as a major problem in web service design. At the same time, the authors argue that older API versions should be removed after a reasonable amount of time, while example code for interaction of the API should be provided for the latest version to help developers implement against an API and to prevent under-specification.

Data Formats

With the idea to make social profile information an asset of the social web, a number of standards and data formats have been created that allow services and applications to provide and store social information in an open and standardized way. While early OSN services employed mostly proprietary data formats, more and more OSN services are adopting and building upon open standards. To allow information to be interpreted and used independently of service architecture, programming language, or operating system, several standards for representation of data and objects exist.

The Hypertext Markup Language (HTML) was introduced by Tim Berners-Lee in the 1990s to describe the structure of web pages with the use of HTML elements or tags [263]. HTML constructs, such as images, interactive forms, or sections allowed web designers to specify how a web page was to be rendered by the end user's browser software. The HTML standard was continuously evolved and extended by the W3C. To overcome the lack of mandatory well-formed documents, XHTML was introduced as an XML-based version of HTML [264]. While XHTML offered several benefits over the original SGML-based HTML standard, acceptance was comparably low resulting in an abandonment of the XHTML standard as work on newer versions of the original HTML standard proceeded. The latest version of the HTML standard, HTML 5.2, was released in 2017 [263].

As HTML does not support self-defined elements, the Extensible Markup Language (XML) was introduced as a standard to create structured documents for arbitrary purposes [258]. XML introduces XML-Schema for specifying a schema that allows to validate the structure and contents of XML documents; Extensible Style Sheets (XSL) [265] and XSL Transformation (XSLT) [266] for transformation of XML documents to human-readable and displayable documents, for example HTML; and XPath [267] for automatically traversing the document structure and addressing specific nodes in the document tree. XML has become a widely adopted standard for structuring data in web based services.

The Atom Syndication Format (ASF, commonly referred to as Atom) is an XML-based language designed for structuring and publishing web feeds [163]. An Atom-document defines a `<feed>` with a list of `<entry>` items that encapsulate published content. Atom feeds can be linked in HTML web pages using a `<link>`

element that links the Atom feed with the page. Feed reader software can then access and download this feed and display the contents in a client application.

The JavaScript Object Notation (JSON) is a widely adopted text-based data format for platform and language independent representation of data and objects, based on a subset of the JavaScript language. JSON was specifically designed to be lightweight and easy to read and compose by humans, while being easy to generate and parse for machines at the same time. The JSON standard describes two data structures, being JSON objects and JSON arrays [268]. While JSON objects store key-value pairs, JSON arrays are simple ordered lists of values. JSON supports a variety of data types, such as strings, numbers, JSON objects and arrays, boolean, and null, and is natively supported by most of today's popular programming languages [269]. The JSON format is described in two competing standards, of which ECMA 404 [268] describes the JSON syntax and IETF RFC #7159 describes interoperability issues [269].

To utilize JSON in various scenarios, several extensions have been specified. To allow a specification of structure of JSON objects, JSON Schema has been introduced. JSON Schema provides structural validation of JSON objects and can be used for automated testing and validation purposes [270]. To support Linked Data (LD) [271] principles for semantic web support, JSON objects can be augmented with semantic annotations [272]. The JSON-LD standard augments JSON objects with context in form of machine-interpretable semantic annotations starting with an `"@"`-prefix, while the original data structure remains intact. This way, JSON objects can be linked to concepts in an ontology.

To securely transport JSON-encoded claims or data over an untrusted communication channel, JSON Web Token (JWT) has been introduced [273]. A JWT is a *"[...] compact, URL-safe means of representing claims to be transferred between two parties"* [236], where the payload of a JWT is either signed as a JSON Web Signature (JWS) [274] or encrypted as a JSON Web Encryption (JWE) [275]. A JWT can be used to transport one's claim of identity between communication parties, assert one's identity, or generally transport data in a secure fashion over an untrusted communication channel. Depending on whether a JWT is a JWS or a JWE, the payload is encrypted or signed. JWTs comprise three parts, being a JSON Object Signing and Encryption (JOSE) header, the payload, and the signature [236]. Both header and payload are formatted as JSON objects, where the JOSE header specifies the JWT's type and used algorithm. For serialization, both header and the - in case of JWE encrypted - payload are separately encoded using base64url [237], where the signature is created for the encoded payload. Encoded header, payload, and base64url-encoded signature are concatenated with a `"."` as separator, thus creating a string representation of the original object that can be easily distributed between systems and services.

Data Formats for Social Information

To model persons or entities as well as the relationships between them, several standards have been proposed. vCard [276] was introduced by the Versit Consortium in the early 1990s as a standard to digitally exchange contact details. vCard stores data in textual files, formatted as key-value pairs with a specific format defined by the standard, where a vCard document typically specifies information such as a person's name, phone number, email, or postal address. While vCard has been widely adopted by email and contact management applications, several proposals have been made to utilize modern and common data formats, such as JSON-based jCard [277], XML-based xCard [278], or XHTML-based hCard [153].

Portable Contacts (PoCo) is an open protocol and data format for describing and accessing contact information [279]. The standard was introduced to facilitate and support data portability of existing contact management applications and services. Portable Contacts is based on JSON and describes a request/response mechanism to query contact details from service providers. Until its abandonment in 2016, Portable Contacts was integrated in OpenSocial and OStatus.

The Friend-of-a-Friend (FOAF) ontology allows to describe connections between individuals in a decentralized fashion [104]. The standard defines a vocabulary using the Resource Description Framework (RDF) [203] and the Ontology Web Language (OWL) [280] to model relationships and store their semantics in a machine-readable format. Using semantic web technologies, the standard allows to store a user's contact details as well as information about connections and relationships to other users. This way, users can model and publish a local view of the social graph, which can then be accessed and interpreted by semantic interpreters. FOAF is used in the WebID specifications [202] and is one of the key components of WebID+TLS [205].

Similar to FOAF, the XHTML Friends Network (XFN) aims to describe and publish social relationship information in a semantic way [281]. XFN is based on XHTML and utilizes `<rel>` attributes to specify how a person or entity represented by a website is related to other entities, where all entities are identified by URLs. This way, XFN allows to specify if and how individuals are related, for example by specifying that someone is a `colleague` and `friend`, or that both have actually `met` in person.

With the idea of a *"pragmatic path towards the vision set forth for the semantic web"* [153], Khare and Çelik defined a set of XHTML-based formats to describe addresses, events, calendars, or contact information, published under the name *Microformats*. The standard targeted information published in blogs that should become machine-readable. The philosophy behind the collection of formats aims at reducing complexity in order to yield simple solutions, reuse existing and widely-adopted standards, and recycle through modularity. In 2011, work on the successor *Microformats2* started, which is based on JSON. As of 2017, the standard comprises formats for addresses, contact information, blog posts, events, news feeds, locations, generic lists, products, cooking recipes, resumes, and reviews[20].

[20] Microformats2: `http://microformats.org/wiki/microformats2`. Accessed: 2.8.2017

Other approaches focus on representing content created by users in OSN services. The Open Graph Protocol (OGP) was proposed by Facebook in 2010 with the intent to enable *"[...] any web page to become a rich object in a social graph"* [282]. The standard was inspired by the Dublin Core metadata element set published by the Dublin Core Metadata Initiative (DCMI)[21], which is a standardized vocabulary for creating metadata [283]. The OGP and makes use of RDFa [284] to enrich a web page with semantic information that describes the web page's contents. Meta information is added in form of HTML `<meta>` tags by specifying a title, a type, an image, the web page's URL, and further optional parameters [285]. Following this standard, social web services can read and use this information to describe the web page or its contents as an entity in a social graph.

Activity Streams is a data format used to provide semantic descriptions of actions, referred to as activities. Activities usually comprise an *actor*, a *verb*, an *object*, and a *target*, with the semantic meaning that someone (actor) carried out an action (verb) with or affecting an entity (object). Activity Streams hence provides *"a model for representing potential and competed activities"* [102]. The JSON-based format has been initially introduced in 2011 [114] and is widely adopted by other platforms and standards such as OpenSocial, Windows Live, or MySpace[22]. Activity Streams 2.0 has been published in 2014 and employs JSON-LD [272] to enrich activity streams objects with semantic information [102].

In direct comparison to Activity Streams 1.0 [114], which defines an activity by specifying who (*actor*) did what (*verb*) with what (*object*) to whom or what (*target*), the Activity Streams 2.0 vocabulary [196] distinguishes 9 different types of activity objects, being *Object*, *Link*, *Activity*, *IntransitiveActivity*, *Actor*, *Collection*, *OrderedCollection*, *CollectionPage*, and *OrderedCollectionPage*, where different types can be specified for objects, activities, and actors. An example is an activity typed as "*view*", indicating that the activity's actor has viewed the associated object, allowing to describe activities in a far more detailed fashion. While the data formats significantly changed in comparison to Activity Streams 1.0, the semantics mostly stayed the same.

OpenSocial

OpenSocial, the first holistic approach to access data from different OSN services using a single API has been proposed by Google in 2007 [115]. The standard, which has been abandoned in 2014, defines a set of APIs for building social applications on the web, and includes not only a protocol and API, but also an extensive set of data formats and furthermore covers user identification. The motivation for the standard is that developers of applications that should access information of a user's social profile had to implement a library for the - mostly proprietary - APIs of all OSN services they wanted their application to work with. OpenSocial provides a common REST API that allows an application that implements it to work with any OpenSocial compliant OSN service,

[21] Dublin Core Metadata Initiative: `http://dublincore.org/`. Accessed: 2.8.2017

[22] Implementors of Activity Streams: `http://wiki.activitystrea.ms/w/page/24500522/Implementors`. Accessed: 2.8.2017

called *container*. OpenSocial applications are then able to access information of social profiles of any compatible OSN service using the common set of APIs. As a reference implementation for OpenSocial, the Apache Foundation developed Shindig[23], which functioned as a blueprint for developing OpenSocial applications until it was retired in late 2015. While the first versions of the OpenSocial specification followed a one-size-fits-all approach, Version 1.0, which was released in March 2010, introduced a modularized architecture and added extensibility to containers. Version 2.0 added support for Activity Streams 1.0 [114] in 2011 and removed outdated data formats in favor of newer ones. After a decline of adoptions of the standard, the W3C announced that further work in the area should be moved to the *W3C Social Web Working Group* and *W3C Social Interest Group* by the end of 2014, effectively announcing the abandonment of the standard[24]. The latest stable version of the OpenSocial API, version 2.5.1, allows CRUD operations for all endpoints, being people, groups, activities, activitystreams, appdata, albums, mediaItems, and messages:

- **people:** The resource people allows to retrieve user profiles or list of user profiles, which are returned as Person objects or lists of those. Using query parameters, this resource is also used to retrieve user profiles of certain groups by specifying a parameter `group-id`, or a person's friends list using the parameter `@friends`. While support of reading access for the resource people is required by the standard, containers may optionally implement support for creating, updating, and deleting links between user profiles. For this functionality, the standard explicitly states that this is a *"generalization of many use cases including invitation and contact creation"* and that dual opt-in can optionally be required by implementations [115]. The Person data format describes a person in the context of the OSN service. While the format only requires only two parameters `id` and `name` to be specified, the standard lists an extensive collection of optional parameters, including one's gender, location, or relationship status.

- **groups:** OpenSocial supports user groups, which may be used to *"tag or categorize people and their relationships"*. An OpenSocial group is owned by a person, who can configure it to be private, invitation-only, public, or personal. The resource group may optionally be supported by containers and allows functionality to read, create, update, and delete user groups. When requested, a Group data object is returned, being a simple container format with two mandatory parameters `id` and `title`, and an optional parameter `description`.

- **activitystreams:** OpenSocial added support for Activity Streams 1.0 [114] in version 2.0, which replaced the former resource activities. OpenSocial's implementation of Activity Streams organizes ActivityEntries in lists, where each ActivityEntry describes a single activity by specifying a verb, actor, object, target, and additional optional parameters. Support for the resource

[23] Apache Shindig: `http://shindig.apache.org/`

[24] OpenSocial foundation moves standards work to W3C Social Web Activity: `https://www.w3.org/blog/2014/12/opensocial-foundation-moves-standards-work-to-w3c-social-web-activity/`. Accessed: 3.8.2017

`activitystreams` is mandatory for containers and features CRUD operations for ActivityEntries, as well as means to retrieve the Activity Stream, for example a list of ActivityEntries, from a person.

- **albums and mediaItems:** Support for albums as well as for mediaItems is implemented as an optional feature. Albums would group mediaItems, where each mediaItem would specify the id of the album it is associated to. Albums specify a title and description, the type and number of contained mediaItems, location, URL of a thumbnail to be used as the album's cover, the owners id, and an object identifier, but don't comprise or link to the actual media items. The mediaItem format itself comprises an extensive list of parameters, including it's album id, comments, number of times the item was viewed, a rating, tags, and a title and description. The actual contents data is linked via a URL. Both the album and mediaItem resource feature CRUD operations to create, read, update, and delete individual albums or media items.
- **appdata:** To allow third party applications to store data when used with a OpenSocial-compliant OSN service, the resource appdata provides support for storing arbitrary information in form of key-value pairs. Information stored using the appdata store is only accessible by the currently active user (viewer), who can read, write, and delete the information.
- **messages:** OpenSocial features optional message support to read, send, and delete messages, where a message is a textual message that can be addressed to one or multiple recipients. To mark messages as new, read, or deleted, a message can be updated by updating its status parameter. Finally, messages can be grouped in message collections, for example by topic. Amongst other information, the message data format specifies a message title, body, list of associated URLs, a timestamp of when the message has been sent, a list of recipients, the sender, and the message's status. Furthermore, the format specifies a list of replies to this message following the ATOM threading model [286].

Table 3.10 provides an overview of existing data formats and their coverage of the Sonic core featureset as introduced in Section 3.4. Specification of social profile pages is covered to a varying extent by most standards. While vCard and its derivatives only support storing basic contact information but support extensions and focus on storing address information such as phone numbers, email addresses, or postal addresses, Microformats provides address formats in addition to it's hCard specification. Activity Streams 2.0 provides an object type *Profile* that allows to link to an *Actor Type* object, representing for example a group, organization, or person. This allows to represent basic information about a person in the context of an activity, but does not support specifying detailed social profile pages. While being extensible, Activity Streams has been designed for describing activities and not for encapsulating data. Portable Contacts specifies an extensive list of profile attributes and hence facilitates formatting and storage of entire social profile pages. The format is directly referenced by Open Social, which specifies the type *Person* that allows specification of an

Table 3.10: Overview of Sonic core featureset coverage per data format

Feature	**vCard** [276]	**PoCo** [279]	**FOAF** [104]	**XFN** [281]	**OGP** [282]	**AS 2.0** [102]	**Open Social** [115]	**Microformats** [153]
Social Profile	◐	●	○	○	○	◐	●	◐
Link	○	○	●	●	○	●	○	○
Conversation	○	○	○	○	○	○	●	○
Stream	○	○	○	○	○	●	●	●
Like	○	○	○	○	○	●	○	○
Comment	○	○	○	○	○	●	○	○
Tag	○	○	○	○	○	●	○	○
Image	○	○	○	○	○	◐	◐	○

extensive list of attributes for a user. Specifications of connections between individuals are covered by both FOAF and XFN. Both standards allow to specify a connection between users and furthermore give a description of the type of the relationship. Yet, both FOAF and XFN lack a description of how links are established. The Activity Streams 2.0 specification specifies a *Relationship* object type that can be used to represent a relationship between individuals. While the standard does not support a format for representing initializing a link between users, a combination of an *Offer* typed object with a *Relationship* is used as an example for requesting a link to be established. Description of messages are supported by Open Social, where assignment of messages to a conversation is not covered. Streams as a representation of posted activities are supported by both Activity Streams 2.0 and Open Social, where the latter standard explicitly references Activity Streams 1.0 as the format for activity description. Activity Streams further supports formatting activities that resemble likes, comments, and tags. Finally, support for images, being media items in general, is supported by both Open Social and Activity Streams 2.0. Both standards provide a way to store information about an image, while the actual image data has to be stored and managed separately.

The comparison shows that none of the surveyed standards is able to cover the entire OSN core featureset as introduced in Chapter 3.4. While Activity Streams 2.0 covers support for most OSN features, the standard is designed for representation of activities in a feed and is hence not fully suited for representation of several OSN features. Sonic data formats are therefore inspired by the aforementioned standards and aim at inheriting their strengths while adding support for use cases that the original standards lack.

Protocols for Social Information Exchange

While representation of information in OSN services is crucial, distribution of content in a standardized way poses an even more challenging task in decentralized, independent service federations. For this domain, a number of protocols have been proposed that address the issue of communication and distribution of content or messages in decentralized services.

The likely most famous protocol in this domain is the Extensible Messaging and Presence Protocol (XMPP). Originally named Jabber, the protocol was invented in 1998 as an open and decentralized alternative to the plethora of mutually incompatible IM services. The first version, *jabberd 1.0*, was published in 2000 and managed to attract a large community of developers, ultimately resulting in a standardization of the protocol under the name XMPP in 2004 [72]. XMPP implements a protocol and data formats that *"power real-time interactions over the Internet"* [72]. While the standard was predominantly perceived for it's use in IM, many extensions have been proposed and implemented that extend the scope of the standard far beyond simple exchange of messages.

XMPP builds on a distributed client-server infrastructure, where XMPP servers are connected in a loosely coupled federation [154]. As protocols and data formats for server-to-server (s2s) and client-to-server (c2s) communication are standardized, any client or server implementation that is compliant with the standard can be used to connect to the XMPP network. This allows users to register with any XMPP server and connect to the network via any XMPP client. When logged in on an XMPP server, messages from a user's XMPP client are sent to the connected server, which forwards them to the recipient's server, from where they can be accessed by the message's recipient. Communication in XMPP uses XML streams for data exchange for c2s and s2s communication where XML-based data primitives, called *XMPP stanzas*, can be either of type *message*, *iq* (info/query), or *presence*. While message stanzas are used to send typed messages that do not require a response or acknowledgment by the receiving entity, iq stanzas require the receiving entity to respond, thus implementing a request-response pattern similar to HTTP [72]. Finally, presence stanzas implement publishing of a user's current status, such as 'busy' or 'online' [287].

XMPP identifies users via Jabber Identifiers (JID), which comprise a local-part, being an identifier that is locally unique on a specific XMPP server, and a domain-part, which represents the FQDN of the XMPP server, where local and domain-part of a JID are separated by an "`@`" [201].

To support versatility and extensibility of the standard for use cases beyond IM, XMPP supports the XMPP Extension Protocol (XEP) as defined in [142]. XEP allows adding support for various functionality, such as Multi User Chat (MUC) [288], feature negotiation [289], or delivery of message receipts [290]. As of August 2017, 189 protocol extensions have been published.

Matrix [291] has been introduced in 2014 as an alternative to XMPP and other traditional communication protocols, being an open, federated set of APIs and protocols for IM, Voice over IP (VoIP), and the Internet of Things (IoT) for real-time communication. As XMPP was deemed to be *"not particularly web-friendly"* and its support for history synchronization and lacking support for

mobile use-cases [292], Matrix was started as an open source project to draft and implement a HTTP and JSON-based communication framework[25]. Similar to XMPP, Matrix consists of a federation of loosely coupled servers, to which users connect with Matrix compatible client applications. Messages called *events* in Matrix are routed to their destination from the originating client via its connected server to the recipients' servers, from where they can be accessed. All servers participating in a Matrix communication maintain a history of sent and received events in an *event graph* and account information for all clients.

WebSub, which initially has been published under the name PubSubHubbub (PuSH) [155], provides a mechanism for content distribution between publishers and subscribers [293] and has become a basis for many other content distribution standards. For a subscriber to create a valid subscription in WebSub, a publisher offers an endpoint called *hub* that validates and processes subscription requests. To establish a subscription, an HTTP request is sent to the publisher's hub comprising information about the subscribers endpoint, being described by a URL. After a subscription has been successfully registered, content posted by the publisher is automatically sent to the subscribers endpoint.

Further protocols targeting microblogging services have been discussed in Chapter 2.3, showing that the existing protocols facilitate a federation of a multitude of instances of the same microblogging service, but mostly fail to implement interoperability with other service implementations.

Protocols and APIs of OSN services

As of today, most OSN platforms provide an API and protocol to allow developers of third party applications and services to connect to the OSN, thus creating well integrated solutions delivering a streamlined QoE for the user [294]. Most existing APIs are based on REST and utilize open data formats and standards. Still, all existing APIs require users and third-party services to register with the service to access its API, where authentication is done in most cases using OAuth 2.0.

Facebook provides an extensive HTTP-based RESTful API for third party applications, named Graph API "*after the idea of a social graph*" [194], where information is represented as nodes, edges, and fields. Following this model, nodes represent items such as posts, comments, or users and are connected by edges, for example linking a comment to a posted photo. Every node is identified and addressable via a unique ID and can be accessed either directly or by following edges connecting individual nodes. Finally, fields contain information about nodes, such as a person's name or date of birth. As support for new features is added from time to time, the API is versioned. New features are added in newer versions, while older versions of the API are supported for a duration of two years after a newer version was released. In order to both provide a stable and continuously supported API for Facebook functionality and support new features at the same time, the Graph API is divided into core and extended APIs. While core functionality is guaranteed to be available and unmodified

[25] Matrix standards are still under development. As of August 2017, most Matrix APIs were marked as unstable [291].

in a specific version for a certain time after release, extended functionality may be modified or even removed and has hence to be accessed by explicitly specifying the API version to utilize. Functionality in version 2.9 of the Graph API [194] covers access to user profiles, pages, groups, and events, but also allows access to content associated or contained in these nodes, such as photo albums and videos, comments, conversations and messages, status updates, or friends lists. Furthermore, the API provides access to information valuable to third party developers and advertisers such as "insights", "offers", or "promotional information".

Google+ offers a REST-based API that allows third party developers to access a user's social profile, which is not versioned [295]. In contrast to most other OSN services, Google+ restricts API access to reading content, while publishing, editing, or deleting content is not supported. The API allows to address the resources people, activities, and comments, where individual objects can be addressed by their IDs. The resource people implements access to individual person resources, where every person represents a unique user profile on Google+. While individual profiles can be retrieved by specifying their ID, the people resource is also capable of listing all person objects associated with a certain object in Google+, for example users who commented on or shared a post. Additionally, users can search for other Google+ users. Activities can be retrieved individually or as a list of available objects. Furthermore, the resource allows to search for activities matching specified search criteria. Finally, comments can also be addressed individually by their ID or as a list, but cannot be searched.

VKontakte provides an RPC-based [256] API for third party applications[26]. Like with most other APIs, applications are authenticated via OAuth 2.0 [296], where the API in version 5.65 facilitates access to an extensive list of resources such as documents, users and friends, groups, feeds, pages, photos and videos, or messages.

RenRen allows access for third party applications via a HTTP-based RESTful API[27]. The API in version 2.0 authenticates applications accessing it using OAuth 2.0 [296] and allows access for most OSN features supported by the service, including profiles, friends of users, blogs and feeds, likes, comments, photos, or check-ins.

Linkedin provides a HTTP-based API[28] supporting XML and JSON [268] data formats. The API provides access to basic features of the network and has two base resources, people and companies. While for companies information such as the company's page, status updates, shares, and a list of followers can be accessed, access to the resource people simply allows access to a user's profile. While most of the service's API allows only reading access, creating comments and shares are supported.

Xing provides a REST-based API[29] for accessing and editing resources. The API allows requesting and editing one's user profile and read, write, and edit content in groups in form of posts and comments and even manage one's

[26] VKontakte API: `https://vk.com/dev/methods` Accessed: 23.6.2017
[27] RenRen API: `http://open.renren.com/wiki/English_version_for_API2`. Accessed: 23.6.2017
[28] Linkedin API: `https://developer.linkedin.com/docs`. Accessed: 3.8.2017
[29] Xing API: `https://dev.xing.com/docs/resources`. Accessed: 3.8.2017

contacts by accessing and managing contact lists. Furthermore, conversations and messages, feeds, news, events, and companies can be accessed and created.

Twitter supports a set of well-documented, distinct APIs[30] to access information of the service. The REST API provides access to content on a request-response basis, the Streaming API allows developers to request a stream of tweets, and a Webhook API provides real-time access to account data. The APIs are based around four main object types, tweets, users, entities, and places, which are formatted using JSON [268]. The REST API facilitates access of a user to his own account and associated data. Functionality covered by the REST API comprises searching, sending, and requesting tweets grouped by topic or users, accessing profiles and lists of followers, and send and receive direct messages. Twitter addresses objects and users within the service using Twitter Snowflake[31], a service to create unique identifiers [211], while authentication is based on OAuth 2.0 [296].

Instagram provides a RESTful API for third party applications. The API allows read access to the endpoints users, media, tags, and location, where for the resources tags and location a list of media objects is returned that fits to the specified tag or location. Furthermore, comments and likes can be read, created, and deleted for specified media items, and follow relationships between users can be created and modified.

Pinterest provides an HTTP-based API[32] for accessing the service. The API supports requesting information about users, including their public boards and pins, and allows users to create and remove follow relationships for boards and other users. Furthermore, the API supports CRUD access for one's boards and pins and allows search requests. Data objects defined as return types are users, boards, and pins.

While most APIs and Protocols are building on open standards to some extent, their design and architecture still remains proprietary and specifically targets the respective service infrastructure. Aside from the discontinued standard OpenSocial, accessing multiple OSN service platforms with one protocol is not supported.

Other Approaches

Hu et al. proposed SNSAPI, a cross-platform middleware for Meta Social Networks (MetaSN), which are defined as a federation of OSN services that are not relying on a common protocol or service architecture [100]. The SNSAPI middleware connects to a list of centralized OSN services using their proprietary APIs to facilitate communication between the different services. In order to allow content to be exchanged between services with incompatible data formats, Hu et al. derived a common data format for data objects, which can be used to reformat content objects from a source format into the format of the target OSN service. This way, communication between OSN services is made possible, yet introduces

[30] Twitter APIs: `https://dev.twitter.com/overview/api`. Accessed: 3.8.2017

[31] Twitter Snowflake: `https://dev.twitter.com/overview/api/twitter-ids-json-and-snowflake`. Accessed: 3.8.2017

[32] PinterestAPI: `https://developers.pinterest.com/docs/getting-started/introduction/`. Accessed: 3.8.2017

some drawbacks. The middleware categorizes OSN functionality into three distinct categories, being base functions, derived functions, and extra functions. SNSAPI exclusively covers base and derived functionality, being *authentication*, *home timeline*, *update*, *reply*, and *forward*, as the authors deemed extra functions, such as album and like support, not to be *"very common across platforms"*. Extra functions are hence not supported by the SNSAPI approach.

Similar to SNSAPI, the EU-funded project SocIoS[33] proposes a SOA middleware that relied on adapters connecting to proprietary APIs of OSN services. The project derived an ontology for OSN functionality comprising persons, media items, activities, events, messages, ratings, and groups [176]. The architecture aims at extracting content from popular OSN services and to aggregate them in the SocIoS service for further processing depending on the desired use case [297] and hence also relies directly on proprietary APIs.

Other approaches also proposed connecting OSN services based on wrappers for the various proprietary APIs. For example, Mostarda et al. proposed an OpenID-based middleware architecture that is based on adapters and converters for OSN APIs [298], while Gouriten and Senellart proposed API Blender, a middleware that aims at unifying proprietary OSN APIs by creating WSDL-based, machine-readable descriptions of API capabilities [299]. Furthermore, a number of software frameworks, such as Spring Social[34], has been proposed that aims at easing integration of OSN APIs in applications and services by providing a wrapper classes.

Even though social features such as commenting, expressing appreciation, messaging, or sharing content have become an integral part of the social web we know, integration of such features as of today is only possible by implementing against proprietary APIs. While an extensive selection of open standards and protocols have been published, they fail to provide a holistic and platform independent communication and interoperability framework. Most solutions cover a specific, isolated aspect of the social web, such as describing activities [114], subscribing to other user's content updates [152], or third-party applications accessing social content [115]. A holistic solution that facilitates seamless interoperability between arbitrary OSN services has yet to be defined.

3.7.2 The Sonic Protocol

The Sonic protocol marshals access to contents of a user's social profile and communication between Sonic compliant platforms using RESTful APIs. The protocol comprises two separate APIs, being the Profile API and the Platform API, while communication with the separate GSLS is covered by the GSLS API as described in Section 3.5.3. While the Profile API facilitates access to a user's social profile information in the scope of a user, the Platform API organizes access to a platform's general functionality, such as searching or negotiating supported platform features. Finally, the GSLS API allows for resolving of GlobalIDs and updating Social Records via the GSLS.

[33] SocIoS project web page: `http://www.sociosproject.eu/`. Accessed: 3.8.2017
[34] Spring Social: `http://projects.spring.io/spring-social/`. Accessed: 4.8.2017

The Sonic protocol exclusively covers communication between OSN platforms and profiles, while resolving GlobalIDs via the GSLS is handled by a separate API. Sonic does not specify any requirements for displaying or handling of information and content to allow both existing and new OSN implementations to have full control about how content is delivered and displayed and hence provide a unique user experience for users on their platform.

Protocol Context

All requests and responses in Sonic are sent in the context of an entity, where an entity is either a user or an OSN platform that creates the request or response. To verify the authenticity of requests and responses of the Sonic protocol, a list of HTTP headers is specified. These HTTP headers comprise information about origin and targeted entity of the request or response. Furthermore, a digital signature is created by the sending entity to make a request's and response's authenticity and integrity verifiable. To digitally sign the request or response, the sending entity's AccountKeyPair is used, while the sending entity's GlobalID is specified in the HTTP header `SonicSourceGID`. This allows the recipient to verify the enclosed digital signature by retrieving the matching public key from the GSLS.

The digital signature is created by concatenating the HTTP method used, the URL of the request, the HTTP headers of the request, followed by the content of the request's or response's body. The signature is then created using OpenSSL with RSA and SHA512 and encoded as of PKCS#1v2.1 [300].

Table 3.11 lists the HTTP headers specified by requests and responses according to the Sonic protocol. All headers are required and need to have valid values. Any request or response with missing headers or invalid values should be discarded by the receiving OSN platform.

Table 3.11: Sonic HTTP Headers

Header	Description
`SonicTargetAPI`	Specifies the version of the Sonic protocol used in a request or response to allow support for different versions of the protocol.
`SonicResourceDate`	Date and time of a request or response in XSDDateTime format.
`SonicPlatformGID`	GlobalID of the OSN platform sending the request or response.
`SonicSourceGID`	GlobalID of the entity (i.e. platform or user) in whose context the request or response is sent.
`SonicRandom`	Random value for added security [301][302].
`SonicFeatureID`	Hash of the supported feature extensions as of Section 3.7.3.
`SonicSignature`	Digital RSA signature of the request or response. The signature comprises the Sonic HTTP headers, the body, and for requests additionally the HTTP method and URL

Request-Response Pattern

Communication in the Sonic protocol follows the Request-Response pattern using HTTP. Upon reception of a request, a Sonic OSN platform must answer the request with an adequate response containing a response object. Response objects are JSON-formatted objects that comprise information about the success or failure of the matching request and can optionally contain serialized payload (see Table 3.12).

Table 3.12: Response object format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`response`" specifying the object type.
responseCode	int	Required. The HTML status code for the response.
message	String	Optional textual message.
errorCode	int	Optional numerical error code identifying the cause of an error.
body	String	Serialized payload of the response. Optional depending on the type of request.

If a response is used to deliver data, the information is transferred within the response object, where the content to be transferred is encoded as a content object as introduced in Section 3.6.4. Sonic defines adequate data formats for every OSN feature in the OSN core featureset to ensure that the receiving side of the response message is able to interpret and use the transmitted information. In the remainder of this section, data formats and API calls are described to retrieve and transmit data in the OSNF.

3.7.3 Platform API

For requests that are not executed in the context of a user, the Profile API cannot be used, as the Profile API only supports access to the content published in the OSN profile of a user. Hence, access to a OSN profile's contents are managed via the Profile API, where the request is handled in the context of the targeted user. Requests, which are not executed in the context of a user, are executed in the context of the respective OSN platform. Here, the active entity is the OSN platform itself. Scenarios, in which the Platform API is used instead of the Profile API are an OSN platform answering a request targeting a not existing user, answers to search requests, or negotiating feature support between OSN platforms.

FEATURE

Negotiation of supported features between servers and clients or servers and servers has been addressed in a list of other open protocols and services. The File Transfer Protocol (FTP) [303] allows clients to request a list of commands from a server that extend the default set of commands as specified in [304]. RFC 2389 specifies a `FEAT` command to be sent by a client to a server, which is to be answered with a list of supported commands. Clients then may use the listed commands when communicating with the server, where all optionally supported commands are described in RFC 2389 and are identified by a unique name [304]. The FTP protocol hence allows support for a limited set of optional commands, which are specified in a central document. A similar implementation for negotiation of supported features exists in the Datagram Congestion Control Protocol (DCCP), which allows clients to may request a list of supported features from a server, where available features are specified in RFC 4340 [305].

XMPP [154] supports extensibility of the protocol via the XMPP extensions protocol [142]. For discovery of support for feature extensions, the service discovery extension (disco) provides a mechanism for clients and servers to request a list of all supported protocol extensions. The requested entity responds to a request with a list of supported protocol extensions, where each listed feature is identified via the unique name it is registered with at the XMPP Standards Foundation (XSF) registrar. To reduce the amount of disco requests, the list of supported features is hashed and transmitted in presence stanzas. Clients and servers may store a received hash and hence are able to detect when the list of supported protocol extensions was changed.

To allow negotiation of supported features between different OSN platforms, the Sonic protocol supports feature negotiation similar to XMPP. This API allows OSN platforms to request a list of feature implementations implemented by other OSN platforms, which are not covered by the OSN core featureset of Sonic as defined in Chapter 3.4. The feature API allows any OSN platform to send a request for supported features to another OSN platform. A feature is described by a feature object as described in Table 3.14 specifying a name, a URI functioning as a unique identifier, a version number, a list of compatible feature versions of the same feature, and the API path to which requests for this feature can be sent. Using this information, it is possible for the OSN platform to define, which features can be used when communicating with another OSN platform. The list of supported features can be accessed via HTTP GET as described in Table 3.13.

To reduce the overhead of requesting an OSN platform's features more often than necessary, each platform exposes a hash value for the supported Sonic protocol version and all additionally supported features and versions in all request and response messages as HTTP headers as specified in Section 3.7.2. OSN platforms may store the hashes published by other platforms. A changed hash value indicates that the featureset of the respective OSN platform was changed. An OSN platform may then request the updated list of OSN features via the feature negotiation protocol.

Table 3.13: Sonic feature API

Method	Path	Parameter
GET	/FEATURE	
GET	/FEATURE/:featureID	

Table 3.14: FEATURE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`feature`" specifying the object type.
objectID	UOID	Required. The UOID identifying the feature object.
namespace	String	Required. Namespace of the feature implementation.
name	String	Required. Name of the feature implementation.
version	String	Required. Version of the feature implementation.
compatVersion	String	Required. Minimal version of this feature implementation that is compatible.
apiPath	String	Required. Base URL path for this feature.

MIGRATION

The resource migration allows users to relocate their entire OSN profile to another OSN platform as described in Chapter 3.5. All data and content, such as posted status updates, images, or linked friends is migrated with the profile. At the same time, connections to and from other OSN profiles, such as posted comments or friends lists, are kept intact through the use of domain-agnostic GlobalIDs. Basically, migrating an OSN profile copies the entire profile to the new OSN platform and then updates the Social Record dataset in the GSLS to point to the new profile location. The steps of the migration process from a platform P_A to P_B are depicted in Figure 3.15.

To migrate an OSN profile from a source OSN platform P_A to a new target OSN platform P_B, the protocol requires the user to register an empty stub account at the target platform P_B first. Here, the GlobalID and the AccountKeyPair of the migrated OSN profile needs to be specified at P_B.

In order to provide an overview to the user about features that can be used at P_B after a completed migration, P_B may optionally retrieve a list of supported features from the source OSN platform P_A using the feature API. This allows P_B to provide an overview about features that can be used at P_B after a completed migration to the user, allowing him to abort the migration in case important additional features he uses at P_A are not supported at P_B.

To prepare the migration, the target OSN platform P_B creates a migration object as specified in Table 3.17, which specifies the parameters of the migration. These parameters include the source OSN platform P_A, the target OSN platform P_B, the GlobalID of the migrated OSN profile, a timestamp, and a UOID to allow unique identification of the migration object via its parameter objectID. As the migration object will be sent to the empty stub account on the target OSN platform P_B, migration objects are remote content and therefore digitally signed. This way, the migration object functions as a kind of digital contract for a specific migration process between source and target platform.

Before starting to copy the OSN profile's contents, the Social Record dataset is updated in the GSLS. Here, the value of the `active` parameter is set to the objectID of the migration object, indicating that the OSN profile is currently being migrated to a new location. In this phase, the OSN profile can still be accessed at P_A by other users, but only in a read-only state. By signing the Social Record dataset including the migration object's objectID value using the user's PersonalKeyPair, the migration is explicitly authorized by the user.

The migration is then initiated by the source OSN platform P_A by sending the migration object to the target OSN platform P_B via HTTP POST. The authorization of the migration can be verified by either OSN platform via the migration object's objectID in the signed Social Record dataset. Platform P_A now starts the data transfer by encapsulating all content objects of the migrated OSN profile in migration data objects, where one or more items can be transferred with each migration item object. This way, platform P_A can limit the size of the requests being transferred and furthermore manage the entire data transfer. The format of migration data objects is described in Table 3.18.

Upon reception of a each migration data object, platform P_B extracts the encapsulated data objects, verifies their content, and stores them in the local database. Each received migration data object is acknowledged by the target OSN platform. This allows the source OSN platform P_A to track what data objects have successfully been received by the target OSN platform P_B. Once all content objects have successfully been transferred to the target OSN platform P_B, the migration is concluded.

The migration can be aborted at any point by either of the OSN platforms via a (signed) HTTP DELETE request for the migration object. In this case, all data associated with the migration is deleted from the target OSN platform P_B, causing the process to be reverted. The process then may be started over again by initiating a new migration.

In case all content objects have been successfully transferred to and received by the target OSN platform, the user's OSN profile may be deleted from the old location, including all data. If the migration fails or is aborted at any point, the Social Record is reset to its original values and already transferred data is deleted from the target OSN platform's database. Following, the Social Record dataset is updated again, where the value for the property `profileLocation` is set to point to the target OSN platform P_B, and the parameter `platformGID` is set to the target platform's GlobalID. Also, the parameter `active` is reset to its original value `1`, allowing access to all functionality of the new OSN platform. To prevent possible

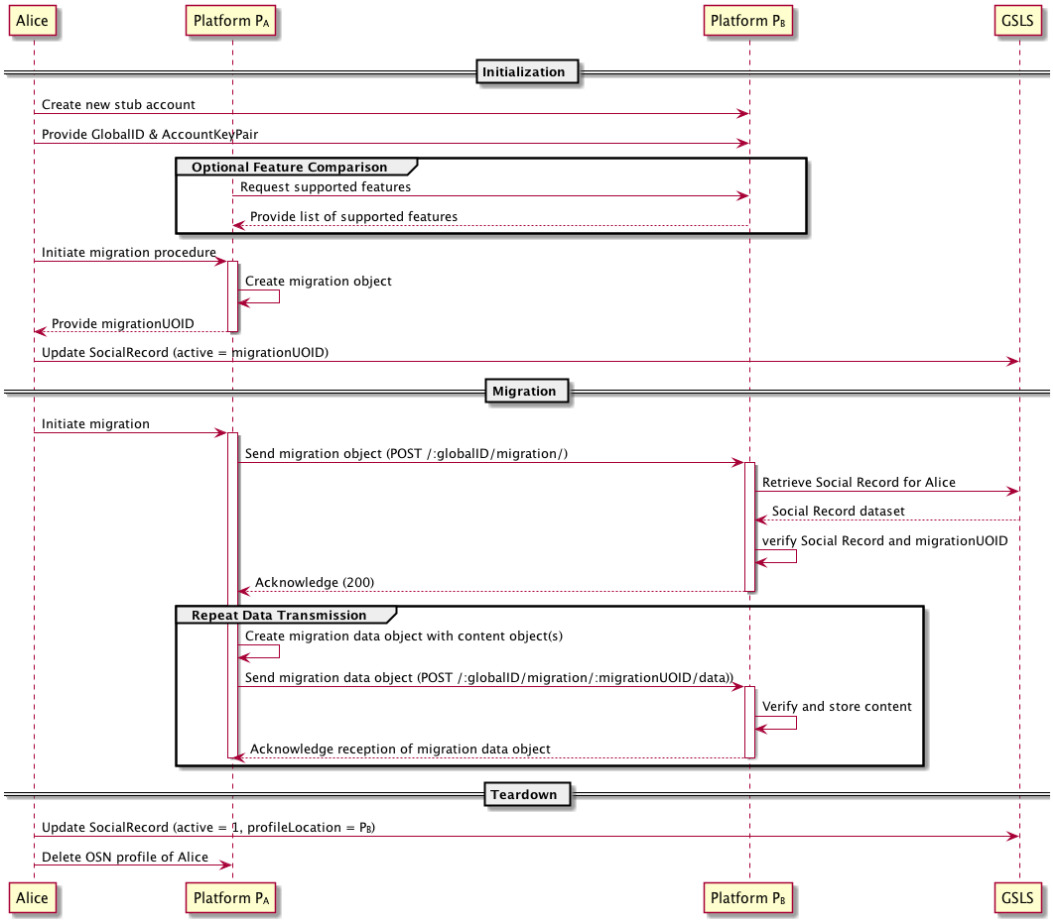


Table 3.15: Sequence of a migration: After providing the user's GlobalID and account key pair to the target OSN platform P_B, all profile data is transferred. Once all data has been successfully transferred, the profile is deleted at the old location and the Social Record is updated to point to the new location.

Table 3.16: Sonic migration API

Method	Path	Parameter
POST	/MIGRATION	Migration object
DELETE	/MIGRATION/:migrationID	
POST	/MIGRATION/:migrationID/DATA	Migration data object

Table 3.17: MIGRATION format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`migration`" specifying the object type.
objectID	UOID	Required. The UOID identifying the migration object.
migrationSource	GloablID	Required. GlobalID of the platform the profile is migrated away from.
migrationTarget	GlobalID	Required. GlobalID of the platform the profile is migrated to.
datetime	XSD-Datetime	Required. The date and time the migration object was created.
signature	Signature	Signature of the migration object.

Table 3.18: MIGRATION DATA format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`migration-data`" specifying the object type.
objectID	UOID	Required. The UOID identifying the migration data object.
targetID	UOID	Required. UOID of the migration object.
datetime	XSD-Datetime	Required. The date and time the migration object was created.
item	Array <JSONObject>	Required. Array of content objects
signature	Signature	Signature of the migration data object.

future misuse of the AccountKeyPair by the source OSN platforms at a later time, the user may revoke the key pair by updating the Social Record dataset in the GSLS, replacing it with a new one as described in Chapter 3.5.

SEARCH

The resource search allows to send search requests to other known OSN platforms. Search requests are sent following a flooding approach similar to Gnutella [222], where the search request is forwarded by each receiving OSN platform recursively. OSN platforms receiving a search request perform a local search for the search term and respond with a list of results if a match was found. The results are directly sent back to the OSN platform that initiated the search query and may be displayed to the user. The sequence of a search request is depicted in Figure 3.16. To limit the congestion effects of this approach, search requests specify a numerical limit stating how often the request is to be forwarded. Each receiving platform then decreases the value by one and abstains from forwarding the search request if the value is 0. To prevent search requests from being forwarded to the same OSN platform multiple times, each search request object is uniquely identified by a UOID. If a search request is received more than once, all subsequently received duplicates are ignored.

The actual search query is formatted as an Elastic Search query [306] and is encapsulated in a search object, which specifies further details, such as the search objects UOID, the hop limit, the querying GlobalID, and the address to which results should be sent back. The format of search request objects is described in Table 3.20. When receiving a search query object, an OSN platform performs a local search and creates a list of search result objects, in which each result is represented as a single object. Search result objects encapsulate an Elastic Search result object along with additional information such as the result's UOID and its owner's GlobalID. All search result objects are then encapsulated in a search result collection object and sent to the OSN platform that originally initiated the search query. Once search results have been received by the OSN platform from which the search has been started, the results from all OSN platforms can be displayed to the user. Due to the distributed nature of the search functionality, search results are likely to be received with a delay. The format of search result objects and search result collection objects is described in Tables 3.22 and 3.21.

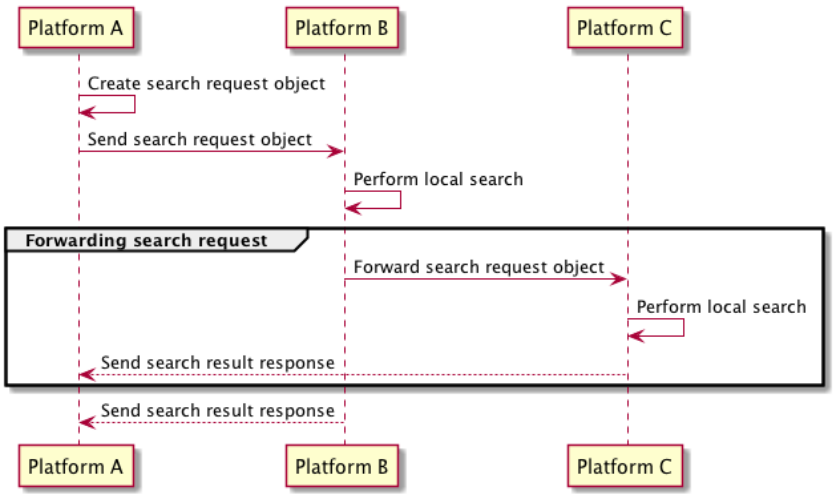


Figure 3.16: SEARCH in SONIC

Table 3.19: Resource SEARCH

Method	Path	Parameter
POST	/SEARCH	Search query object
POST	/SEARCH/:searchID/RESULT	Search result collection object

Table 3.20: SEARCH QUERY format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`search-query`" specifying the object type.
objectID	UOID	Required. The UOID identifying the search-request object.
initiatingGID	GlobalID	Required. The GlobalID of the user initiating the search request.
query	JSONObject	Required. Elastic Search query as defined in [306].
hopCount	int	Required. Hop count value. Not part of the signature.
datetime	XSD-Datetime	Required. The date and time the search-request object was created.
signature	Signature	Required. Digital signature of the search-request object.

Table 3.21: SEARCH RESULT COLLECTION format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`search-result-collection`" specifying the object type.
objectID	UOID	Required. The UOID identifying the search-result-collection object.
targetID	UOID	Required. The UOID identifying the original search-request object.
platformGID	GlobalID	Required. The GlobalID of the platform returning the search results.
datetime	XSD-Datetime	Required. The date and time the link object was created.
results	Array <search-result>	Required. The list of search-result objects.

Table 3.22: SEARCH RESULT format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`search-result`" specifying the object type.
objectID	UOID	Required. The UOID identifying the search-result object.
targetID	UOID	Required. The UOID identifying the original search-request object.
resultOwnerGID	GlobalID	Required. The GlobalID of the owner of the found content.
resultObjectID	UOID	Required. The UOID identifying the found content object.
resultIndex	int	Required. The result index as of Elastic Search.
resultType	String	Required. The result type, matching the `@type` parameter of the found content object, e.g., `profile`.
displayName	String	Human readable string representation of the search result for displaying purposes.
datetime	XSD-Datetime	Required. The date and time the link object was created.

3.7.4 Profile API

The Profile API of the Sonic protocol facilitates access to the contents of a user's OSN profile. While all features of the OSN core featureset are covered by the Profile API, additional features can be accessed via the feature extension functionality as described in 3.7.3. All requests in the Profile API address a specific user profile, which is identified via its GlobalID.

LINK

The resource link covers the creation and deletion of links between different social profiles, as well as retrieving a user's friend roster. To provide compatibility to other OSN implementations, links in Sonic are unidirectional, where bidirectional links can be composed by establishing two opposite links. To initiate a unidirectional connection between two users, a link-request object as of Table 3.24 is created by the requesting user and sent to the targeted user's OSN profile via HTTP POST. The targeted user can now react to the link request by accepting or rejecting the request, where responses are encoded in link response objects. As the decision on rejecting or accepting a request has to be made by the targeted user and therefore may be delayed, the response to a request is sent back in a separate request. If the link request is accepted, the accepting user creates a signed link object as of Table 3.23 that states that a unidirectional link exists from the requesting user, identified by the parameter `initiatingGID`, to the targeted user, identified by the parameter `targetedGID`. As the link object is signed by the targeted user, the authenticity and integrity of a claim of being connected to someone can be easily verified by anyone.

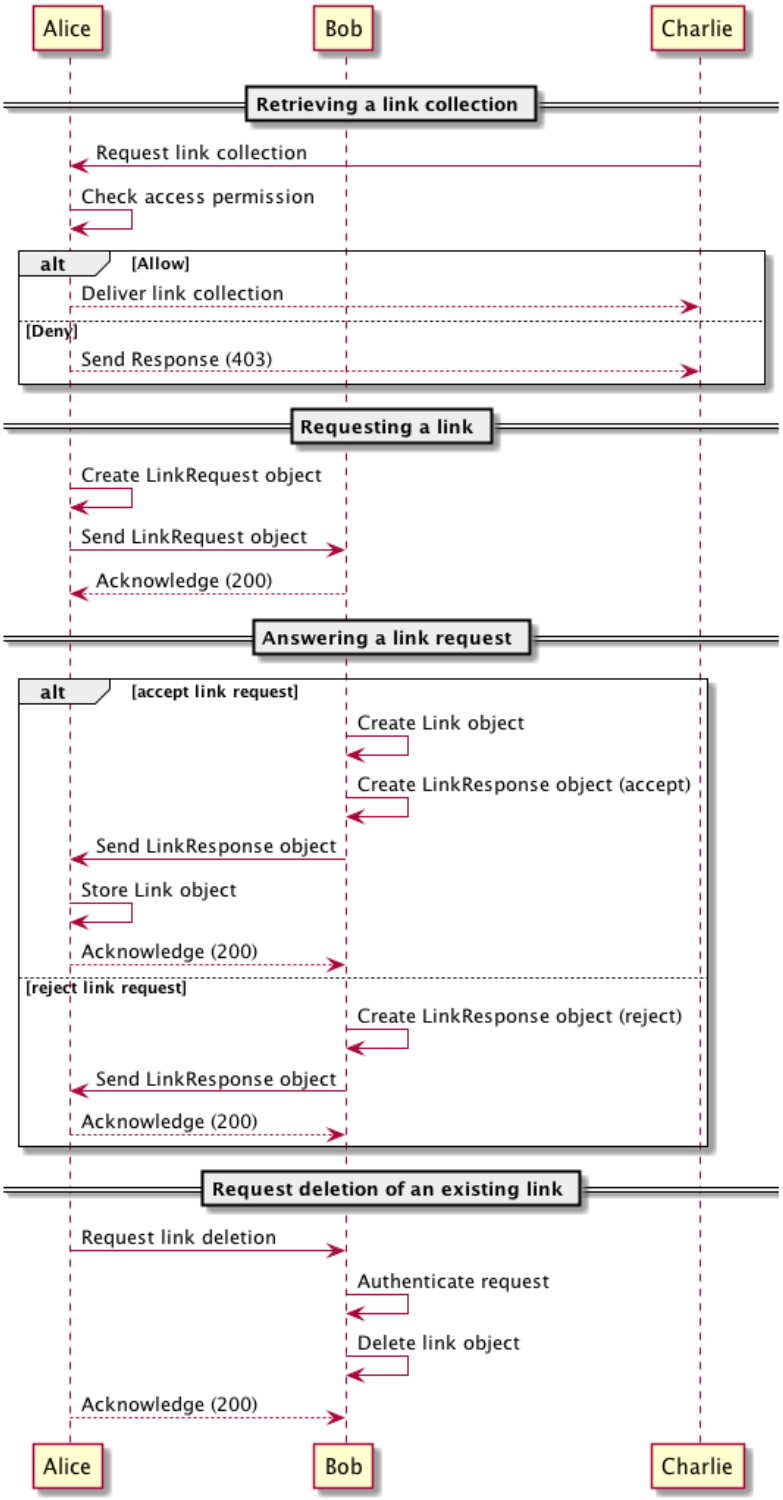


Figure 3.17: Resource LINK

The link object is then send back to the requesting user's profile via HTTP POST, encapsulated in a link-response object that references the matching link-request object and specifies whether the request was accepted or not and may furthermore include an optional textual message. The format of link response objects are described in Table 3.25. In case the request is not accepted, a link-response is returned without the enclosed link-object. All successfully established links are then stored in the OSN profile of the requesting user, where the targeted user of a link can always request deletion of the link to him by sending a HTTP DELETE request targeting the link-object's UOID. Sonic platforms must then verify whether the request for deletion is correctly signed and delete the link if it has been signed correctly by the author of the respective link object. Finally, users can retrieve a collection of all available link objects owned by a user. Using HTTP GET, a collection object can be requested comprising a list of all links to other users owned by a user. The sequence of interaction with the link resource is depicted in Figure 3.17.

Table 3.23: LINK format

Parameter	**Type**	**Description**
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`link`" specifying the link type.
objectID	UOID	Required. The UOID identifying the link object.
owner	GlobalID	Required. The GlobalID of the user who owns the link object.
datetime	XSD-Datetime	Required. The date and time the link object was created.
signature	Signture	Required. Digital signature for the link object created by the author.

Table 3.24: LINK-REQUEST format

Parameter	**Type**	**Description**
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`link-request`" specifying the object type.
objectID	UOID	Required. The UOID identifying the link-request object.
initiatingGID	GlobalID	Required. The GlobalID of the user that initiated the request.
targetedGID	GlobalID	Required. The user targeted by the link request.
datetime	XSD-Datetime	Required. The date and time the link-request object was created.
message	String	Optional textual message.

Table 3.25: LINK-RESPONSE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`link-response`" specifying the object type.
objectID	UOID	Required. The UOID identifying the link-response object.
targetID	UOID	Required. The UOID identifying the associated link-request object.
datetime	XSD-Datetime	Required. The date and time the link-response object was created.
accept	Boolean	Required. Boolean value that determines, whether a link request was accepted (true) or not (false).
message	String	A textual message for specifying a reason why a request was accepted or not.
link	Link object	Required if accept is true.

Table 3.26: Resource LINK

Method	Path	Parameter
GET	/:globalID/LINK	
GET	/:globalID/LINK/:linkID	
POST	/:globalID/LINK/:linkID/REQUEST	Link-request object
POST	/:globalID/LINK/:linkID/RESPONSE	Link-response object
DELETE	/:globalID/LINK/:linkID	

PROFILE

The resource profile allows access to a user's social profile information in form of a profile object. As social profiles comprise a variety of information in existing OSN implementations, profile objects comprise an extensive list of attributes, similar to OpenSocial's implementation of Person objects [115], where only few parameters, such as the profile owner's GlobalID and a displayable user name are required. Further parameters, such as the user's age, gender, name, a description, or even a list of postal addresses, can be omitted if desired. To further allow a high grade of personalization of a user's profile, a profile can specify an extensive list of key-value pairs to allow specifying additional arbitrary information about a user. Following this paradigm, detailed social profiles as used in Facebook or VKontakte as well as minimalistic profiles as used in Instagram or Pinterest are supported, where only the most basic information that is required to identify a user is required. With all other parameters of a profile being optional, OSN platforms may omit additional information when displaying the profile of a user. Furthermore, OSN platforms may specify access

control rules that disclose certain parts of a profile object to authorized users, for example only showing a user's birthday to users who are already linked to the profile's owner. As listed in Table 3.27, a user profile can only be read by other users, while profile creation, editing, and deletion are not supported by the API itself. Table 3.28 describes the base profile format as returned by the API.

Table 3.27: Resource PROFILE

Method	Path	Parameter
GET	:globalID/PROFILE	

Table 3.28: PROFILE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`profile`" specifying the object type.
objectID	UOID	Required. The UOID identifying the profile object.
globalID	GlobalID	Required. The GlobalID identifying the profile's owner.
displayName	String	Required. A human-readable username to used for display purposes.

ACTIVITY

The resource activity describes functionality that allows publishing a stream of activities, being status updates, check ins, or shared content, that make up a user's activity stream. The format of individual activity objects is based on Activity Streams 2.0 [102], where an activity object encapsulates the actual JSON-formatted Activity Streams object and augments it with information required for addressing content objects in the Sonic OSNF. This way, the full potential of the Activity Streams 2.0 standard can be used to describe a user's actions. Activity objects are remote content as they can be published in a remote user's OSN profile. Therefore, activity objects comprise a digital signature of the respective author. Remotely stored activities can be updated by their author via HTTP PUT, where the activity's contents are overwritten with a new activity object.

As listed in Table 3.29, activities can be created, read, updated, and deleted by other users, where update and delete requests can only be created by the respective object's author. Furthermore, activity objects can be liked, commented on, and tagged, where a collection of likes, comments and tags for an individual activity object can be retrieved via the API. Table 3.30 specifies the format of activity objects, which can be accessed individually or as a collection.

Table 3.29: Resource ACTIVITY

Method	Path	Parameter
GET	:globalID/ACTIVITY	
GET	:globalID/ACTIVITY/:activityID	
GET	:globalID/ACTIVITY/:activityID/LIKE	
GET	:globalID/ACTIVITY/:activityID/COMMENT	
GET	:globalID/ACTIVITY/:activityID/TAG	
POST	:globalID/ACTIVITY	activity object
PUT	:globalID/ACTIVITY/:activityID	activity object
DELETE	:globalID/ACTIVITY/:activityID	

Table 3.30: ACTIVITY format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`activity`" specifying the object type.
globalID	GlobalID	Required. The GlobalID identifying the owner of the activity.
displayName	String	Required. A human-readable username to used for display purposes.
objectID	UOID	Required. The UOID identifying the activity object.
author	GlobalID	Required. The GlobalID identifying the profile's owner.
datetime	XSD-Datetime	Required. The date and time the activity object was created.
activity	JSON Object	Required. Serialized Activity Streams 2.0 object.
signature	Signature	Required. Digital signature for the activity object created by the author.

COMMENT

The resource comment allows users to comment on content, where comments are textual messages that allow to express a detailed opinion, explanation, or feedback. Comment objects comprise a textual message, and furthermore a timestamp indicating the date and time of the comment's creation, and the GlobalID of the comment's author. To uniquely specify what a comment is targeting, comment objects specify a parameter `targetID` as a reference to the targeted content object's UOID. Comment objects are remote content, as they can reference content objects published in other user's OSN profiles, which causes

the comment to be stored in a OSN profile other than the author's. As of this, comment objects specify a mandatory signature created by the author of the content. The signature allows to verify the integrity of the comment object and prohibits that the comment's owner alters the content's data without permission of its author. Comment objects can be updated by their author, where the comment's text is overwritten and a parameter `dateUpdated` is set to specify the date and time at which the object was updated. The complete format of comment objects is described in Table 3.31. As described in Table 3.32, comment objects can be created, read, updated, and deleted by their author. As users can be tagged in comments and furthermore like comments, a list of likes or tags can be requested.

Table 3.31: COMMENT format

Parameter	**Type**	**Description**
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`comment`" specifying the object type.
objectID	UOID	Required. The UOID identifying the comment object.
targetID	UOID	Required. The UOID of the content object that this comment object targets.
body	String	Required. The comment message.
author	GlobalID	Required. The GlobalID of the author of the comment object.
datetime	XSD-Datetime	Required. The date and time the comment object was created.
dateUpdated	XSD-Datetime	The date and time the comment object was updated.
signature	Signature	Required. Digital signature for the comment object created by the author.

Table 3.32: Resource COMMENT

Method	**Path**	**Parameter**
GET	:globalID/COMMENT/:commentID	
GET	:globalID/COMMENT/:commentID/TAG	
GET	:globalID/COMMENT/:commentID/LIKE	
POST	:globalID/COMMENT/	Comment object
PUT	:globalID/COMMENT/:commentID	Comment object
DELETE	:globalID/COMMENT/:commentID	

LIKE

The resource like implements the OSN feature like as described in Chapter 3.4.4 and allows users to express that they like certain content that has been published. As like objects do not comprise content by themselves and merely represent an indication that a user likes certain content, like objects simply specify a the liked content's UOID, a timestamp indicating the date and time of creation, and the GlobalID of the like's author. Like objects are remote content and therefore specify a mandatory signature created by the author of the like object. The complete format of like objects is described in Table 3.33. As described in Table 3.34, like objects can be created, read, and deleted by their author, yet cannot be updated.

Table 3.33: LIKE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`like`" specifying the object type.
objectID	UOID	Required. The UOID identifying the like object.
targetID	UOID	Required. The UOID of the content object the like object targets.
author	GlobalID	Required. The GlobalID of the user that created the like object.
datetime	XSD-Datetime	Required. The date and time the like object was created.
signature	Signature	Required. Digital signature for the like object created by the object's author.

Table 3.34: Resource LIKE

Method	Path	Parameter
GET	:globalID/LIKE/:likeID	
POST	:globalID/LIKE/	Like object
DELETE	:globalID/LIKE/:likeID	

TAG

The resource tag implements the OSN feature tag as defined in Chapter 3.4.4 and allows users to express that a user is linked to the content object in a certain way. An example is users being tagged in a photo or in a status update describing an event. Tag objects simply express which user is associated with what content and therefore specify a mandatory tagged GlobalID and the targetID of the content the tagged user is tagged in. Furthermore, tag objects specify the creator of the object, the date and time of creation, and an objectID. Tag objects are remote content and therefore specify a mandatory signature created by the author of the tag object. The complete format of tag objects is described in Table 3.36. As described in Table 3.35, tag objects can be created, read, and deleted by their author, yet cannot be updated.

Table 3.35: Resource TAG

Method	**Path**	**Parameter**
GET	:globalID/TAG/:tagID	
POST	:globalID/TAG/	Tag object
DELETE	:globalID/TAG/:tagID	

Table 3.36: TAG format

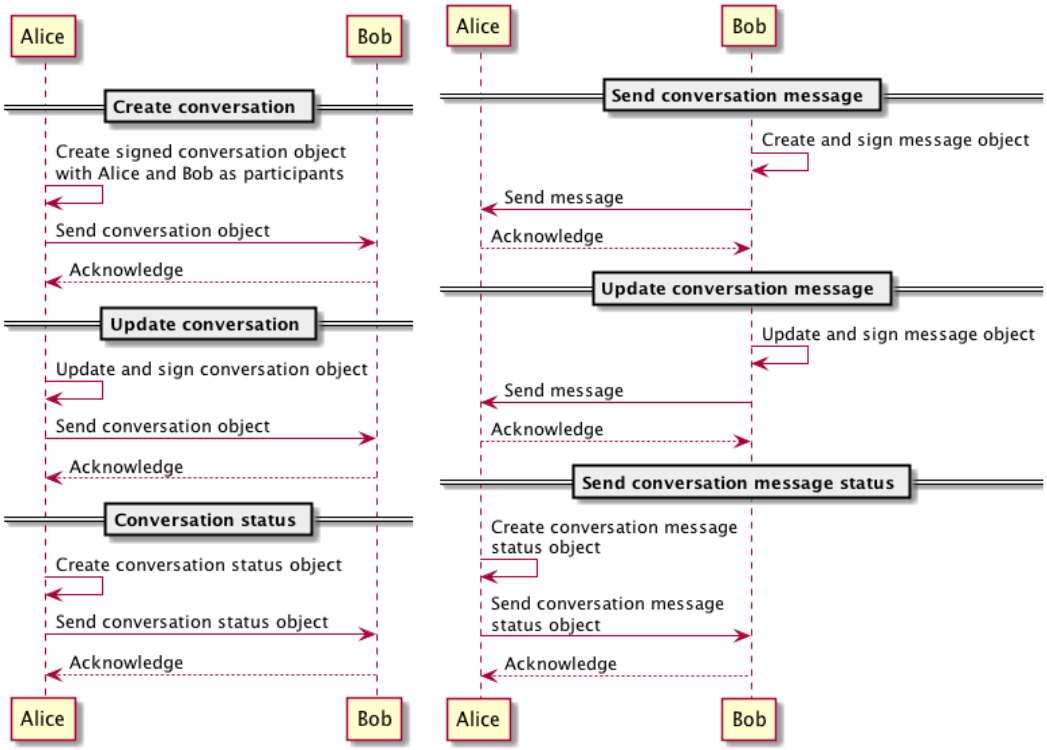
Parameter	**Type**	**Description**
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed JSONLD-value "`tag`" specifying the object type.
objectID	UOID	Required. The UOID identifying the tag object.
targetID	UOID	Required. The UOID of the content object the tag object targets.
tag	GlobalID	Required. The GlobalID of the user tagged by the tag object.
author	GlobalID	Required. The GlobalID of the user that created the tag object.
datetime	XSD-Datetime	Required. The date and time the tag object was created.
signature	Signature	Required. Digital signature for the tag object created by the object's author.

CONVERSATION

The resource conversation describes the exchange of messages between two or multiple users. Each conversation encapsulates a conversation's state and the messages exchanged between all participants. Upon initiation, an empty conversation object as described in Table 3.39 is created, with the initiating user as its administrator and optionally one or more other users as participants. The conversation object is then sent to all participants via HTTP POST. After a conversation has been initialized, a conversation's state can only be changed by the administrator, where conversation objects are remote content and digitally signed by their administrator. Conversation administrators can change a conversation's topic and add or remove participants. When a conversation's state is changed, an update is broadcast to all participants of the conversation. Accordingly, when a new participant is added to a conversation, the conversation object is replicated to the new user's OSN profile, causing all further updates to the conversation being send to him as well. Actions of participants of the conversation are broadcast the all participants via conversation status updates, where conversation status updates can be an update of the conversation topic or adding or removing participants. Each conversation status change is communicated via a conversation status object as described in Table 3.40, which is sent via HTTP POST to all conversation participants. Conversation status updates are informal and do not invoke any change of the conversation's state. Figure 3.37a depicts the communication flow for conversation and conversation status messages.

Communication in conversations is implemented via messages, where each message is broadcast by its author to all participants. After being sent, messages can be updated by their author. To announce that a message has been received, read, or (locally) deleted by a participant, message status updates are broadcast. A message status is encoded as described in Table 3.42. This allows to track which participants of a conversation have received or read a specific message, or even deleted it manually. Similar to conversation status updates, message status updates are informal and do not invoke changes in a conversation or message. As conversation data is replicated to one or multiple other OSN profiles, conversation objects, message objects, and status updates for both conversations and messages are implemented as remote content and digitally signed by their respective author. This allows all participants to monitor who received or read a specific message. Figure 3.37b depicts the communication flow for messages and message statuses.

The individual endpoints for the resource conversation are described in Table 3.38. Figure 3.18 depicts a sequence of message flows for an example conversation between three participants Alice, Bob, and Charlie. In this example, participant Alice initiates the conversation with her and Bob as participants. As the initiator and administrator of the conversation, Alice then adds Charlie to the conversation and sends the conversation object to him. As the conversation object has been changed, the updated object is also sent to Bob. Alice then creates a conversation status object stating that Charlie has been added as a participant. The conversation status object is broadcast to both Bob and Charlie,



(a) Message flow for conversation and conversation status messages.

(b) Message flow for conversation-message and conversation-message-status messages.

Table 3.37: Message flow for resource conversation.

Table 3.38: Resource CONVERSATION

Method	Path	Parameter
POST	:globalID/CONVERSATION/	conversation object
PUT	:globalID/CONVERSATION/:conversationID	conversation object
DELETE	:globalID/CONVERSATION/:conversationID	
POST	:globalID/CONVERSATION/:conversationID/ CONVERSATION-STATUS	conversation-status object
POST	:globalID/CONVERSATION/:conversationID/ MESSAGE	message object
PUT	:globalID/CONVERSATION/:conversationID/ MESSAGE/:messageID	message object
POST	:globalID/CONVERSATION/:conversationID/ MESSAGE/:messageID/STATUS	conversation-message-status object

announcing that a new participant has been added. Next, Charlie creates a message object which he sends to both Alice and Bob. Both Alice and Bob create a conversation message status object stating that the message has been received and broadcast this conversation message status to all other participants. When the message is actually read by Alice, she creates a new conversation message status to announce that she read the message. The new status is also broadcast to all other participants of the conversation. This example demonstrates, how communication between multiple communication partners is implemented by the Sonic protocol, where messages are broadcast to all communication partners.

Table 3.39: CONVERSATION format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`conversation`" specifying the object type.
objectID	UOID	Required. The UOID identifying the conversation object.
topic	String	Topic of the conversation.
owner	GlobalID	Required. The GlobalID identifying the conversation's creator.
members	Array<GlobalID>	Required. Array of GlobalIDs of all participants. At least one participant needs to be listed.
datetime	XSDDatetime	Required. Datetime of creation or last update of the conversation.
signature	Signature	Required. Digital signature for the conversation object created by the object's author.

Table 3.40: CONVERSATION STATUS format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`conversation-status`" specifying the object type.
objectID	UOID	Required. The UOID identifying the conversation-status object.
targetID	UOID	Required. UOID of the conversation the conversation status object targets.
status	String	Status conveyed by this conversation-status. Can be `ADDED`, `REMOVED`, `INVITED`, `JOINED`, `LEFT`, or `DECLINED`.
author	GlobalID	Required. The GlobalID identifying the status' author.
targetGID	GlobalID	Required. The GlobalID of the user targeted by the conversation-status.
datetime	XSDDatetime	Required. Datetime of creation of the conversation-status.
signature	Signature	Required. Digital signature for the conversation status object created by the object's author.

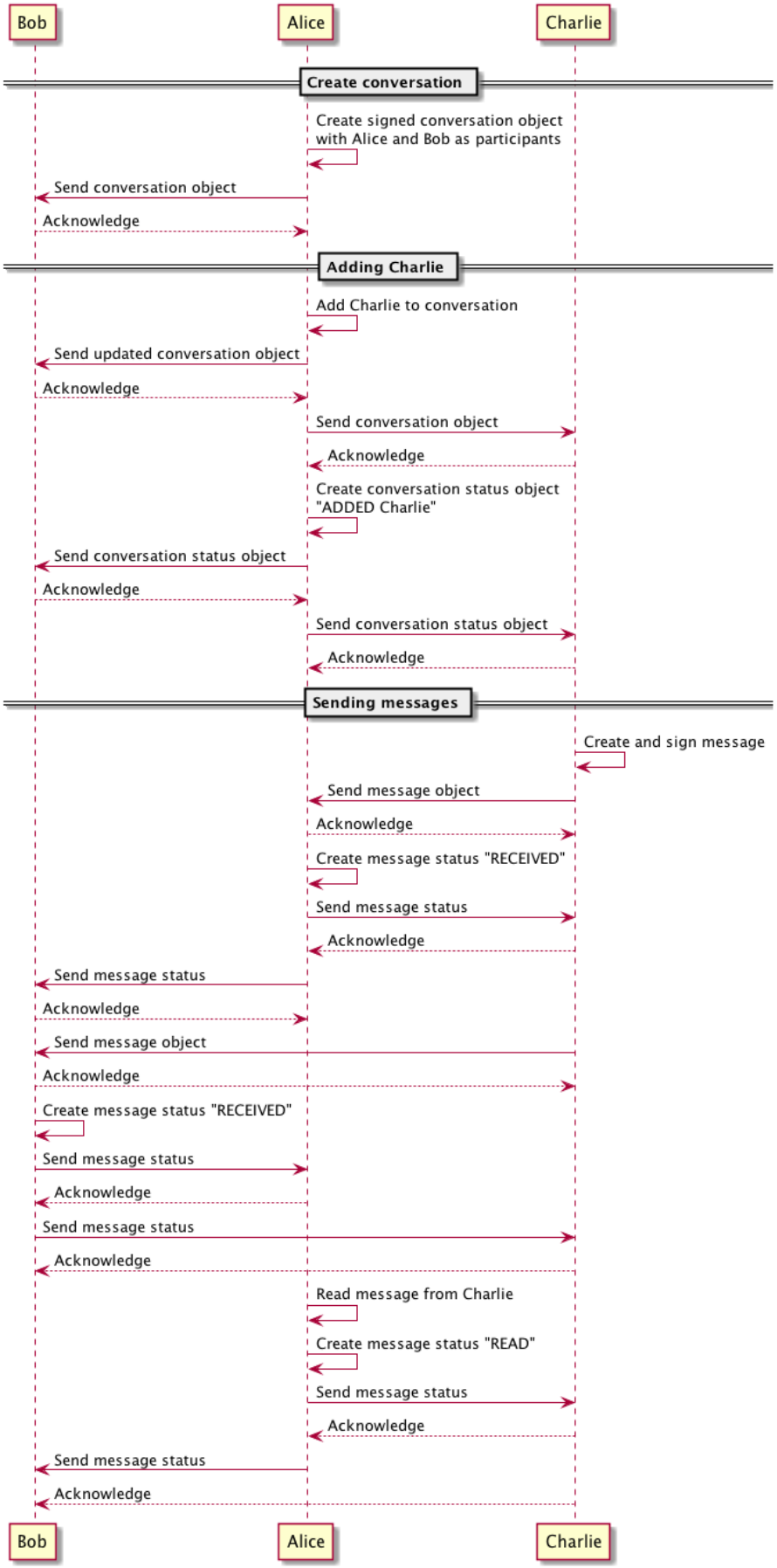


Figure 3.18: Example sequence of communication flow for a conversation between three participants Alice, Bob, and Charlie.

Table 3.41: CONVERSATION MESSAGE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`conversation-message`" specifying the object type.
objectID	UOID	Required. The UOID identifying the conversation-message object.
targetID	UOID	Required. UOID of the message the conversation message status targets.
title	String	Title of the message.
body	String	Required. The message body.
author	GlobalID	Required. The GlobalID identifying the author of the message.
datetime	XSDDatetime	Required. Datetime of creation or last update of the conversation-message.
status	String	Required. Local status of the message, used to mark messages. Can be `NEW`, `READ`, or `DELETED`.
signature	Signature	Required. Digital signature for the conversation message object created by the object's author.

Table 3.42: CONVERSATION MESSAGE STATUS format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`conversation-status`" specifying the object type.
objectID	UOID	Required. The UOID identifying the conversation-status object.
targetID	UOID	Required. UOID of the targeted message.
conversationID	UOID	Required. The UOID of the conversation the targeted message belongs to.
status	String	Required. Status conveyed by this conversation-message-status. Can be `RECEIVED`, `READ`,
author	GlobalID	Required. The GlobalID identifying the message status' author.
datetime	XSDDatetime	Required. Datetime of creation of the conversation-message-status.
signature	Signature	Required. Digital signature for the conversation message status object created by the object's author.

IMAGE

The resource image implements support for images to allow users to publish photos and pictures. As described in Table 3.44, an image object specifies a parameter objectID, the GlobalID of its owner, date and time of its creation, the images width and height and an optional title and description of the image.

In order to support portability of the image data and prevent having image metadata and the actual image file stored in separate locations, all data is stored in the Sonic image object. The actual binary image data is stored encoded as base64 using the data URI scheme [307]. As base64-encoded images are usually approximately 33% larger as the original binary image file [308], the data URI is compressed using gzip [309], resulting in only a minimal inflation of file size [308]. Using data URIs, images can be directly displayed in a browser, while extraction of the original binary image file is possible. Optionally, image objects can store a reduced-size thumbnail of the image as a separate gzipped data URI. Image objects are local content and therefore do not specify a signature. As described in Table 3.43, image objects can only be read from their owner's OSN profile, allowing him to exert full control over each image's distribution.

Table 3.43: Resource IMAGE

Method	Path	Parameter
GET	:globalID/IMAGE/:imageID	

Table 3.44: IMAGE format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`image`" specifying the object type.
objectID	UOID	Required. The UOID identifying the image object.
owner	GlobalID	Required. The GlobalID identifying the image's owner.
datetime	XSD-Datetime	Required. The date and time the link object was created.
title	String	Title for the image.
description	String	Textual description of the image contents.
imageData	String	Required. The base64-encoded image content as a data URI.
imageThumbnail	String	The base64-encoded thumbnail content as a data URI.
imageWidth	Integer	Required. Width of the image in pixels.
imageHeight	Integer	Required. Height of the image in pixels.

Supporting data formats

Besides the aforementioned data formats for encapsulating OSN profile information, additional general data formats are defined that are independent of OSN features.

- **Collection** Collection objects are used to describe lists of content objects. In comparison to simple arrays, collection objects specify information about the enclosed content, the owner of the collection, and are addressable via a UOID. The format of the collection object is described in Table 3.45.
- **Signature** Remote content objects in Sonic comprise a digital signature that guarantees that the original content is unaltered. Signature objects are encoded similar to the JSON-based Linked Data Signature standard [310]. Besides the actual base64-encoded digital signature, signature objects comprise information about the creator of the signature and the date and time the object was signed. Furthermore, signature objects comprise a random value for added security [301][302]. Signature objects are linked to the content object they were created for by specifying the targeted content's UOID. The format of the signature object is described in Table 3.46.

Table 3.45: Collection object format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`collection`" specifying the object type.
objectID	UOID	Required. The UOID of the collection object.
owner	GlobalID	Required. The GlobalID identifying the collection's owner.
collectionType	String	Required. Type of the items stored in the collection.
items	Array <JSONObject>	Required. List of items in the collection

Table 3.46: Signature object format

Parameter	Type	Description
@context	String	Required. Fixed value "`http://sonic-project.net/`".
@type	String	Required. Fixed value "`signature`" specifying the object type.
targetID	UOID	Required. UOID of the content object the signature was created for.
creatorGID	GlobalID	Required. GlobalID of the user creating the signature.
timeSigned	XSD-Datetime	Required. XSD-Datetime specifying the date and time of creation of the signature.
random	String	Required. Random value for added security [301][302].
signature	String	Required. Digital RSA signature of the content object. The signature comprises the content object's contents except for the signature object itself.

4 Implementation

The solutions presented and described in Chapter 3 have been implemented to provide a proof of concept and reference implementation, as well as to evaluate the concept and design. This chapter describes the individual implementations, including the respective procedure for building and installing the individual projects, including their configuration and operation. First, the implementation of the GSLS is described, followed by a description of the Sonic SDK. Finally, the implementation of the Sonic OSN is described.

4.1 GSLS

The GSLS plays a crucial role in registering and connecting to users in the Sonic OSNF. As described in Section 3.5, users register a GlobalID which is generated from an RSA public key pair using SHA256 with PBKDF#2. This way, GlobalIDs are created without a central authority controlling the process. The GlobalID and public key of a user are stored in the Social Record, a dataset that furthermore comprises information about the actual OSN profile's URL. The Social Record is digitally signed by its creator and stored as a JWT (JWS) to allow verification of the dataset's integrity. Resolving a known GlobalID to the associated Social Record hence provides all information required to initiate a connection to a user's OSN profile.

The GSLS functions as a decentralized and globally distributed directory service that stores all Social Record datasets. To achieve a fully decentralized operation of the GSLS, data is stored in a DHT, a key-value store that is distributed over all participating GSLS nodes. For retrieval of a Social Record, its GlobalID is used as the lookup key, where the Social Record is tethered to its GlobalID via the enclosed public key as described in Chapter 3.5. As by design of the GSLS, no central authority exists that authenticates users or authorizes requests, Social Record datasets stored in the GSLS are publicly readable, while authorization for write access (create, update) is verified by the authenticity of the JWT's digital signature. This way, every user and service can request and access any Social Record, while only a GlobalID's creator and owner can update his own dataset.

For ensuring availability of both the service as well as the managed datasets, the GSLS relies on the self-management of the underlying DHT. As described in Chapter 3.5, DHTs implement self-organization and self-healing capabilities that result in a high resilience against individual node failures and various directed attacks against the network. For data availability, the GSLS employs *indirect replication* of the datasets. Indirect replication of data distributes each

dataset to multiple nodes using a set of alternative KBR procedures. This way, even when several nodes fail, availability of a dataset is ensured.

4.1.1 Functionality

The GSLS implements functionality to write new Social Records to the DHT, update existing ones, or retrieve a Social Record identified by its GlobalID. When resolving a GlobalID, the GSLS attempts to retrieve the signed JWT stored for this GlobalID from the DHT. In case a JWT is found, the enclosed Social Record dataset is extracted and the JWT's signature is verified. Furthermore, the data format of the Social Record is checked, specifically verifying whether the GlobalID is valid. This is done by hashing the values of `personalPublicKey` and `salt`, as described in Chapter 3.5, creates the specified GlobalID. In case no error occurred, the signed JWT is returned to the requesting entity. When a new identity is created or an existing one is updated, the respective Social Record is sent to the GSLS as a signed JWT. The GSLS then extracts the public key from the dataset in order to verify the signature. Furthermore, the dataset's format and GlobalID are checked similarly to the checks performed when retrieving a Social Record. Only if no error occurred, the JWT is written to the DHT, thus making it globally available.

4.1.2 Implementation

The GSLS is implemented in Java 8[1] as a Spring Boot[2] application. Spring Boot *"takes an opinionated view of building production-ready Spring applications"* and *"favors convention over configuration"* and hence allows to build scalable and robust software services based on an industry standard-grade framework. The DHT functionality of the GSLS is provided by the open source project TomP2P[3] [311], which provides an easy way to connect nodes into a structured overlay network based on Kademlia [227]. The project uses Apache Maven[4] to manage code dependencies and the building process of the project and Docker[5] to build runnable containers for easy and platform-independent service deployment. The requirements for building and running the GSLS service are summarized in Table 4.1. Further requirements, such as the Spring Boot or TomP2P libraries, are managed by Maven. The source code of the GSLS has been released under the MIT open source license[6] and is available on GitHub via `https://github.com/sgoendoer/gsls`.

Build

To build the project, the Maven plugins `clean` and `install` are used, which create a single jar file including all dependencies according to the Maven project configuration as specified in the project's `pom.xml` file. To ensure

[1] Java 8: `https://www.java.com/en/download/faq/java8.xml`. Accessed: 30.8.2017
[2] Spring Boot: `https://projects.spring.io/spring-boot/`. Accessed: 30.8.2017
[3] TomP2P: `https://tomp2p.net/`. Accessed: 30.8.2017
[4] Apache Maven: `https://maven.apache.org/`. Accessed: 30.8.2017
[5] Docker: `https://www.docker.com/what-docker`. Accessed: 30.8.2017
[6] MIT Software License: `https://opensource.org/licenses/MIT`. Accessed: 30.8.2017

Table 4.1: GSLS Requirements

Requirement	Version	Build/Run
Java 8 JDK	8	✓/ ✗
Java 8 JRE	8	✗/ ✓
Maven	4	✓/ ✗
Docker	17	✓/ ✓

the functionality of the code, a set of JUnit[7] tests are implemented that are automatically run by Maven during compilation of the project. Finally, a Docker container is created that runs the GSLS service once started. Listing 4.1 lists the commands required to build the project and create a runnable Docker container identified as `sonic/gsls:0.2.5`.

```
git clone https://github.com/sgoendoer/gsls.git
git ckeckout tags/0.2.5
mvn clean
mvn install
docker build -t sonic/gsls:0.2.5 .
```

Listing 4.1: Building the GSLS server from sources with Maven and Docker

Run

To run the GSLS service, the Docker container is started via the command `docker run`. The parameters `-d` and `-restart=always` start the container as a background service that runs in detached mode and is restarted in case of an unexpected exit or crash. The container exposes the ports 4001 (TCP and UDP) and 4002 (TCP), where 4001 is used by the TomP2P overlay management of connections with other nodes and exchange of messages, and 4002 is used for the REST interface used by clients to access the service. Alternatively, the compiled jar file can be started directly via `java -jar gsls-0.2.5.jar -h`.

```
docker run -d -p 4001:4001/tcp -p 4001:4001/udp -p 4002:4002/tcp
        --restart=always sonic/gsls:0.2.5
```

Listing 4.2: Building the GSLS server with Maven and Docker

Configuration

The GSLS features a list of configuration options that control specific behavior of the service during runtime. The individual configuration parameters are listed in Table 4.2. If no configuration values are explicitly specified during service startup, default values are used. The most important configuration parameter `connectNode` is specifying an existing GSLS node to connect to during

[7]JUnit testing framework: `http://junit.org`. Accessed: 1.9.2017

startup. If at the specified location no GSLS service is detected, the GSLS will bootstrap a new DHT. To prevent a separation the GSLS into multiple distinct DHTs due to misconfiguration or network connectivity issues, GSLS nodes will periodically attempt to re-connect to other known GSLS nodes. In case two separate DHT overlays were created at some point, a reconnect will join both DHT overlays back into one. The configuration parameter defaults to an IP address managed by Technische Universität Berlin at which a GSLS node is running. The configuration parameter `portREST` specifies the TCP port at which the GSLS service is listening for incoming HTTP requests to retrieve, create, or update Social Records. The configuration parameter `logPath` determines to what path log files created by the service are written. The parameter defaults to `logs`, which is created in the base folder of the service. When running the service as a Docker container, log messages are managed by the Docker service and can be accessed via the command `docker $container logs`, where `$container` is the container's id. The configuration parameter `networkInterface` allows to specify the network interface to be used by TomP2P for maintaining and managing the DHT overlay.

Table 4.2: GSLS Configuration Parameters

Requirement	Default	Explanation
`connectNode`	`130.149.22.220`	IP address to connect to during service startup.
`portREST`	`4002`	Port of the HTTP REST interface.
`logPath`	`logs`	Path of the logfiles created by the service
`networkInterface`	`eth0`	Network interface to use for DHT connectivity.

4.2 Sonic SDK

Sonic can be seen as a framework that marshals format and encapsulation of data as well as the exchange of data objects between OSN platforms. Here, the Sonic SDK provides a reference implementation for the Sonic protocol and associated functionality. As Sonic simply marshals how content and messages are exchanged between OSN platforms, any OSN platform implementing the Sonic protocol only needs to provide functionality to send and receive such requests and responses, where content needs to be formatted according to the protocol specifications. A platform hence also needs to implement functionality to format and parse information according to the data formats specified by the Sonic protocol and furthermore provide means to digitally sign and verify the data. The Sonic SDK aims to automate the entire process of sending and receiving requests and responses, allowing developers of OSN platforms to integrate Sonic into new and existing projects with little overhead.

The Sonic SDK is implemented in PHP, a popular scripting language with OOP features. PHP is used by popular open source projects such as GnuSocial, Mastodon, or Friendica and even is used in parts of Facebook [312]. Furthermore,

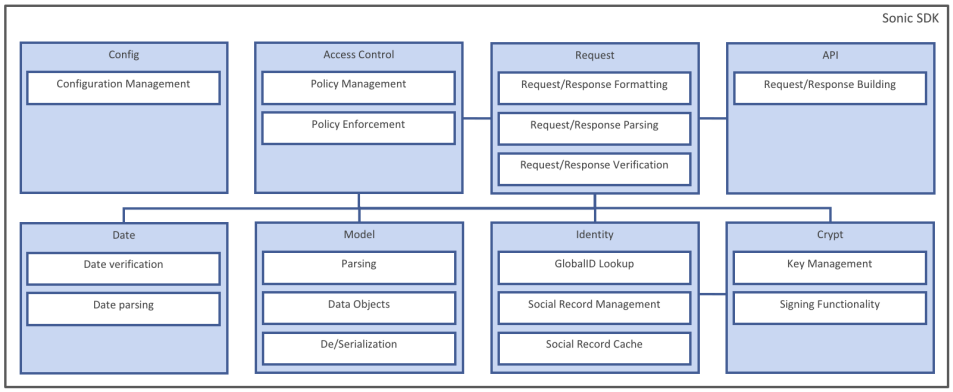


Figure 4.1: Overview of the package structure and dependencies in the Sonic SDK.

a broad variety of open source frameworks and libraries based on PHP exist, allowing a PHP-based SDK to be used in a broad variety of services and platforms.

4.2.1 Functionality

The Sonic SDK provides functionality that allows developers to automate tasks and procedures required by the Sonic protocol. For example, the Sonic SDK provides means to create and resolve GlobalIDs and Social Record datasets via the GSLS in an automated fashion. Similar to the concept of DNS caches, the Sonic SDK provides functionality to cache Social Record datasets in order to minimize the amount of requests for resolving GlobalIDs. Furthermore, the Sonic SDK provides functionality to create, format, and sign, as well as parse and validate request and response messages to be sent or being received. Request and response messages can be parsed, where transmitted content objects can be transfered to data objects as specified by the Sonic content model. Digital signatures of messages and content objects are automatically created and verified, where the keys required for the verification of the digital signatures are retrieved via the GSLS. Finally, the Sonic SDK implements support management of access control policies in order to allow specification of access control policies and rules as well as algorithms for policy evaluation.

The Sonic SDK implements a main class `Sonic.php` that initializes all dependencies in the SDK according to implementation and configuration. Functionality for the various domains is encapsulated into separate packages as depicted in Figure 4.1, of which each is responsible for a distinct type of functionality. The packages of the Sonic SDK are *AccessControl*, *API*, *Identity*, *Config*, *Crypt*, *Date*, *Model*, *and Request*, encapsulating functionality as follows:

AccessControl

The package AccessControl implements optional access control functionality for the Sonic SDK. The implementation allows to specify access control rules for accessing resources, and furthermore allows to control access to individual content objects. For evaluation and enforcement of access control rules, a class AccessControlManager is provided that implements functionality for

interpretation and application of rules. If configured in the Sonic SDK instance, access control rules are applied and evaluated when accessing all resources and content objects.

Identity

The package Identity encapsulates classes and functionality for management of a user's identity and resolving other users GlobalIDs. While PersonalKeyPairs of users should not be made available to an OSN platform in general, the Sonic SDK implements functionality to create and manage both PersonalKeyPairs and AccountKeyPairs, and furthermore allows to create and manage Social Record datasets. For verification purposes, the package Identity implements functionality to verify integrity and authenticity of Social Records and also implements functionality for interaction with the GSLS and caching of Social Record datasets.

Model

The package Model provides classes and functionality to format and parse objects specified by the Sonic protocol and architecture. Model classes implement serialization functionality and deserialization via builder classes [313] to ensure automated validation of enclosed information and signatures.

Crypt

The package Crypt provides functionality for creating and handling of cryptographic key pairs, creation of random strings, as well as creation and verification of digital cryptographic signatures.

Request

The package Request encapsulates functionality for creating, parsing, and verifying incoming and outgoing requests and responses. For outgoing requests and responses, the mandatory Sonic HTTP headers are automatically set and the entire request is digitally signed by the sending entity, being either the user or the OSN platform. For incoming requests and responses, data formats and digital signatures are automatically verified, so that invalid messages received by an OSN platform are directly intercepted.

API

The package API implements functionality to create requests for all resources specified by the Sonic protocol. Requests can be created and dispatched via static methods, thus minimizing the interaction with other packages and classes for developers.

Date

The package Date encapsulates management of datetime functionality as well as creation and verification of timestamps used by the Sonic SDK.

Config

The package Config encapsulates management of the SDK's configuration. The SDK's configuration is stored and managed by a class implementing the Singleton pattern [313], so that it can directly be accessed from within the project.

4.2.2 Implementation

The Sonic SDK has been implemented as a Composer[8] project, allowing dependencies to be installed and updated automatically. The Sonic SDK is implemented in PHP 7.0[9] and is fully compatible with PHP 5.6, 7.1, and 7.2, where OpenSSL[10] is required for signature and key management and cURL[11] is required for request handling. The SDK includes a set of PHPUnit[12] tests, where a set of tests has been implemented for each package.

Table 4.3: Sonic SDK Requirements

Requirement	Version
PHP	5.6+ / 7.0+
cURL	7.20.0+
OpenSSL	1.0.0+
composer	1.0+

The Sonic SDK is instantiated via the main class `Sonic.php`, which implements the Singleton pattern [313]. A basic example for initialization and configuration of the Sonic SDK is described in Listing 4.3.

[8] Composer: `https://getcomposer.org/`. Accessed: 7.9.2017
[9] PHP: `https://www.php.net`. Accessed: 7.9.2017
[10] OpenSSL: `https://www.openssl.org/`. Accessed: 7.9.2017
[11] cURL: `https://curl.haxx.se/`. Accessed: 7.9.2017
[12] PHPUnit test framework: `https://www.phpunit.de`. Accessed: 1.9.2017

```php
require_once(__DIR__ . '/vendor/autoload.php');

use sgoendoer\Sonic\Sonic;
use sgoendoer\Sonic\Config\Config;
use sgoendoer\Sonic\Identity\EntityAuthData;
use sgoendoer\Sonic\Identity\SocialRecord;
use sgoendoer\Sonic\Identity\SocialRecordManager;

try {
  // Sonic requires the Social Record of the OSN platform for
  // initialization.
  $platformSR = '{"socialRecord":{"@context": --truncated-- }';
  $sr = SocialRecordManager::importSocialRecord($platformSR);
  $platformSocialRecord    = $sr['socialRecord'];
  $platformAccountKeyPair  = $sr['accountKeyPair'];
  $platformPersonalKeyPair = $sr['personalKeyPair'];

  // Import SocialRecord for a user Alice
  $aliceSR = '{"socialRecord":{"@context": --truncated -- }';
  $sr = SocialRecordManager::importSocialRecord($aliceSR);
  $userSocialRecord        = $sr['socialRecord'];
  $userAccountKeyPair      = $sr['accountKeyPair'];
  $userPersonalKeyPair     = $sr['personalKeyPair'];

  // Configuring SDK. Default values are used otherwise.
  Configuration::setVerbose(1);

  // Initialize the Sonic SDK. The SDK's context is set "platform"
  Sonic::initInstance(new EntityAuthData( $platformSocialRecord,
                      $platformAccountKeyPair,
                      $platformPersonalKeyPair));

  // From this point on, the Sonic SDK is fully initialized. In order
  // to perform requests in the context of a user, the context must
  // be set to "user":
  Sonic::setUserAuthData(
    new EntityAuthData($userSocialRecord, $userAccountKeyPair));
  Sonic::setContext(Sonic::CONTEXT_USER);

  // Perform a request to another user's profile using a GlobalID
  $globalID = '28B6TE8T9NUO202C5NZIUTNQSP88E70B8JAWH4FQ58OJOB8LIF';
  $response = (new ProfileRequestBuilder($globalID))
      ->createGETProfile()->dispatch();
  $p = ProfileObjectBuilder::buildFromJSON($response->getPayload());

  echo "Profile object: " . $p->getJSONString() . "\n\n";
} catch (\Exception $e) {}
```

Listing 4.3: Example instantiation of the Sonic SDK

4.2.3 Configuration

The Sonic SDK allows to specify a set of configuration parameters that influence certain behavior of the framework. Configuration values may be passed to the configuration object as described in Listing 4.3 in Line 26. Table 4.4 lists all configuration parameters of the Sonic SDK.

Table 4.4: GSLS Configuration Parameters

Requirement	Default	Explanation
`primaryGSLSNode`	`130.149.22.220:4002`	IP address and port of the GSLS node to resolve GlobalIDs.
`secondaryGSLSNode`	`130.149.22.227:4002`	Secondary GSLS node to be contacted in case the primary GSLS node is not available.
`apiPath`	`/sonic/`	Path from the base domain to reach a script handling Sonic protocol requests.
`timezone`	`Europe/Berlin`	Timezone of the OSN platform.
`verbose`	`0`	Verbosity of the service. `0`: log nothing, `1`: log errors, `2`: log info, `3`: log everything.
`requestTimeout`	`3`	Timeout of protocol requests in seconds.
`gslsTimeout`	`3`	Timeout of GSLS requests in seconds.
`logfile`	`sonic_log.txt`	Path to the logfile.

Setup

In order to setup the Sonic SDK for an OSN platform, Composer is used to install the SDK resources and manage all dependencies. To setup the SDK, it hence needs to be added as a requirement to the overall project's `composer.json` project file as listed in Listing 4.4.

```
1 "require": {
2   "sgoendoer/sonic": "0.5.0"
3 }
```

Listing 4.4: Setup of the Sonic SDK

The Sonic SDK has been published on Packagist[13], the package repository of Composer. To install or update the Sonic SDK from Packagist, the command `./composer update` is used. This causes Composer to install the Sonic SDK and all of its dependencies automatically in the specified location.

[13]Sonic SDK on Packagist: `https://packagist.org/packages/sgoendoer/sonic`. Accessed: 30.8.2017

4.3 Sonic OSN

In order to test, evaluate, and demonstrate Sonic, the Sonic OSN has been implemented as a reference implementation for Sonic-based OSN platforms utilizing the Sonic SDK. While being initially developed for testing purposes only, the implementation has been extended to provide an option for end users to install and use an OSN platform compliant with the specifications of the Sonic OSNF. The Sonic OSN implementation supports all functionality of the Sonic SDK and allows to interconnect with other OSN platforms in the Sonic OSNF. Users can register with a OSN platform running the Sonic OSN and create a social profile page. In order to create a network of friends or acquaintances, the Sonic OSN supports adding users to a friend roster using the Sonic protocol. Users can post activities in a stream, access other user's streams and profile pages, and comment on, like, or tag posted content. The Sonic OSN furthermore supports private communication between users via the Sonic protocol as described in Section 3.7. Figure 4.2 depicts two example screens of the web frontend of the Sonic OSN implementation.

Still, the Sonic OSN has been developed as a reference implementation to develop and test the Sonic SDK. The implementation hence did not focus on a streamlined user experience or state of the art user interface. For example for evaluation purposes, raw contents of data objects used by the Sonic architecture are displayed, allowing developers analyze sent and received data objects and messages.

4.3.1 Implementation

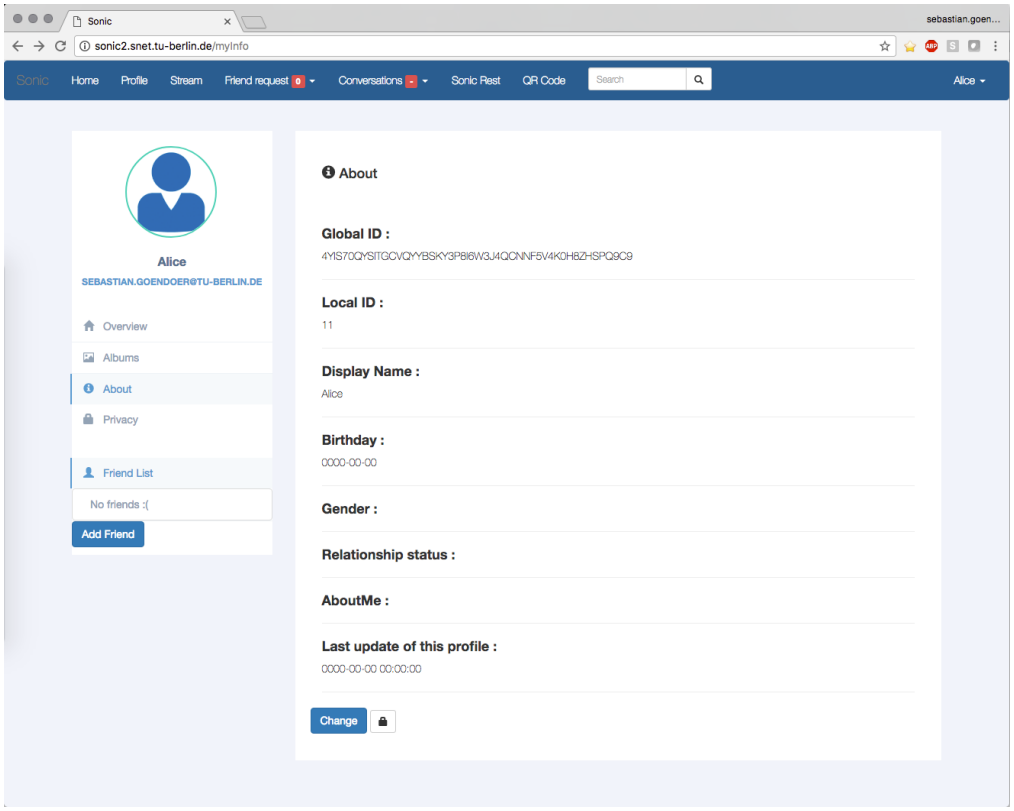
The Sonic OSN has been implemented using the PHP-based framework Laravel[14]. Laravel is a free and open source framework for building web applications and services. The framework builds on the model-view-controller (MVC) pattern [313] and utilizes the Composer package management system.

The Sonic OSN specifies a list of dependencies as listed in Table 4.5. The list includes PHP, the Sonic SDK, as well as the Laravel framework. Furthermore, the Sonic OSN requires a web server, for example a Apache HTTPd, for handling incoming requests as well as a Database Management System (DBMS), for example MySQL, for persisting data objects.

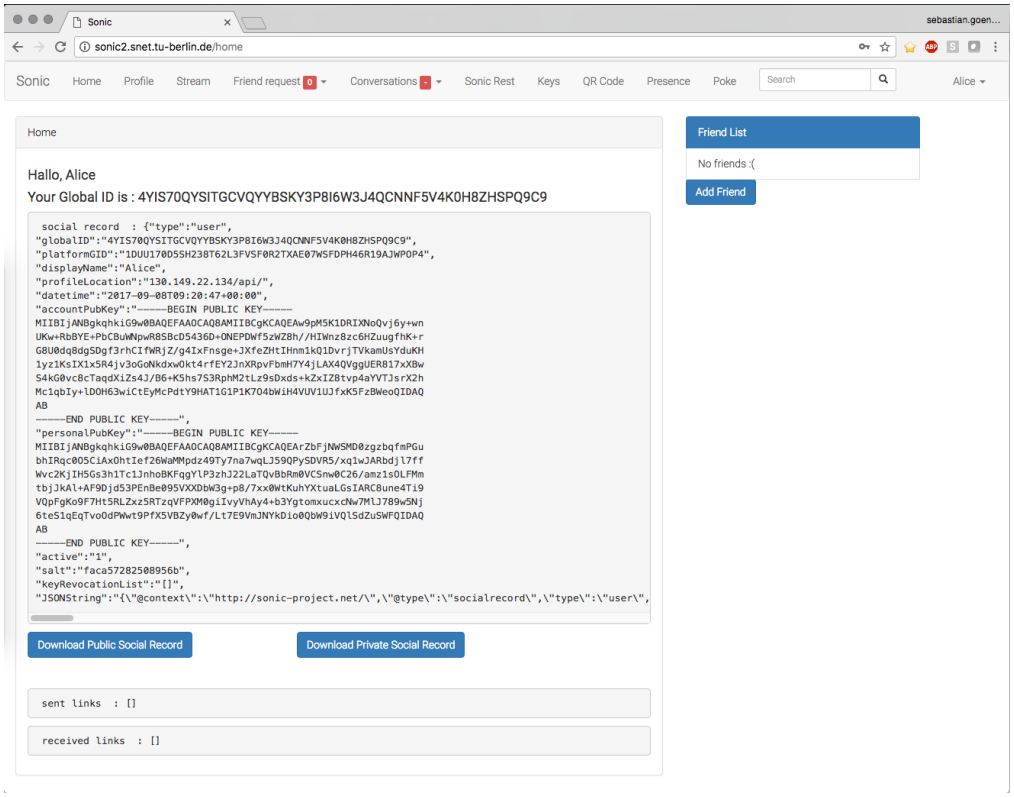
Table 4.5: Sonic OSN Requirements

Requirement	Version
PHP	5.6+ / 7.0+
Sonic SDK	0.3.0+
Laravel	5.4.0+

[14] Laravel: `https://laravel.com/`. Accessed: 12.11.2017



(a) Profile in the Sonic OSN.



(b) Display of the Social Record in the Sonic OSN.

Figure 4.2: Example screens of the Sonic OSN reference implementation.

4.3.2 SonicPi

To allow users to host a own Sonic-based OSN platforms without the need to own a administrate their own server, the project SonicPi was developed as part of a masters thesis in the Sonic project [314] and provides an installable version of the Sonic OSN implementation, which has been adopted for deployment on RhaspberryPi-computers. RhaspberryPi computers are affordable and small single-board computers[15]. SonicPi has been built using a RhaspberryPi 2 Model B, which features a ARM BCM2836 CPU based on the ARMv8-A 64/32-bit architecture and 1GB RAM. The board features several connectivity options, including USB 2.0, HDMI 1.3, 10/100 Ethernet, Bluetooth 4.1, 802.11n wireless LAN, and a bootable MicroSDHC card slot. The board, excluding external components, consumes approximately 350mA and does not require cooling and hence is an ideal option for cost-efficient hosting of small web services at home[16]. A SonicPi computer can be integrated into a user's personal computer network at home, for example by connecting it to the Internet router as depicted in Figure 4.3.

The implementation of the SonicPi features an easy-to-use installation script for setting up a home-hosted Sonic platform and connect it to the OSNF. As broadband Internet connections typically do not support static IP addresses and therefore change the IP address periodically, the SonicPi implementation provides functionality to automatically update the IP address in the GSLS [314]. This ensures availability of the SonicPi instance at all times, even when a new IP address is assigned by the network operator, for example after a (periodically forced) disconnect. The implementation of the SonicPi showed that hosting an OSN instance within the home network of each user is feasible. Still, RaspberryPi-based computers only provide a limited performance, causing requests for a user's profile and associated data to be comparably slow.

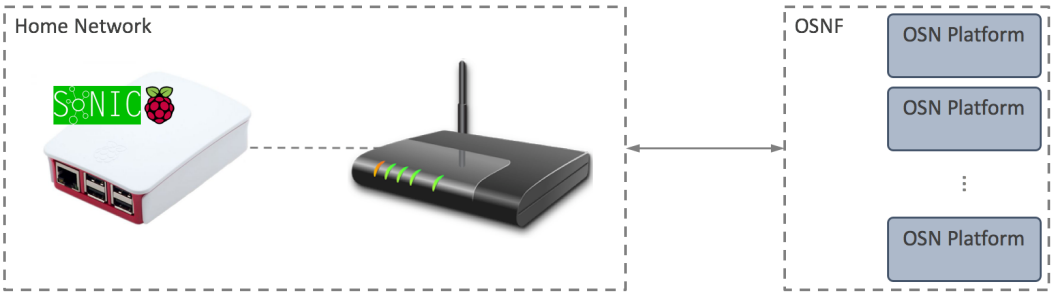


Figure 4.3: Architecture of the SonicPi implementation.

[15] RhaspberryPi documentation: `https://www.raspberrypi.org/documentation/`. Accessed: 12.11.2017
[16] RhaspberryPi FAQ: `https://www.raspberrypi.org/help/faqs/`. Accessed: 12.11.2017

5 Evaluation

To evaluate the proposed approach for an open and heterogeneous OSNF, a four-fold evaluation is performed. First, a qualitative analysis is presented, which assesses the proposed solution using requirements and challenges defined in scientific literature. In a second qualitative analysis, the proposed solution is evaluated based on the architectural significant requirements (ASR) defined in Chapter 3.3, where a comparison of the developed components with existing solutions in other OSN services is presented. Following, results of a quantitative performance analysis of the GSLS are presented. To conclude the evaluation of this thesis, a brief review of projects integrating the Sonic protocol and architecture in new and existing solutions is given.

5.1 Qualitative Assessment of Requirements and Challenges

Challenges faced by distributed OSN services and systems have been analyzed and specified in scientific literature, where requirements have been derived that should be fulfilled by DOSN services in order to successfully address and overcome the issues of closed, centralized OSN services. Fitzpatrick and Recordon defined several goals in their much-noticed article *Thoughts on the Social Graph*, including to *"make the social graph a community asset [...]"*, namely by establishing a *"[...] non-profit and open source software"* ensuring that *"[...] the design [of components] is such that others can run their own instances, sharing data with each other"* [40]. Furthermore, Applequist et al. listed four major issues to be tackled by DOSN services, being portability of OSN profiles and data, use of global identifiers, *"linkability"* of data, and data privacy [105]. Paul et al. identified a list of requirements for approaches to successfully distribute OSN services in [106]. The list comprises transparency of the distribution of the service, integration of functionality and content from all nodes in the service in a unified user interface, support of all relevant OSN features, support for connections between users on different nodes, uninterrupted availability of content and functionality, and the ability of users to protect their privacy and specify who is allowed to access what part of their profiles. In [315] and [88], Buchegger et al. present an early overview of challenges to be addressed by P2P-based DOSN services. The identified challenges specifically focus on issues in DOSN services based on P2P-systems, but lists several challenges that also apply to federated DOSN services. Challenges also applicable for federated DOSN services comprise technical issues to solve such as topology of the architecture, locality of data storage and message routing, identification and dissemination of content

and users, security and data privacy, or robustness against failures and attacks. Since user numbers in open DOSN alternatives are small compared to numbers as reported by Facebook, Koll et al. analyzed technical reasons why DOSN services were not able to attract a significant amount of users so far [101]. The survey provided a list of identified challenges that should be met by DOSN services in order to be able to succeed on the market. The list of challenges comprises independence from any centralized component or provider, general availability of the service independently of time and location, support for mobile-only operation, minimization of communication overhead caused by the distribution of the service, the ability to scale to a large number of users, resiliency against attacks and potential misuse, support for data privacy for users, and overall good performance of the service. Finally, Pathak et al. analyzed handling of privacy in OSN services and implemented mechanisms for access control [249]. From their findings, they compiled a set of recommendations for federated DOSN services regarding to privacy and access control, including support of open standards, a comprehensive and extensible data model, fine-grained access control mechanisms, identification of users, support for communication between any two entities of the DOSN service, and discovery mechanisms for content. From a more technical perspective, the W3C defined a collection of requirements for web services, including interoperability, integration with the World Wide Web, security, scalability, and extensibility [244]. A set of design principles has also been defined by the FIArch group [245] comprising heterogeneity, scalability, robustness, loose coupling, and locality. While heterogeneity as one of the major characteristics of the Internet in general [316] describes the support for different types of services and implementations, the principle of loose coupling allows individual components to remain independent from each other. Loose coupling should be supported because *"loosely coupled systems are said to have more flexibility in time constraints, sequencing, and environmental assumptions than do tightly coupled systems"* [245]. Scalability describes that services should be able to scale to millions of users, while robustness suggests that services should be able to deal with any possible kind of error [317]. Finally, locality suggests that in order to prevent communication overhead in distributed systems, data should be stored close to where it is actually processed.

The identified challenges and requirements can be grouped into thirteen categories to be addressed, being support of *core features and extensibility*, *independence* from central entities and distributed control, *data portability*, use of *global identities*, *openness* of standards and formats, *interoperability* of services and platforms, *transparency* of the federation and integration, *data privacy and access control*, the ability to create and create and maintain *relations* with other users, *availability* of data and functionality, *support for mobile* devices, *performance & scalability*, and *third-party application support*. In the remainder of this section, the Sonic approach is evaluated against the aforementioned thirteen categories.

5.1.1 Core Features and Extensibility

Functionality in the social web differs between individual OSN platforms. Researchers therefore note that novel OSN services have to support all important functionality and furthermore also allow extensibility for support of possible future features and functions. Paul et al. argue that a DOSN service must implement and provide all core functionality of today's OSN services, being *"publication, search, and retrieval of profiles and attributes"*, including *"all data related functions"* [106]. Koll et al. further note that a DOSN service has to allow integration and support for new features [101], while Pathak et al. note that an OSN's data model should not only support the description of existing social resources, but also allow extensibility in order to support new types of functionality and concepts [249]. Also, extensibility has been identified as an important criterion for web services by the W3C [244].

As part of this thesis, featuresets of today's most popular OSN services are analyzed. Based on this analysis, a taxonomy of OSN features has been derived, which is used to define a OSN core featureset. This core featureset comprises all functionality supported by a two thirds majority of today's most popular OSN services and can hence be interpreted as standard functionality of the social web of today. The Sonic protocol implements functionality for the OSN core featureset and furthermore supports the implementation of additional functionality via feature extensions. This way, different OSN platforms can implement a set of additional OSN features not included in the OSN core featureset, where the Sonic protocol provides functionality to negotiate supported feature implementations between different OSN platforms.

5.1.2 Openness

Openness is an important factor for the long-term acceptance of technologies and standards in the social web. Fitzpatrick et al. state that a solution to unify the social web into one social federation should be open source and non-profit [40], in order to allow anyone to contribute and adopt formats and protocols. Pathak et al. also argue that DOSN services should comply to open standards in order to support the integration of heterogeneous systems [249]. Openness has also been stated as a requirement by the W3C for web services in general, demanding a integrability with the World Wide Web [244]. By using open standards instead of proprietary and closed ones, web services are able to freely access relevant information from remote locations - in contrast to information being locked up via restrictive interfaces. Similarly, the FIArch group listed support for heterogeneity as well as independence of services via loose coupling as requirements for future Internet services [245].

Sonic aims to build an open and heterogeneous OSNF of which any OSN service can become a part of by implementing the required functionality and protocol. The proposed protocol and employed data formats are therefore built on existing open standards, which have been widely adopted by the web in general and by OSN services in specific. For example, data in Sonic is encoded in JSON, a platform-independent and human-readable format for encoding and

exchanging information, while status updates are encoded using the Activity Streams 2.0 standard. To ease the integration of existing OSN services into the federation, a PHP-based SDK is provided, which implements all required functionality to connect to other OSN platforms in the OSNF. Sonic intentionally avoids the usage of proprietary APIs or data formats and instead builds on existing open standards and formats. This allows both large OSN services as well as developers of smaller solutions to connect their services to each other without relying on proprietary formats or protocols. Both the Sonic architecture and protocol are published in the public domain, where reference implementations have been published on GitHub under the MIT open source license.

5.1.3 Independence & distributed control

The World Wide Web itself was designed as an open and distributed network of information, in which no single entity is able to control and inhibit information exchange. Still as of today, many services and platforms of the web are built in a centralized fashion, creating strong dependencies on specific services and the companies behind them. As pointed out in Chapter 1, the social web itself has become a landscape of centralized, isolated walled gardens, in which users are locked in. To address this issue, Fitzpatrick et al. point out that it is important for the design of an open and federated social web to be designed in a way that allows users to *"run their own instances [and] sharing data with each other"* [40]. Koll et al. further specify that a *"DOSN must not depend on any external resource provider, neither commercial nor altruistically motivated"* as a centralized resource provider might not only to *"observe communication patterns, but also [might] change the terms and conditions or shut down their service [...]"* [101], effectively impairing or entirely interrupting a DOSN service's functionality.

The architecture of the Sonic OSNF is designed in a distributed, open fashion. This way, OSN platforms that implement the Sonic protocol remain independent from each other following the principle of loose coupling. In case one OSN platform in the federation fails, user profiles and content hosted on the failed platform become unavailable for everyone in the OSNF as OSN services are operated independently from each other. Consequently, interoperability of all other OSN platforms in the OSNF is not impaired. Furthermore, the GSLS as the common identity management service is organized in a distributed fashion itself. GSLS servers join a P2P network that distributes content and control over the DHT to all participating GSLS servers, where no centralized authority is able to interrupt or censor datasets and requests.

5.1.4 Data portability

In today's social web, users are locked into the OSN service they signed up with. Missing support for interoperability of most OSN services inhibit seamless communication between users of different OSN platforms. Furthermore, implicit network effect bind users to the service they signed up with, relieving them of any option to migrate to a competitor's OSN service in case they become dissatisfied for any reason. While many OSN services already provide export

functionality for OSN profile data, the implemented data formats are not suitable for importing the data at a new location. In consequence, users can only change their OSN provider by abandoning their entire OSN profiles and creating a new one in the desired service. Following this process, all data is lost unless it is imported in a cumbersome, manual process. While few implementations support basic and mostly manual portability of OSN profiles, user identifiers are inevitably changed, resulting in broken links in the social web.

The fact that users are inevitably bound to the OSN platform they initially signed up with has been identified as one of the major issues of today's social web and web services in general. Applequist et al. note that the lack of support for data portability is one of the four major problems experienced by users of OSN services [105]. As it might be desirable for users to utilize content they created in OSN services in other OSN services as well, Graef argues that OSN services should provide functionality for a user to automatically extract his entire OSN profile from any OSN platforms and import it in any other OSN service without losing data [35]. As of 2016, the European General Data Protection Regulation (GDPR) grants customers of web services the right to export personal information, including the ability to transmit the exported information to other services with the intention to import the data at the new location for further use [109].

One of the main contributions of Sonic is the support of seamless and automated migration of OSN profiles. Following the proposed concept of OSN profile migration, data is automatically extracted from one OSN platform and imported into another one. Here, the migration functionality encapsulates all OSN profile information in a common data format that allows any OSN platform to interpret and import an OSN profile. As user identifiers are resolved via the GSLS to a OSN profiles current location, identifiers to and from a migrated OSN profile remain intact and unchanged. This way, Sonic provides an automated and seamless process for portability of data and profiles.

5.1.5 Global Identity

User identifiers in today's social web are bound to the OSN platform they signed up with and therefore have to be exchanged in case an OSN profile is migrated to a new location. As described in Chapter 3.5, user identifiers usually comprise the OSN platform's domain name. In case a profile is migrated to a new service, discovery of an OSN profile identified in this manner would therefore yield the old OSN profile's OSN service. As in consequence, the user identifier needs to be changed to comprise the new OSN service's domain name, links to the migrated OSN profile become invalid.

Applequist et al. point out that a problem exists with the inability of users to reuse their identity when signing up with a new OSN service. Specifically, users need to *"re-find their friends"* while connections to friends are lost when a profile is moved [105]. Buchegger and Datta [315] as well as Pouwelse et al. [91] also note that it is essential to discover peers independently of their physical address. Pathak et al. also note that means to securely identify communication partners are essential for a DOSN service, where an *"advanced cryptographic system"* should

provide trust and security [249]. They also note that resources need to be able to be distributed within the federation, including mechanisms for identification and discovery [249]. While focusing on P2P-base OSN services where a node's location in the network may frequently change, this also holds as a requirement for the Sonic OSNF as it allows migration of OSN profiles.

Sonic introduces a self-issued, globally unique and domain-agnostic identifier, the GlobalID, as well as an identity management service, the GSLS. GlobalIDs are self-issued by users without the assistance of a central authority and are derived from an RSA public key pair using PBKDF#2 hashes with SHA256. This way, identifiers in Sonic are not only based on strong cryptographic mechanisms, but also remain independent of the domain of a user's OSN profile location. As domain-agnostic identifiers require an alternative form of discovery mechanism, the GSLS provides a distributed and decentralized lookup of OSN profiles. Therefore in case an OSN profile is migrated to a new location, the user's identifier can remain unchanged, so that links to and from other user's OSN profiles are not interrupted.

5.1.6 Interoperability

Today's social web is a landscape of isolated islands, where users of OSN services are intentionally kept from seamlessly communicating with the users of competitors' services. This lack of interoperability has been identified as one of the major challenges not only in DOSN services, but also in web services in general [244]. In OSN services, interoperability would allow users to communicate with other users on different OSN platforms. Interoperability has been implicitly proposed by Smarr et al. in their *Bill of Rights for the Social Web* [1], and has been listed by Graef as one of the main issues of closed, centralized OSN services. Graef also claims that interoperability of OSN services is even more important than data portability alone, as *"porting personal data from one network to another does not solve the problem of loosing one's friends if one moves"* [35]. Finally, the ability of any two entities in a federation to communicate with each other regardless of the used service has also be noted as an important requirement by Pathak et al. in their assessment of privacy in a federated OSN environment [249].

Sonic addresses the issue of lacking interoperability in the social web by proposing a holistic approach for seamless communication. The approach comprises the Sonic protocol, which facilitates data exchange and communication between OSN platforms. As OSN featuresets differ between existing OSN implementations the Sonic protocol supports the OSN core featureset, which is supported by a two-thirds majority of today's most popular OSN services as described in Chapter 3.4. Furthermore, support for additional features can be implemented via the feature extension of the Sonic protocol, allowing for support of new functionality as well as evolution of existing features. This way, Sonic implements seamless interoperability of OSN services and allows users of different OSN platforms to connect to and interact with each other. This way, platform borders become transparent for users, as it is rendered irrelevant on which OSN platform a user's OSN profile is hosted.

5.1.7 Transparency and Integration

The social web has become a communication medium for people worldwide and is used by individuals regardless of nationality, gender, age, or educational background. Paul et al. therefore state that the distribution of OSNs therefore needs to be entirely transparent, as it has to address a broad range of users, including inexpert audiences [106]. Transparency mandates, that it is rendered irrelevant whether an accessed profile or data object is hosted on a local or remote OSN platform, and whether the hosting OSN platform is a different OSN service implementation than the one the profile or data is accessed from or not. To achieve this, Paul et al. argue that a single integrating user interface must be provided, in which data from all connected OSN profiles and platforms can be displayed for a user [111].

The Sonic approach provides a protocol and a set of data formats to exchange and describe information in a unified fashion between different OSN platforms. In the resulting OSNF, accessing data from another user's OSN profile will retrieve this data from the respective OSN platform hosting the profile, where data is formatted using the data formats described by Sonic. The requesting OSN platform may then extract and verify the received information and display it in its own user interface to the user. Following this approach, data from different OSN profiles hosted on different OSN platforms will be displayed in a common way, following the design and *look & feel* of the respective OSN platform's interface as depicted in Figure 5.1. Platform borders are hence eradicated, as it is rendered irrelevant for users at which OSN platform data is hosted.

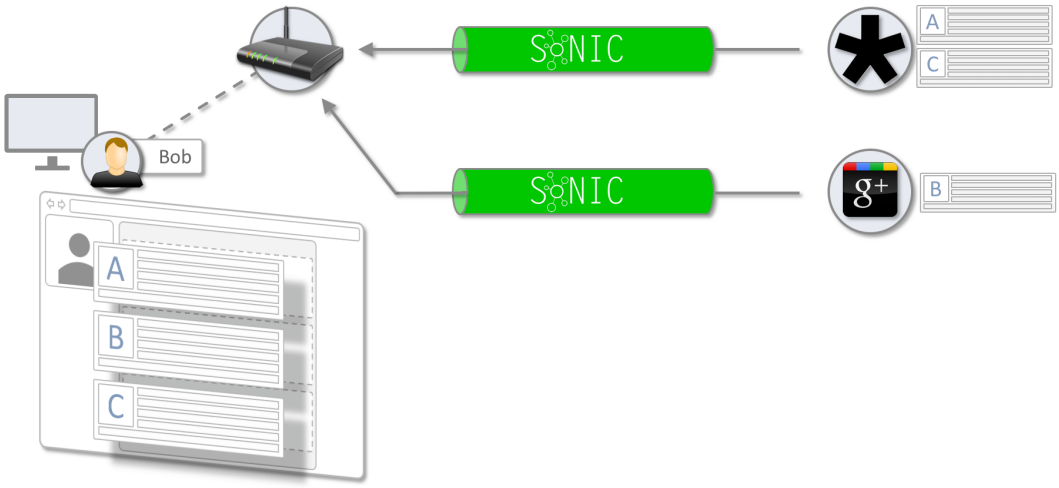


Figure 5.1: Dissemination of content in the OSNF: Content is retrieved from connected OSN platforms and displayed locally in a single integrating interface. Following this principle, platform borders become transparent to the user as it is rendered irrelevant at which OSN platform content is hosted.

5.1.8 Privacy

Protecting a user's data privacy is a major issue web services in general and DOSN services in specific face [244], where data should be stored locally within the OSN profile of its owner following the locality principle [245]. Being one of the main reasons for decentralization of OSN services, support for controlling access to one's data, for example via definition of fine-grained access control, is crucial and must be provided by all DOSN implementations [101]. Furthermore, Paul et al. [106] note that confidentiality, defined as the fact *"that information is not made available or disclosed to unauthorized individuals, entities, or processes"* [318] should be ensured and access to every attribute of OSN profiles needs to be controlled. To address privacy issues of today's centralized OSN services, Applequist et al. [105] propose a policy-based approach allowing users to define fine-grained permissions to control who is allowed to access what parts of their profiles. Similarly, Pathak et al. note that users should be able to define fine-grained access control policies to control who is allowed to access certain parts of their OSN profiles [249].

Based on the Sonic protocol, different kinds of OSN platforms are connected, rendering it irrelevant which type of OSN service is used to host a user's OSN profile. Hence, users can freely choose a OSN provider they trust, or even run their own OSN platform, for example on a virtual machine in the cloud or at home. This improves data privacy for all users, as they are able to control, which OSN provider gains access to their entire OSN profiles, effectively preventing unwanted use of profile information for targeted advertisement or similar uses. As Sonic proposes a open and heterogeneous federation of OSN services in which OSN profiles and all associated data is hosted on different OSN platforms, attackers are implicitly hindered from accessing all user's information when successfully attacking one OSN platform as *"a decentralized system without a single, central data repository [...] limits the risk of large-scale privacy breaches"* [88]. Finally, Sonic implements a role-based access control mechanism as proposed by Applequist et al. [105] and Pathak et al. [249]. By defining fine-grained access control rules, users are able to specify, which users are able to access what part of their social profile data and in what way.

5.1.9 Relations

OSN services are built on the principle of users being able to connect to each other, forming a global network of friends and acquaintances referred to as the social graph. A decentralized federation of OSN services needs to address the issue of how to create and maintain the social graph in a distributed fashion, without a central entity being able to monitor and control the process. Paul et al. note that it is imperative that relations between users need to be able to be modeled just as in traditional, centralized OSN services [106]. They further state that modeling relations is important as they are often used for *"publish-subscribe-like communication, ease of access control, and publicly announcing real world friend- and other relationships"*.

Sonic allows to define unidirectional, explicit links between individuals, where each link needs to be requested explicitly and confirmed by the targeted user before it is considered to be established. In case a link request is confirmed, a digitally signed link object is created, acting as proof of the fact that the connection has been confirmed by the user to whom the request was directed. This way, a distributed, reliable social graph can be built, in which the creation of fake friends lists is prevented.

5.1.10 Availability

OSN services in general build on the idea of building a digital representation of oneself, which then can be accessed and viewed by other users. In consequence, availability of data and functionality of an OSN service is a crucial criteria and needs to be guaranteed [106][245]. While the use of dedicated servers in federated DOSN environments provides a basis for uninterrupted availability, single-user OSN platforms hosted at home or on mobile devices might fail to provide the required availability due to changing IP addresses or interrupted network connections. Especially P2P-based mobile DOSN approaches introduce the issue of availability of data, as nodes hosting content may become unavailable and information therefore unreachable. Here, Koll et al. state that functionality and availability of a DOSN service must be ensured at all times [101].

The Sonic OSNF creates a heterogeneous ecosystem of loosely coupled, independent OSN platforms. While in this approach, individual platform servers may fail causing OSN profiles hosted on these OSN platforms to become unavailable, the overall functionality of the OSNF would remain unimpaired and fully functional. Still, availability of functionality and data provided by individual OSN platforms needs to be ensured by the respective OSN platform operator.

The GSLS as the overarching directory of identities is organized itself as a DHT-based P2P service. To provide uninterrupted availability and operation of the service, all Social Record datasets are replicated to multiple GSLS nodes in the overlay using indirect replication. In case one or some GSLS nodes fail, the DHT reorganizes itself using its built-in self-healing capabilities, while the replication strategy ensures that no data is lost.

5.1.11 Mobile support

Due to the ongoing trend towards mobile Internet access [319], Koll et al. argue that an OSN service has to be able to be run on mobile devices and remain fully functional even if all components of an OSN service are hosted solely on mobile devices [101]. Mobile devices face a series of technical limitations, such as bandwidth limitation, battery lifetime, allowed high speed traffic, availability due to insufficient network coverage, or computational power. These limitations make running and hosting a OSN platform in the Sonic OSNF a complicated task, as it requires the mobile device to be constantly reachable while running an application server and other services.

Sonic was not designed with mobile hosted services in mind and hence lacks support for addressing the aforementioned limitations, as well as resulting issues

such as availability of content or changing IP addresses of the device. Current hardware of mobile devices is anyhow capable of running a Sonic-compliant OSN platform. While this claim has been proven by the SonicPi project as described in Chapter 4.3, integrating mobile devices as OSN platforms into the OSNF would require to tackle several issues regarding to availability and connectivity.

5.1.12 Third Party Support

Buchegger and Datta note that support of third-party applications and services is of importance, as third-party applications and services add and extend the usability and attractiveness of OSN services in general [315]. The architecture of Sonic allows OSN platforms to implement any form of additional APIs and interfaces, such as Open Social, in order to provide optimal flexibility for developers. Still, Sonic does not specify any APIs or guidelines for the integration of third-party applications and services.

5.1.13 Performance & Scalability

As the social web has become one of the main communication mediums of today, OSN architectures are required not only to be designed for efficiency and performance, but also to be able to scale to several millions of individual users. Koll et al. [101] note that a DOSN service needs to provide an at least *"user-friendly performance"*, as especially P2P-based solutions often exhibit high latencies. They argue that even though a distributed OSN service will not be able to perform as good as a centralized one due to the inevitable communication overhead, *"[...] query delays should be well within 2 seconds"*. Generally, Koll et al. note that storage and communication overhead should be kept at a minimum in order to prevent the entire service to be slowed down, while at the same time the service needs to be able to scale to a high number of users.

As of today, the user base of the social web is concentrated in a small number of OSN services, which therefore each host a high number of OSN profiles. For example, Facebook as today's largest OSN service worldwide, is used by more than 2 billion users [74] while RenRen is used by 240 million users [83]. In consequence, providers of large OSN services are forced to employ large datacenters to satisfy a high number of customers with an at least reasonable performance [242]. DOSN architectures distribute both OSN profiles of users and workload over a number of nodes, thus lowering the need for highly performant servers being deployed. Still, low performance nodes might negatively influence the overall performance of a DOSN service.

The OSNF, as envisioned by Sonic, introduces a paradigm shift towards a larger number of OSN platforms each hosting a smaller number of OSN profiles. This would allow even small companies with limited resources to operate an OSN service for a smaller number of users in the OSNF. In the most extreme version of this scenario, all users would host their own OSN profile on a self-hosted server, resulting in every OSN platform maintaining exactly one OSN profile. This way, users are able to remain in full control of their OSN profiles and associated information. Individual users who demand and value a high level of privacy and

control for their OSN profile can setup and run their own private server. At the same time, users who prefer to have their OSN profile hosted for them can choose an OSN service operator that fulfills their demands and expectations.

As found by Darwish and Ghazinour [320], users can be grouped by how much they value privacy in OSN services. Users, who value data privacy and direct control of their OSN profile high enough, are able to to go through the required overhead of operating their own private server in order to run their own OSN platform in the OSNF. Still, the average user of OSN services will most likely choose an OSN service provider that offers the convenience of hosting one's OSN profile.

The current federated social web demonstrates, how users choose their DOSN platform in a federation of DOSN servers. Here, the website podupti.me maintains a list of available servers in the federated social web, including the respective number of users hosted on each server[1]. As servers have to be added to the list proactively, the list cannot be considered as comprehensive as many existing servers are probably not listed. Still, the numbers published by Podupti.me[2] allow to derive an estimation of how the number user accounts per server varies. Podupti.me lists 133 individual servers with an availability of more than 50%[3], of which 102 reported the number of hosted users profiles. In total, 42,812 users are reported with an average number of hosted user profiles of 419.7 and a median of 50 users. 18 of the listed servers report ten or less registered users, where 5 servers report more than 1,000 users each. The largest server is run by the Diaspora project and reports 16,967 individual users. Hence, even though a federated approach introduces overhead for communication and data exchange [101], the overall feasibility of federated social networks has proven itself through the evolution of federation protocols and services of the past decade. Further analysis of federated social web services has been conducted, for example by Marcon et al. [107], Koopmans [321], or Koll et al. [322].

Sonic implements distribution of content with little communication overhead as content is not replicated to other OSN platforms. Following this paradigm, content is only transferred when it is actually requested, allowing to minimize the overall overhead of storage.

Scalability has also been identified as an important criterion for web service design in general [244][245]. As identified by the FIArch group as a design principle for web services in general, a diverse, loosely coupled, and heterogeneous architecture should be provided [245]. Loose coupling reduces dependencies between individual components, which function independently from each other and communicate using open protocols. This allows to connect different implementations in a scalable, diverse federation of services.

As the performance of a specific OSN platform directly depends on multiple aspects, such as implemented functionality, quality of the implementation, or hardware specifications, the proposed approach cannot guarantee a good performance. Still, the design of the OSN platform architecture allows OSN

[1]Project Podupti.me: `https://podupti.me/`. Accessed: 26.11.2017
[2]Numbers as published by Podupti.me by the end of November 2017
[3]Uptime is tracked via Uptimerobot: `https://uptimerobot.com/`, Accessed: 26.11.2017

service operators to implement an OSN service with a good efficiency and performance.

Furthermore, while scalability of the Sonic approach with millions OSN profiles and a high number of servers couldn't be verified in practice, existing DOSN implementations in today's federated social web demonstrate that distribution of OSN services in a federated approach is working in terms of performance and scalability.

5.1.14 Discussion

For creating a foundation for an open and decentralized social web, several challenges have been identified that can be grouped into thirteen distinct categories. The challenges and requirements listed not only address technical issues that are required to be solved in order to build a viable solution for a decentralized social web, but also address the question of success factors for OSN services. While some of the identified challenges are mutually exclusive or at least hard to satisfy at the same time [101], they point the way towards an open, decentralized, and successful social web.

Sonic addresses all issues and challenges of the aforementioned thirteen categories. While all important features of the social web as of today are defined in the OSN core featureset and supported by the Sonic protocol, support for additional OSN feature implementations can be added through protocol extensions (*Core Features & Extensibility*). This way, a common way of communication and interoperability is established (*Interoperability*). The protocol builds on and adapts existing open protocols and standards such as Activity Streams 2.0, PortableContacts, or JWT, and therefore allows for easy integration and adoption by other services and implementations (*Openness*). Users in the OSNF are able to create social profile pages and define connections to each other (*Relations*), where individual users and data objects are identified via self-issued, globally unique, and domain-agnostic identifiers (*Global Identity*). The use of domain-agnostic identifiers for both users and objects allows OSN profiles to exist independently from the OSN service they were created in, and thus facilitates addressing user profiles regardless of the domain of the OSN service they are hosted on. This results in an interoperable federation of servers, where platform borders are rendered entirely transparent to users (*Transparency*). Moreover, OSN profiles being independent from OSN services facilitates support for seamless migration of entire OSN profiles between OSN platforms without the connections between OSN profiles being severed (*Data Portability*).

All components of the OSNF that is being realized by Sonic are connected in a loosely coupled fashion without any centralized entity being able control or inhibit communication or the general functionality of the federation (*Independence & Distributed Control*). The design of the Sonic architecture aims to minimize overhead for communication in the OSNF, and hence allows the federation to scale to a high number of users, while the performance of individual OSN platforms depends on the respective implementation and deployed equipment (*Performance & Scalability*), including the respective uninterrupted availability of a service (*Availability*). The proposed OSNF generally stores

content within the OSN profile of its creator, without replicating copies to all connected users. Only content objects that are associated with other content are stored in remote locations, where unauthorized editing of content is prevented via digital signatures. This allows users to keep full control of their data, including reactions of other users to their content, for example in form of textual comments or likes. Furthermore, users can define fine-grained access control rules to specify which users are allowed to access which parts of their OSN profiles (*Privacy & Access Control*). One important aspect of today's social web is the integration of third-party applications into OSN services. As described in Chapter 3.7, today's OSN services provide APIs for external services to connect to and access their platforms, while Open Social has been proposed as an open alternative for access of third-party services to OSN platforms. To allow optimal flexibility for OSN platform implementations, the Sonic architecture hence does not explicitly specify APIs or protocols for integration of third-party applications and services (*Third-party support*). Finally, while several DOSN approaches have been proposed that implement functionality and data storage to be managed on mobile devices, the architecture of the proposed OSNF envisions a federation of stable OSN platforms, each one hosting one or multiple users. While in theory, OSN platform solutions may be implemented for and deployed on small and mobile devices, this would raise issues regarding to availability and performance of both functionality and data (*Mobile Support*).

The proposed solution therefore successfully addresses nine of the thirteen categories of challenges. Two more categories, being availability and performance & scalability, are addressed partly. Two categories, being mobile support and third-party application support, are not covered. While not being directly addressed and solved by the proposed approach, the partially supported and not supported challenges can be addressed and fully solved by respective OSN platform implementations. Table 5.1 summarizes the challenges addressed by Sonic.

5.2 Comparison

The OSNF, as proposed and designed by Sonic, addresses a series of challenges and issues a solution for an open and federated social web has to tackle. While Sonic proposes a holistic solution that addresses a list of issues and challenges identified in scientific literature as described in Chapter 5.1, other solutions exist in the social web that also address some of these issues. In order to provide an overview of existing solutions for the aforementioned issues in the social web, this chapter presents a comparison of how the individual issues are addressed in other solutions for a federated social web.

5.2.1 Architecture

Centralized OSN architectures combine all control over the service as well as their users' data in one single company or organization, causing concerns regarding to data privacy and how one's personal data is handled. To address the drawbacks of

Table 5.1: List of Requirements and Challenges for DOSN services covered by Sonic.

Core Features & Extensibility: [106][249][244]	●
Openness: [40][105][249]	●
Independence & Distributed Control: [40][101]	●
Data Portability: [105][35]	●
Global Identity: [105][249]	●
Interoperability: [105][35][249][244]	●
Transparency: [106]	●
Privacy: [105][106][101][249][244][245]	●
Relations: [106]	●
Availability: [106][101][245]	◐
Mobile Support: [101]	○
Third-party Application Support: [315]	○
Performance & Scalability: [101][244][245]	◐

centralized architectures, decentralized alternatives for OSN services have been proposed. As discussed in Chapter 1, an open and heterogeneous federation of OSN services would allow to address lock-in effects in the social web and would furthermore allow users to freely choose their OSN provider. Through the distribution of all functionality as well as data storage to multiple, mutually independent OSN operators, no single organization or service operator is able to control, monitor, or inhibit functionality of the OSNF as a whole. One requirement for an open and decentralized federation of OSN services is hence the full distribution of control of functionality and data by disallowing any central point of failure in the architecture. The necessity for a decentralized architecture and control has been defined as non-functional requirements in Chapter 3.3 in Requirements R3 (Decentralized and Federated Architecture) and R4 (Distributed Control and Management), while the need for an easy integrability of the architectural components is addressed by Requirement R1 (Non-Intrusive Design).

As described in Chapter 2.1.2, DOSN services commonly employ either a federation of loosely coupled, but otherwise independent servers, a

self-organizing P2P network of client devices, or hybrid architectures combining both approaches. P2P-OSN architectures, such as Peerson or LotusNet execute OSN functionality and store data on the end users' devices, which organize themselves in a P2P-network so that no central server is required that could inhibit or control communication flows in the network. F-OSN services host functionality and data on servers, which remain mutually independent from each other. In federated service architectures, each server operator is able to control all communication to and from the respective OSN platform, and furthermore control the content of all hosted OSN profiles. Anyhow, functionality of the overall OSN federation can neither be controlled nor inhibited. Federated architectures for DOSN services have been proven to be potent and scalable foundations for OSN services in general and have been implemented in various services, such as Diaspora, Friendica, Mastodon, or Hubzilla. For identification of OSN profiles and content objects, F-OSN services implement domain-dependent identities as described in Chapter 3.5. In consequence, OSN service operators are able to inhibit availability of OSN profiles hosted on their domain. While Friendica offers a simple migration mechanism for OSN profiles that allows users to move their OSN profile to another Friendica server, the user identifier is changed in the migration process, resulting in possible broken links.

Sonic proposes a highly diverse, heterogeneous OSN ecosystem, in which OSN platforms are connected in a loosely coupled fashion based on a common protocol that holistically covers all OSN functionality as defined in the OSN core featureset. As seamless interoperability between arbitrary OSN platforms in the OSNF is supported, users are able to choose an OSN service and platform they prefer in order to maintain an OSN profile in the social web without being cut off from communicating with users of other OSN platforms. To implement independence of all OSN platforms in the OSNF, Sonic introduces a decentralized architecture, in which no single entity can control any vital part of the OSNF. As OSN platforms in the OSNF are connected in a loosely coupled fashion, each OSN platform operator is able to exert control of the respective platform and the OSN profiles stored on it, but cannot influence or inhibit functionality of the overall OSNF. Furthermore, the GSLS as the identity management system of the Sonic OSNF is in itself distributed and therefore exposes no single point of failure. At the same time, individual node failures in the GSLS are compensated by the self healing functionality of the underlying DHT. As OSN profiles as well as content in the OSNF is identified and addressed in a domain-agnostic fashion, OSN profiles and all associated data can remain functional even after the OSN platform they were created on is discontinued. The Sonic OSNF hence introduces no single point of failure, as all components are able to work independently in case any component or service ceases to function.

5.2.2 Interoperability

Interoperability in the social web enables different OSN platforms to seamlessly connect to each other in order to form an open and heterogeneous network, in which users are able to communicate and connect to each other across platform borders. To allow OSN platforms to connect to each other in a seamless and

transparent fashion, communication protocols and data formats for information exchange are required. These protocols and data formats should rely on existing and widely adopted open standards of the social web and support all standard OSN features of today's OSN services. Furthermore, as noted in Chapter 3.4, featuresets of OSN services are diverse and evolve over time as new or improved functionality is continuously added to the existing OSN implementations. A communication protocol as a solution for interoperability in the social web is therefore required to support a variety of different implementations and featuresets, and furthermore needs to be extensible in order to support the integration of new functionality. Seamless interoperability, enabled through open and extensible protocols and data formats, has therefore been defined as non-functional requirements in Chapter 3.3 as Requirements R6 (Seamless Communication and Interoperability), R7 (Open Protocols and Data Formats), and R11 (Extensibility).

As described in Chapter 2.3, a variety of open protocols exist that implement interoperability between different OSN services. Being mostly based on open and widely adopted data formats and standards such as Activity Streams or Portable Contacts, these protocols facilitate exchange of content between different OSN service implementations without the need of a user to be registered in more than one OSN service. Individual OSN platforms connect to each other in a federation of mutually independent, loosely coupled servers, where messages are routed to the OSN platform of a targeted OSN profile. While this way basic interoperability is implemented, only a very limited set of functionality is supported as analyzed in Chapter 2.3. Yet, most of the standard functionality of the OSN core featureset as defined in Chapter 3.4 is not supported. Furthermore, the existing protocols lack of an extensive documentation, which aggravates the task of implementing support for a specific protocol suite. In consequence, interoperability in the federated social web is error prone and faulty, and furthermore lacks support for most standard functionality in the social web [13]. While being published under open source licenses, the existing federation protocols do not implement support for extensibility. As a result, developers are able to implement extended functionality for a protocol. Still, as long as the proposed changes are not published as a new version of the original standard, altered versions of the same protocol may become incompatible to the original and other extended versions. Anyhow, as a large variety of OSN features exist and new functionality is added to the existing OSN implementations on a regular basis, this would result in a high number of different, incompatible versions for the same protocol.

Sonic implements seamless communication and interoperability with all OSN platforms through the Sonic protocol as described in Chapter 3. Differences between OSN platforms in the Sonic OSNF become entirely transparent to users, as access to another user's OSN profile is made possible independently from its actual location and service type. In order to facilitate compatibility and easy adoption, the protocol is based on widely adopted standards and data formats. For example, a user's stream of status updates is based on Activity Streams 2.0, while social profile pages are encoded in a format based on Portable Contacts. Finally, all data in Sonic is encoded using JSON and JSON-LD, a platform

independent and widely adopted open format. At the same time, the protocol is built in a REST-ful fashion, allowing for an easy integration in existing web-based services. This eases the task of building software and services for the OSNF. Based on the proposed protocol, Sonic supports all main features of today's popular OSN services and furthermore supports additional features though its built-in extensibility. For extensibility support, the Sonic protocol allows OSN services to implement and use individual features in collaboration with all other OSN platforms in the OSNF. All feature extensions implemented by an OSN platform are then published via the feature API, allowing other OSN platforms in the OSNF to request a list of supported extensions. This way, the compatibility for specific functionality of an OSN platform can be determined and synchronized between communicating OSN platforms. Implemented feature extensions may then be used if both communication partners support the same feature implementation with a compatible version. The proposed solution hence supports seamless communication between users of different OSN services as well as holistic service interoperability.

5.2.3 User Identification

By inhibiting users of different OSN services to seamlessly communicate with each other, users are confined within the OSN platform they signed up with. In order to connect to and communicate with users of other OSN services as well, users are forced to sign up with multiple OSN services simultaneously. Not only is the simultaneous maintenance of multiple OSN profiles cumbersome for users, but furthermore discloses personal data to multiple OSN providers at the same time as data is published in multiple OSN services simultaneously. Support for interoperability of different OSN platforms provides a basis for singular OSN profiles in the social web, allowing users to use one single OSN profile to connect to friends and acquaintances on any other OSN services. Furthermore, OSN profiles should be portable in order to allow users to migrate their entire OSN profile to a new service at any time, without connections to and from other OSN profiles being broken due to the change of location and hosting service. In Chapter 3.3, Sonic hence specifies a set of non-functional requirements for identity management of users in Requirements R2 (Platform Independent Social Personas), R9 (Singular OSN Profiles), R10 (Migration of OSN Profiles), and R8 (Global User Identification).

In contrast to centralized OSN services that lack support for interoperability with other OSN services, F-OSN alternatives based on protocols such as OStatus, Pump.io, or ZOT allow communication between different OSN services and platforms to a certain extent. By supporting interoperability, users may use a single OSN profile to connect to and communicate with users from other OSN platforms. This way, singular OSN profiles are supported, even though functionality of existing social federation protocols is limited to few core OSN features. As described in Chapters 2.3 and 3.5, most F-OSN architectures rely on email-style identifiers for users, thus ultimately binding a user identifier to its domain. While this way discovery of OSN profiles is simple and straight forward, OSN profiles cannot be migrated to other OSN platforms without the

user identifier being changed. In consequence, links to the migrated OSN profile in content or in other OSN profiles become invalid and inevitably break. Friendica as the only F-OSN service implements a manual profile migration, following which a user's identifier is changed. To minimize the amount of broken links in the network resulting from a profile migration, Friendica supports a notification mechanism that informs users of connected OSN profiles about the change of the identifier. Still this way, only links in other user's address books are updated, while links in content such as status updates, tags, or images are left unchanged.

To address the problem of being forced to register with multiple OSN services at the same time, the Sonic OSNF introduces the concept of singular OSN profiles that allow users to maintain only a single OSN profile that is not replicated to other locations. This singular OSN profile can then be used to connect to users of arbitrary other OSN services without the need to register in these other OSN services as well. Singular OSN profiles in the Sonic OSNF are enabled via a unified identity management via the GSLS, which implements a global and domain-agnostic way of identifying users and content via GlobalIDs. This allows any OSN platform to access information of a - potentially remote - user's OSN profile, while all content is protected against unauthorized changes via digital signatures. Furthermore, OSN profiles in the Sonic OSNF remain independent from any OSN service or domain and can hence be migrated to other OSN platforms at any time. As described in Chapter 3.5.4, Sonic provides migration functionality to transfer entire user accounts seamlessly from one OSN platform to another one in an automated fashion. The data transfer is supported by the Sonic protocol, while the use of domain-agnostic SonicURIs, as introduced in Chapter 3.6.4, ensures that links to migrated content are kept intact. At the same time, connections to other user profiles are kept intact through the use of globally unique and domain-agnostic GlobalIDs.

5.2.4 Data Privacy

One of the main reasons for alternative DOSN services being implemented is the lack of data privacy and control users are forced to accept in traditional centralized OSN services. As discussed in Chapter 1, centrally organized OSN services, such as Facebook or Twitter, claim ownership of all data their customers create on or upload to their OSN service platforms. The data is then used for building highly detailed user profiles, which again are utilized for purposes such as targeted advertisements. In consequence, the individual user has no control over this process and is usually not even informed about what parts of his personal data are used how. Control of one's personal data as well as access to it is hence defined in Chapter 3.3 as Requirement R5.

Decentralized approaches allow users to host their personal data on any OSN platform operator they trust, thus giving users a choice to whom to entrust their data without being cut off from the rest of the users of the social web. While choosing an existing OSN platform operator requires users to trust that their data won't be used and accessed in ways they would not agree to, it provides a convenient way of using OSN services for users with less or no technical background. Users with a high demand for privacy and security can furthermore

host their own OSN platform in order to host their own OSN profile, allowing them to exert ultimate control over the entire social profile. Still, most existing DOSN services disseminate content to all friends, creating multiple copies of the data which are then stored in remote locations at which the originating author has no direct control.

In order to allow users to remain in control of their data, Sonic introduces a content model following which content is always stored at the OSN platform of the author with no replicas being created and distributed to other connected OSN platforms per default. Only remote content, such as comments on another user's content, are stored under certain conditions within the OSN profile of the respective other user, who then assumes full ownership of such a content object. Sonic furthermore defines a fine-grained access control model, allowing users to specify detailed access control policies and therefore fully control, who is allowed to access what parts of their OSN profile.

5.2.5 Discussion

While several aspects that are able to decentralize control and data in the social web have been successfully been solved and implemented in existing DOSN approaches, several issues of the closed and proprietary social web haven't yet been solved by existing OSN implementations. While existing DOSN implementations solve issues stemming from centralized architectures by distributing both functionality and data across multiple servers or nodes, seamless interoperability between different OSN platforms is mostly not supported at all. Protocol suites such as DFRN, OStatus, or Pump.io address the issue of lacking interoperability, but only cover a very limited set of functionality of today's social web. Furthermore, existing approaches are not extensible and therefore cannot be dynamically adapted to changing featuresets of the social web. The proposed solution implements a holistic protocol, which not only supports the standard functionality of today's social web, but also allows OSN platforms to dynamically extend their implemented functionality via feature extensions.

Furthermore, today's OSN profiles and user identities are inevitably bound to the OSN service and platform they are created in and therefore cannot be easily migrated to a new OSN platform. While Friendica offers a very limited implementation of portability for OSN profiles, Hubzilla implements a concept following which content and user account information is replicated to multiple nodes, so that user accounts can be controlled from multiple OSN platform servers. Still, no holistic concept for data portability exists for OSN profile migration. Sonic implements a holistic profile migrations mechanism, which allows to migrate user accounts between OSN services without loss of information of connections.

5.3 Performance Evaluation of the GSLS

As requests for other OSN profiles rely on the availability of the associated Social Record dataset in the GSLS, the OSNF is one of the most crucial components of the OSNF. The GSLS is therefore designed in a decentralized manner with no central entity or organization able to control its overall availability and functioning. Any user or organization involved in the OSNF is able to run and maintain a GSLS server, which is automatically included in the DHT based on its self-organizing architecture. This way, service availability of the GSLS is guaranteed by all participants of the OSNF, while no single entity is able to control the service or the managed datasets.

As all requests in the OSNF are being routed with help from the GSLS, its performance and availability is of key importance. DHTs are designed for use with very high numbers of users running individual nodes with a reasonable performance. For example, analysis of a Kademlia-based DHT showed that DHTs with approximately one million individual nodes provide a robust operation based on data replication and redundant routing, providing an overall tradeoff between availability, overhead, and performance [230].

For a quantitative evaluation of the GSLS, a network of three separate virtual servers was set up. The GSLS was installed on each of the servers and configured to connect to the other servers. Each virtual machine featured 1 CPU, 1 GB of RAM, and 10 GB of disk space running Debian Linux "Jessie" with a `3.16.0` kernel. The GSLS daemon itself was running as a dockerized service using Docker `17.09.0-ce`. The conducted test evaluated the performance of the GSLS under high load. To simulate a high load situation, Apache JMeter[4] was used to simulate multiple clients accessing the service simultaneously using 100 threads. The machine used for sending the requests to the GSLS was a late-2011 Apple MacBook Pro, featuring an Intel i7-2620M dual code processor at 2.70 GHz and 12GB of RAM running MacOS X.

As creating Social Record datasets and the associated key pairs is the computationally most demanding part of registering an identity in the GSLS, 5000 individual Social Record datasets were created in a first offline phase and stored in a comma-separated-value (CSV) file. Each stored record comprised the respective record's number, the GlobalID, and the Social Record dataset encoded as a signed JWT. Using this file, the computed Social Records could be read from the file and directly sent to the GSLS without any need for further encoding.

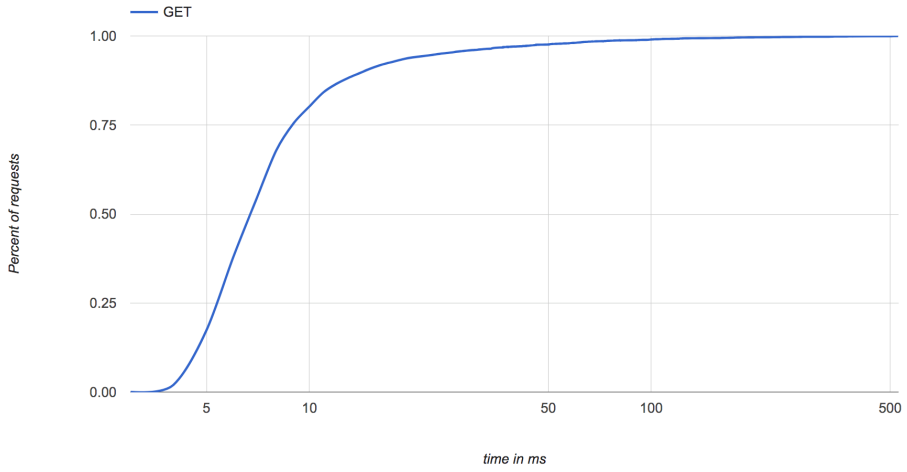
In the second online phase, the Social Record datasets were read from the CSV file and send via HTTP POST to the configured GSLS server. To simulate simultaneous requests, JMeter was configured to perform 10 requests per second until all 5000 datasets were sent to the GSLS via HTTP POST. Following the writing phase of Social Record datasets, a second phase was performed, in which all Social Record datasets were read from the GSLS.

In the given scenario, when resolving a GlobalID to the associated Social Record dataset via HTTP GET, the GSLS service performed well with an average response time of 11.24ms and a median response time of 7ms. While the fastest

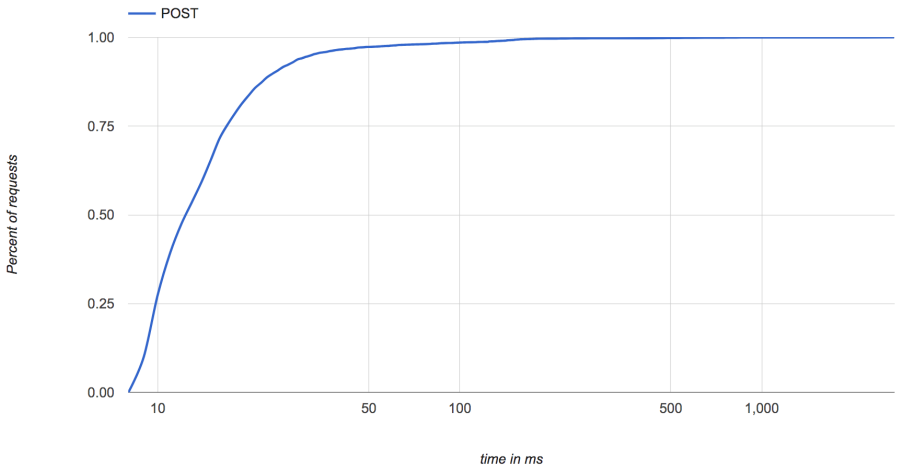
[4] Apache JMeter: `http://jmeter.apache.org/`. Accessed: 15.11.2017

response was completed after 3ms, few requests took several seconds to complete with a maximum response time of 528ms. When registering identities by sending new Social Record datasets via HTTP POST to the GSLS, an average response time of 19.24ms and a median of 13ms was measured. The increased duration when writing Social Record datasets is thought to be caused by the internal overhead of writing data to the DHT. Here, the slowest response observed took 2746ms to complete, the fastest one 8ms.

While the performance of the GSLS showed very fast response times, this could be caused by the fact that all GSLS nodes as well as the machine running the performance test were located in the same network. In scenarios where the individual nodes of the DHT are distributed over multiple networks, routing paths will be longer and most likely result in worse service response times. The results reflect the findings of Ramasubramanian and Sirer presented in [232], which found their DHT-based DNS service to perform well with a mean of 106ms for resolving queries.



(a) CDF of performance measurements for read operations of the GSLS.



(b) CDF of performance measurements for write operations of the GSLS.

Figure 5.2: Cumulative Distribution Function (CDF) of performance measurements of the GSLS.

5.4 Integration

The concepts and components proposed in this work have been utilized and integrated in a list of software development and research projects. This section provides an overview of the integration activities that used or adopted concepts of this thesis.

5.4.1 Friendica

In order to demonstrate and test the applicability of the OSNF, the architectural components of Sonic have been integrated into the open source OSN Friendica by Hebbo [323] as part of the research project Sonic (see Appendix A). The integration replaced the formerly used protocol suite of Friendica as described in Chapter 2.3 entirely, allowing an adapted Friendica server to seamlessly connect to any other OSN platform in the OSNF. The functionality of the architectural components of the Sonic architecture could be added with little overhead. As Friendica implements a complex and not documented database structure tailored to its specific functionality, the database table structure needed to be replaced in larger parts for the integration. As Friendica stores multiple, separately encrypted versions of content for all connected users, the redesign of the table structure resulted in a more efficient and simple database table structure. Finally, the GUI of the original Friendica OSN was left unchanged apart from minor details such as displaying Sonic-compliant user identifiers.

In direct comparison with the original Friendica protocol suite, messages exchanged between Sonic-compliant Friendica platforms showed 20% less overhead with a slightly increased computational overhead due to Sonic's digital signatures for both content and messages. Summarizing, Hebbo notes that implementing the Sonic protocol into Friendica makes the *"development process [of connecting OSN platforms] easier and simpler"*, as multiple fragmented protocols are replaced by a single, well-defined one [323]. The resulting Sonic-compliant Friendica implementation is able to interact with the Sonic OSN reference implementation, while the solution allows the Friendica platform to maintain its look and feel as well as its overall user experience. Still, by integrating the Sonic architecture into Friendica, a *"better usability and [a] more convenient user experience"* is achieved through a more transparent way of handling requests and visualizing content without redirecting the user agent.

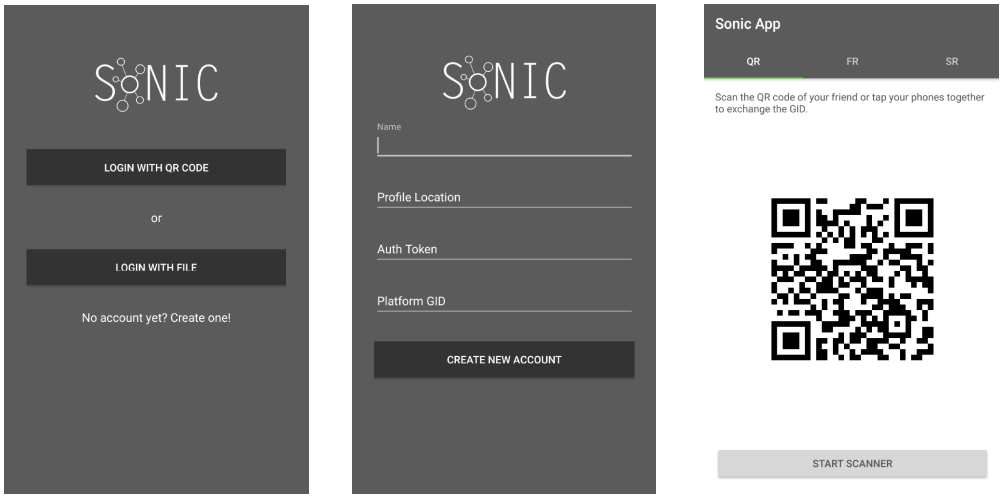
5.4.2 Feature Extensions

Apart from the integration of Sonic into the Friendica OSN, Beckmann developed feature extensions for the Sonic protocol to demonstrate the feasibility of the approach. The implemented functionality comprised a presence notification feature as well as a poke feature. The presence notification extension allows users to set a status in their OSN profile, such as *away* or *available for chat*, which can be retrieved by other users. The poke feature extension implements the functionality of Facebook's poke feature as described in Chapter 3.4. Both OSN features were implemented and integrated in the Sonic OSN. Summarizing his

experiences with the Sonic SDK, he noted that he found the Sonic SDK to be an *"easy form for developers who want to include the [Sonic] protocol in their OSNs"* [324]. Furthermore, he highlighted that the extendability of the core featureset via the feature extension protocol would provide a viable advantage of the architecture.

5.4.3 Sonic App

As management of the RSA key pairs associated with one's Social Record dataset can be a cumbersome task that might overwhelm normal users of OSN services Sonic App, a mobile application for the management of the PersonalKeyPair and AccountKeyPair as described in Chapter 3.5, has been implemented. The Sonic App is based on Google Android and allows users to create or edit a Social Record dataset and associated cryptographic key pairs. The application implements the GSLS API and is therefore able to synchronize the managed Social Record dataset directly with the GSLS. Furthermore, the application is capable to synchronize the created key pairs directly with a OSN platform via a special API, which has been integrated into the Sonic OSN. This way, management of a user's identity is made simple and convenient. Furthermore, the app allows users to create a QR code comprising their GlobalID, which can be scanned by other users. Scanning another user's QR code hence allows to verify one's identity or add someone to one's friend list. Examples of the Sonic App user interface are depicted in Figure 5.3.



(a) Main Screen. (b) Editing a Social Record Dataset. (c) Scanning a QR code.

Figure 5.3: User interface of the Sonic App.

5.4.4 ReThink

The concept and architecture of the GSLS was adopted by the EU-funded project ReThink (see Appendix A). ReThink proposes a novel architecture for decentralized web-based communication facilitating dynamic trusted relationships among distributed applications [8][11]. Based on the ReThink architecture, service providers are able to build solutions for a novel distributed Internet, in which various communication platforms are able to communicate with each other using a set of automatically negotiated protocols. As one of the core requirements of ReThink, independence of a user's identity from any service provider's domain could be realized by applying the concept of Sonic's globally unique, domain-agnostic GlobalIDs. ReThink implements the concept of the GSLS in its Identity Mapping and Discovery Service (IMaDS) [10], in which users are able to manage self-asserted identities. The IMaDS comprises a discovery service that implements functionalities of a contextual search engine, as well as registry services that manage the identity records of the users of the ReThink architecture. Identity information is then stored in a distributed directory services that maps the globally unique identities to user accounts in the various service domains.

6 Conclusion

OSN services by long have become an integral part of our everyday lives. The rise of the social web changed how we use and see the web, introducing a shift from static to dynamic and from separated to connected. As of today, we use OSN services to express ourselves, create and collect content such as images or videos, share content and information with our friends and colleagues, exchange messages, plan and organize events, or keep track of what's happening in the world. Yet, despite social communication being intrinsically a distributed, decentralized phenomenon, most OSN services are built in a central, monolithic fashion, following which all knowledge and power is concentrated in one company or organization. When people started to realize that their data was being used by OSN providers for monetary gain, discontent started to spread. Users of the social web started to feel apprehensive about the fact that their personal and in parts very private data was being used for targeted advertisement while at the same time they were implicitly forced to forfeit their rights of ownership and control to OSN providers. Yet despite users continuously voicing concerns regarding these practices, the design of today's closed and proprietary OSN services still disallows users from freely communicating with other services, while implicit network effects are used to create walled gardens, in which users are locked-in and are unable to migrate to alternative solutions without losing their data.

To address the prevalent situation of the social web being run by few organizations in a restrictive manner, alternative OSN architectures were proposed that distribute control and data to multiple independent servers. Decentralization as an architectural concept promised users the ability to remain in control of any data they collected in their social profiles. Still, the implicit network effects that exist in large OSN services keep preventing users from migrating to alternative solutions in significant numbers, as abandoning one OSN service for another one would inevitably result in losing one's friend list, profile page, and created data. Trying to avoid losing all this information and content they accumulated over years of usage, the majority of users of OSN services are reluctant to consider alternative solutions. Ultimately, today's OSN market is dominated by one single service, which was able to attract a significant amount of users while a large number of competing services and alternative solutions exist that combine a comparably rather small number of users. The social web, which once was diverse and heterogeneous, has become a monoculture of few dominating OSN services.

Two main aspects have been identified that contribute to the current situation of few OSN services heavily dominating the entire market [35]. First, the lack of data portability prevents users from moving existing OSN profiles to alternative

solutions that would better fit their demands and expectations. Without functionality to export one's OSN profile data and re-import it at another arbitrary OSN platform with the ability to keep using it at the new location, users would have to abandon their OSN profiles and enclosed information altogether and create a new and empty OSN profile in a new OSN service. Moreover, the situation wouldn't be resolved at all as they would simply be locked in again in just another OSN service. Besides the lack of support for data portability, the lack of interoperability prevents users of different OSN services from seamlessly communicating with each other. The lack of support for interoperability creates isolated islands of OSN services, to which users and data are confined. While data portability alone would simply facilitate users to move their OSN profiles to a new service without losing data, they would still be left isolated from all users who refused to migrate with them without any means to communicate with them. Therefore, interoperability represents an even more important aspect of an open and decentralized social web, as it would allow users to connect to each other and furthermore seamlessly communicate.

6.1 Summary of Results

In this thesis, the concept, design, and implementation of Sonic, an architecture for an open and heterogeneous federation of OSN services, is presented. Sonic is built on the idea of utilizing and combining existing OSN microstandards to provide a holistic framework for OSN interconnectivity. The overarching vision of Sonic is an open, seamlessly interconnected, and heterogeneous ecosystem of OSN platforms, in which users are not restricted to only communicating with users of the same OSN platform, but can seamlessly interact and collaborate with users of arbitrary other OSN services. Following this approach, borders between individual OSN platforms become transparent to users and are therefore rendered irrelevant. In consequence, lock-in effects of today's social web, which are keeping users from abandoning an OSN service they are dissatisfied with, are eradicated as user profiles may be freely migrated from one OSN platform to another at any time without losing established relationships in the social graph. This allows users to freely choose an OSN platform of their liking instead of being limited in their choice to the platform used by the majority of one's friends as a decision for another OSN service essentially would result in being cut off from all users of other OSN services. This vision of an Online Social Network Federation (OSNF) is built on a holistic approach comprising an open and extensible social API as well as data formats.

The main contribution of this thesis is four-fold. First, an extensive analysis of supported functionality in today's most popular OSN services is presented in Chapter 3.4. Based on the results of this analysis, a taxonomy of OSN functionality is introduced that allows to map different, yet equivalent feature implementations to each other. Furthermore, the core OSN featureset is defined, which comprises the most popular functionality of today's social web and therefore represents today's "default OSN functionality". Designers of novel OSN services can use this taxonomy to build OSN services that support the

standard functionality of today's social web. Second, an architecture model for OSN services is presented in Chapter 3.6. The Sonic architecture introduces components for OSN platforms to seamlessly connect to each other in the OSNF. The architecture is designed to be easily integrable, thus allowing existing OSN service implementations to retain their look and feel and overall user experience. Third, as user identifiers in the social web are traditionally domain-bound, Sonic introduces a novel concept for domain-agnostic and globally unique identifiers in Chapter 3.5. These GlobalIDs are derived from a user's RSA key pair and are registered in a distributed directory service based on DHT technology, the GSLS. This allows users to register identities without the need of a central entity controlling the process. As GlobalIDs are domain-agnostic, OSN profiles and all associated data may be migrated between OSN platforms in the OSNF without links between user accounts being broken in the process. Finally, based on the OSN core featureset, the Sonic protocol is presented in Chapter 3.7. The Sonic protocol is based on REST and adopts existing open standards of the social web. This way, seamless interoperability between OSN platforms in the OSNF is implemented. To further allow OSN services to implement and support additional features, the Sonic protocol supports feature extensions.

To support integration of both architecture and protocol into new and existing implementations, the PHP-based Sonic SDK is introduced in Chapter 4. The Sonic SDK automates the handling of communication in the OSNF, formatting data, as well as communication with the GSLS. As a proof of concept, the Sonic OSN has been developed, which demonstrates the applicability of the approach. Both the Sonic SDK and the GSLS have been published under open source licenses. To furthermore evaluate the feasibility integrating the architectural components and the protocol into existing OSN implementations, separate projects assessing the integration of Sonic were conducted. In a first project, the proposed architecture and protocol has been integrated into the DOSN service Friendica, demonstrating the feasibility of integration into existing implementations. In a second project, the proposed architecture and protocol has been applied to a RhaspberryPi implementation that allows self-hosting of a fully functional Sonic compliant OSN platform at home. Finally, to ease handling one's RSA key pairs and Social Record datasets, an Android-based mobile application has been implemented that allows users to create and edit identity information in the GSLS.

6.2 Addressed Research Questions

Sonic addresses a set of research questions as defined in Chapter 1.6, comprising an enabler for OSN diversity, mapping of OSN features and featuresets, seamless interoperability between OSN services, data portability for OSN profiles, and domain-agnostic identification of users and content in a distributed social web.

- **Enabler for OSN Diversity** OSN markets are traditionally of a concentrated nature, as network effects make it hard for new and smaller services to prevail on the market [35]. This ultimately limits the number of viable

companies in the market, resulting in a homogeneous landscape of OSN services. Today, few large OSN services dominate the entire market, leaving little room for smaller competitors. Research Question RQ1 hence addresses ways how to re-introduce heterogeneity and diversity into the OSN market, allowing small and new OSN services to compete with other companies without being eclipsed by the network effects of large services. With the vision of an open and heterogeneous OSNF, Sonic introduces a platform for a diverse and open social web.

- **Mapping OSN Featuresets** To allow different OSN featuresets of the various existing OSN services to be mapped to one another as stated in Research Question RQ2, OSN features of today's most popular OSN services are analyzed in Chapter 3.4. The analysis shows that while OSN services differ in user experience, design, and functionality, certain recurring OSN features can be identified. While individual implementations of OSN features differ between OSN services, synonymous OSN features can be mapped to one another. Based on the analysis of today's twelve most popular OSN services, a taxonomy of OSN features is derived, where the OSN features supported by the majority of the surveyed OSN services define the Sonic core featureset, comprising the OSN features profile, link, conversation, stream & activities, image, like, comment, and tag.

- **Enabling OSN Interoperability** As stated by Graef in [35], interoperability of different OSN services is needed in order to enable competition in the OSN market, which currently is dominated by few overly powerful OSN services. Research Question RQ3 therefore targets viable solutions for implementing interoperability protocols for OSN services. Sonic proposes an open and platform independent protocol and furthermore a REST-ful API that enables OSN platforms to communicate with each other and exchange information. In order to guarantee compatibility with existing standards, Sonic specifies data formats for the exchange of content objects based on JSON and other open standards.

- **Enabling Data Portability** The European General Data Protection Regulation (GDPR) mandates that users of web services must be able to migrate existing user accounts and associated information to another service with the intent to use it at the target location [109]. Data portability in the social web would allow users to abandon an OSN service in case one becomes dissatisfied and in consequence would force OSN services to compete with each other. Given the option to move to a competitor's OSN service at any time, OSN services would need to attract users by providing a better service quality or user experience than their competitors. As this would contribute to more innovation and competition in the OSN market [35], Research Question RQ4 hence addresses viable ways of migrating OSN profiles between OSN platforms without existing connections to the migrated profile to break. Sonic proposes a migration API, which allows users to export their OSN profiles along with all comprised information from their current OSN service and re-import it at the desired target OSN service.

- **Identification** As user profiles in OSN services are usually identified by local, domain-bound identifiers, migrating an OSN profile to another OSN service and therefore domain will inevitably break connections to the migrated OSN profile. Research Question RQ5 therefore addresses identification of users and data objects in the Sonic OSNF independently from an OSN service's domain. Sonic introduces a globally unique and domain agnostic identifier, the GlobalID. GlobalIDs are self-issued by the users in the Sonic OSNF and are derived directly from cryptographic key pairs. As GlobalIDs are domain-agnostic by design, Sonic introduces the Global Social Lookup System (GSLS), a decentralized directory service built on DHT technology, for resolving identifiers to the associated OSN profile locations.

As of today, implicit network effects draw users to the OSN services with the most users. Once registered, users are caught and bound by lock-in effects and the resulting threat to lose all data when abandoning the service. The solution proposed by Sonic allows arbitrary OSN services to connect to each other in a seamless fashion. In the resulting heterogeneous and diverse federation of OSN services, users are able to connect to OSN profiles of friends, relatives, and acquaintances regardless of what OSN service in the OSNF they use. Sonic introduces a non-intrusive design that can easily be integrated into existing OSN implementations with little overhead or impact on the existing user experience and business logic. The architecture facilitates OSN service federation using the Sonic protocol and data formats, allowing arbitrary OSN platforms to seamlessly communicate and exchange data, hence implementing interoperability in the social web. As OSN functionality continuously changes and evolves, Sonic implements extendability in order to support future functionality and use cases of the social web. As Sonic implements migration of user profiles, data portability as mandated by the GDPR is supported, where global and domain-agnostic user identification is provided. This way, Sonic promotes platform independent social personas which can be migrated between different OSN platforms on demand. Sonic supports the use of singular OSN profiles, which allows users to access the social web and collaborate and interact with users of arbitrary OSN services without the need for maintaining multiple OSN profiles in order to stay in touch with users of different OSN services. Finally, Sonic allows users to keep end exert full control over their personal data and information, as OSN profiles can be hosted at trusted locations and can be moved in case an OSN platform is not trusted anymore.

6.3 Future Work

One of the main challenges the distributed social web faces today is the lacking motivation of users to abandon closed OSN services and migrate to new, open alternatives. Reasons for this are not only the strong network effects that draw users to the OSN services with the largest user base, but also aspects such as the fear of losing one's carefully created and collected data due to the OSN lock-in effect, being cut off from the majority of the users of the social web in smaller,

alternative DOSN services, or the fact that alternative services such as Diaspora or Friendica again constrain their users in their own ecosystems, in which users are again mostly unable to freely communicate with other services in the social web. As long as users don't have a user-friendly, attractive alternative for the predominant centralized OSN services, willingness to change to alternative solutions is likely to remain low for most users.

With Sonic, a solution is proposed that facilitates to eradicate lock-in effects, allowing users to move away from closed, centralized solutions without losing their data. Still, today's largest OSN services such as Facebook or RenRen are most likely not willing to open up to a federated, open social web and allow their users to freely choose whether to migrate to a competitor's solution or not. As long as the threat of losing one's OSN profile and carefully curated data exists, the majority of users of OSN services will most likely be reluctant to abandon their profiles in favor of an open, decentralized alternative.

As found by Krasnova et al., the most important criteria for a user's choice for an OSN service are price and network popularity [56]. Yet, as most OSN services allow users to sign up and use the service without a monetary fee, the remaining main criterion for choosing a network is its popularity. DOSN services today represent comparably isolated solutions themselves and thus do not provide enough motivation for users to abandon their existing OSN profiles in closed OSN services.

Sonic presents a holistic solution that allows small OSN services to connect to each other seamlessly, thus connecting the small and in itself isolated communities they represent in the process. Given the assumption that a larger number of alternative OSN platforms adopts Sonic as a standard for OSN interoperability, users in the resulting OSNF would be able to use the network of OSN services as a free and open social web. The resulting OSNF would feature OSN services targeting different needs and demands, thus offering every user a platform he or she might find more suitable than the closed alternatives in existence. Ultimately, this will attract more and more users to join the decentralized, federated social web, fostering heterogeneity and openness. Once a large enough number of users and OSN services start using services in the ONSF, it will eventually become self-sustainable as analyzed by Westland [58] and establish itself as a de facto standard. Ultimately, the large centralized OSN services that dominate the market as of today will be coerced to join to OSNF themselves to avoid being cut off from the majority of users. Therefore, the main objective for the future of the social web is to implement and support Sonic as a common, open standard for interoperability in a large number of OSN services. This way, the OSNF will eventually be able to attract a significant number of users.

A Projects

A.1 Sonic

Sonic (SOcial Network Interconnect) is a research project that aimed at developing an innovative concept for online social networks (OSN) that addresses problems of current OSN platforms. Sonic mainly focused on novel concepts for data privacy, data portability, and interoperability issues. As part of Sonic, innovative protocols and algorithms were designed and implemented, which allow seamless communication and data exchange between different brands and types of OSN platforms. The developed solution facilitates a seamless integration of different OSN platforms and allows users of these services to change operator and platform at any time without losing any data.

Sonic has been chosen by the Federal Ministry of Education and Research (BMBF) as one of the IT projects in 2014 to be funded within the *Software Campus* initiative. *Software Campus*[1] integrates leading research and practical experience in a new type of concept in order to create a new generation of top managers and entrepreneurs with an excellent IT background.

- Project title: SONIC (SOcial Network Interconnect)
- Project coordinator: Sebastian Göndör
- Project partners:
  - Telekom Innovation Laboratories (T-Labs), Germany
  - Technische Universität Berlin (TUB), Germany
- Project duration: 03/2014 – 07/2016
- Programme: SoftwareCampus (BMBF / EIT-ICT)
- Förderkennzeichen: 01IS12056
- Project website: `http://sonic-project.net`

[1] Software Campus website: `http://www.softwarecampus.de/`. Accessed: 19.7.2017

A.2 ReThink

The EU-funded research initiative ReThink addresses the status quo of web based services being built in a closed fashion. Similar to the situation with OSN services, a large number of web services is designed and operated in a closed, proprietary fashion. ReThink proposes a design and prototype for a new, web-centric P2P service architecture enabling dynamic trusted relationships among distributed applications called *Hyperlinked Entities* (Hyperties) that supports use-cases beyond traditional telephony such as contextual communication, social communication, or content oriented services, facilitating interoperability, data portability, and a globally unique and domain-independent identity management for arbitrary web services.

- Project title: ReThink
- Project coordinator: Anastasius Gravas (Eurescom)
- Project partners:
  - Eurescom GmbH, Germany
  - Orange SA, France
  - Deutsche Telekom AG, Germany
  - Portugal Telecom Inovação e Sistemas SA (TPInS), Portugal
  - Fraunhofer Gesellschaft zur Förderung der angewandten Forschung e.V. (Germany)
  - Apizee, France
  - Institut Mines-Telecom (IMT), France
  - Instituto de Engenhariade Sistemas e Computadores, Investigação e Desenvovlimento em Lisboa (INESC-ID), Portugal
  - Quobis Networks SL (QUOBIS), Spain
  - Technische Universität Berlin (TUB), Germany
- Project duration: 01/2015 – 06/2017
- Programme: EU-H2020
- Grant Agreement Number: 645342
- Project website: `http://rethink-project.eu`

Bibliography

[1] J. Smarr, M. Canter, R. Scoble, and M. Arrington. *A Bill of Rights for Users of the Social Web*. `http://www.opensocialweb.org/2007/09/05/bill-of-rights`. Accessed: 14.7.2017 via `https://web.archive.org/web/20080314063658/www.opensocialweb.org/2007/09/05/bill-of-rights`. 2007.

[2] S. Göndör and J. Devendraraj. "C2M: Open and Decentralized Cloud Contact Management". In: *Procedia Computer Science* 18 (2013), pp. 2076–2085. issn: 1877-0509. doi: `https://doi.org/10.1016/j.procs.2013.05.377`.

[3] S. Göndör and H. Hebbo. "SONIC: Towards Seamless Interaction in Heterogeneous Distributed OSN Ecosystems". In: *Proceedings of the 10th International IEEE Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE. 2014, pp. 407–412.

[4] S. Göndör, F. Beierle, E. Küçükbayraktar, H. Hebbo, S. Sharhan, and A. Küpper. "Towards Migration of User Profiles in the SONIC Online Social Network Federation". In: *Proceedings of the 10th International Multi-Conference on Computing in the Global Information Technology (ICCGI)*. IARIA. 2015.

[5] S. Göndör, F. Beierle, S. Sharhan, H. Hebbo, E. Küçükbayraktar, and A. Küpper. "SONIC: Bridging the Gap between Different Online Social Network Platforms". In: *Proceedings of the 8th International IEEE Conference on Social Computing and Networking (SocialCom)*. IEEE. 2015.

[6] F. Beierle, K. Grunert, S. Göndör, and V. Schlüter. "Towards Psychometrics-based Friend Recommendations in Social Networking Services". In: *arXiv preprint arXiv:1705.10512* (2017).

[7] S. Göndör, F. Beierle, S. Sharhan, and A. Küpper. "Distributed and Domain-Independent Identity Management for User Profiles in the SONIC Online Social Network Federation". In: *Proceedings of the International Conference on Computational Social Networks*. Springer. 2016, pp. 226–238.

[8] I. T. Javed, R. Copeland, N. Crespi, F. Beierle, S. Göndör, A. Küpper, M. Emmelmann, A. A. Corici, K. Corre, J.-M. Crom, F. Oberle, I. Friese, A. Caldeira, G. Dias, R. Chaves, and N. Santos. "Global Identity and Reachability Framework for Interoperable P2P Communication Services". In: *2016 Conference on Innovations in Clouds, Internet and Networks (ICIN)*. 2016, pp. 59–66.

[9] F. Beierle, S. Göndör, and A. Küpper. "Towards a Three-tiered Social Graph in Decentralized Online Social Networks". In: *Proceedings of the 7th International Workshop on Hot Topics in Planet-scale Mobile Computing and Online Social Networking*. ACM. 2015, pp. 1–6.

[10] I. Friese, R. Copeland, S. Göndör, F. Beierle, A. Küpper, R. L. Pereira, and J. Crom. "Cross-Domain Discovery of Communication Peers. Identity Mapping and Discovery Services (IMaDS)". In: *2017 European Conference on Networks and Communications (EuCNC)*. 2017.

[11] I. T. Javed, R. Copeland, N. Crespi, M. Emmelmann, A. Corici, A. Bouabdallah, T. Zhang, S. El Jaouhari, F. Beierle, S. Göndör, A. Küpper, K. Corre, J.-M. Crom, F. Oberle, I. Friese, A. Caldeira, G. Dias, N. Santos, R. Chaves, and R. L. Pereira. "Cross-Domain Identity and Discovery Framework for Web Calling Services". In: *Annals of Telecommunications* (2017). issn: 1958-9395. doi: 10.1007/s12243-017-0587-2.

[12] S. Göndör. "The Importance of Data Portability and Interoperability in the Social Web". In: *Practical Implementation of the Right to Data Portability - Legal, Technical and Consumer-Related Implications*. Ed. by N. Horn and A. Riechert. ISBN: 978-3-00-058336-0. Stiftung Datenschutz, Nov. 2017.

[13] S. Göndör and A. Küpper. "The Current State of Interoperability in Decentralized Online Social Networking Services". In: *Proceedings of the 2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2017. doi: 10.1109/CSCI.2017.313.

[14] N. Carr. *The Big Switch: Rewiring the World, from Edison to Google*. W.W. Norton & Company, 2009. isbn: 0393333949, 9780393333947.

[15] S. Fox and H. Rainie. *The Web at 25 in the US*. http://www.pewinternet.org/2014/02/25/the-web-at-25-in-the-u-s. Accessed: 6.7.2017. Pew Research Center [Internet & American Life Project], 2014.

[16] Ofcom. *Communications Market Report 2016*. Tech. rep. https://www.ofcom.org.uk/__data/assets/pdf_file/0026/95642/ICMR-Full.pdf. Accessed: 1.6.2017. Ofcom, Dec. 2016.

[17] M. McLuhan. *The Gutenberg Galaxy: The Making of Typographic Man*. Canadian University Paperbooks. University of Toronto Press, Jan. 1962. isbn: 9780802060419.

[18] D. Tapscott. *Grown Up Digital: How the Net Generation is Changing Your World*. 1st ed. Mcgraw-Hill, 2008. isbn: 0071508635, 9780071508636.

[19] M. Petronzio. *A Brief History of Instant Messaging*. http://mashable.com/2012/10/25/instant-messaging-history/. Accessed: 29.5.2017. Oct. 2012.

[20] *Internet Growth Statistics*. Tech. rep. http://www.internetworldstats.com/emarketing.htm. Accessed: 18.7.2017. Miniwatts Marketing Group, 2017.

[21] T. Berners-Lee. "The World Wide Web - Past, Present and Future". In: *Journal of Digital Information* 1.1 (2006). issn: 1368-7506. url: https://journals.tdl.org/jodi/index.php/jodi/article/view/3.

[22] T. Berners-Lee and M. Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor*. 1st. Harper San Francisco, 1999. isbn: 0062515861.

[23] G. Cormode and B. Krishnamurthy. "Key Differences Between Web 1.0 and Web 2.0". In: *First Monday* 13.6 (2008). issn: 13960466. doi: 10.5210/fm.v13i6.2125. url: http://www.ojphi.org/ojs/index.php/fm/article/view/2125.

[24] D. M. Boyd and N. B. Ellison. "Social Network Sites: Definition, History, and Scholarship". In: *Journal of Computer-Mediated Communication* 13.1 (2007), pp. 210–230. issn: 1083-6101. doi: 10.1111/j.1083-6101.2007.00393.x.

[25] L. O'Neill. *Friendster Deploys OpenSocial Support for Benefit of 75 Million Users, Developers and Industry*. http://www.friendster.com:80/info/presscenter.php?A=pr42. Accessed: 23.7.2017 via https://web.archive.org/web/20110623174254/http://www.friendster.com:80/info/presscenter.php?A=pr42. Aug. 2008.

[26] J. Yang and J. Leskovec. "Defining and Evaluating Network Communities Based on Ground-Truth". In: *2012 IEEE 12th International Conference on Data Mining*. 2012, pp. 745–754. doi: 10.1109/ICDM.2012.138.

[27] E. Barnett. *MySpace Loses 10 Million Users in a Month*. http://www.telegraph.co.uk/technology/myspace/8404510/MySpace-loses-10-million-users-in-a-month.html. Accessed: 15.7.2017. Mar. 2011.

[28] V. Cosenza. *World mMp of Social Networks*. http://vincos.it/world-map-of-social-networks/. Accessed: 12.6.2017. 2016.

[29] Facebook. *Company Info*. https://newsroom.fb.com/company-info/. Accessed: 16.7.2017. 2017.

[30] M. Hicks. *500 Million Stories*. https://www.facebook.com/notes/facebook/500-million-stories/409753352130. Accessed: 16.7.2017. July 2010.

[31] M. Zuckerberg. *One Billion People on Facebook*. https://newsroom.fb.com/news/2012/10/one-billion-people-on-facebook/. Accessed: 16.7.2017. Oct. 2012.

[32] A. Acquisti and R. Gross. "Imagined Communities: Awareness, Information Sharing, and Privacy on the Facebook". In: *International Workshop on Privacy Enhancing Technologies*. Springer. 2006, pp. 36–58.

[33] T. Gerace and R. Barbour. *Computer Method and Apparatus for Targeting Advertising*. US Patent App. 11/451,995. 2006. url: https://www.google.com/patents/US20060282328.

[34] G. Orwell. *1984*. Houghton Mifflin Harcourt, 1983. isbn: 9780547249643.

[35] I. Graef. "Mandating Portability and Interoperability in Online Social Networks: Regulatory and Competition Law Issues in the European Union". In: *Telecommunications Policy* 39.6 (2015), pp. 502–514.

[36] J. Heidemann, A. Landherr, F. Probst, M. Klier, and F. Calmbach. "Special Interest Networks — Eine Fallstudie am Beispiel von Netzathleten.de". In: *HMD Praxis der Wirtschaftsinformatik* 48.6 (2011), pp. 103–112.

[37] M. Nowak and G. Spiller. *Two Billion People Coming Together on Facebook*. https://newsroom.fb.com/news/2017/06/two-billion-people-coming-together-on-facebook/. Accessed: 3.7.2017. June 2017.

[38] S. Greenwood, A. Perrin, and M. Duggan. *Social Media Update 2016*. Tech. rep. http://www.pewinternet.org/2016/11/11/social-media-update-2016/ Accessed on 30.5.2017. PEW Research Center, 2016.

[39] *Number of Social Network Users Worldwide from 2010 to 2021 (in Billions)*. Tech. rep. https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/ Accessed on 23.7.2017. Statista, 2017.

[40] B. Fitzpatrick and D. Recordon. *Thoughts on the Social Graph*. http://bradfitz.com/social-graph-problem/ Accessed: 15.5.2017. 2007.

[41] M. Curtiss, I. Becker, T. Bosman, S. Doroshenko, L. Grijincu, T. Jackson, S. Kunnatur, S. Lassen, P. Pronin, S. Sankar, et al. "Unicorn: A System for Searching the Social Graph". In: *Proceedings of the VLDB Endowment* 6.11 (2013), pp. 1150–1161.

[42] V. Venkataramani, Z. Amsden, N. Bronson, G. Cabrera III, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, J. Hoon, et al. "TAO: How Facebook Serves the Social Graph". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM. 2012, pp. 791–792.

[43] M. Newman and J. Park. "Why Social Networks are Different From Other Types of Networks". In: *Physical Review E* 68.3 (2003).

[44] M. Roth, A. Ben-David, D. Deutscher, G. Flysher, I. Horn, A. Leichtberg, N. Leiser, Y. Matias, and R. Merom. "Suggesting Friends Using the Implicit Social Graph". In: *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2010, pp. 233–242.

[45] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Vol. 8. Cambridge University Press, 1994.

[46] G. Magno, G. Comarela, D. Saez-Trumper, M. Cha, and V. Almeida. "New Kid on the Block: Exploring the Google+ Social Graph". In: *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*. ACM. 2012, pp. 159–170.

[47] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. "Measurement and Analysis of Online Social Networks". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. ACM. 2007, pp. 29–42.

[48] Y.-Y. Ahn, S. Han, H. Kwak, S. Moon, and H. Jeong. "Analysis of Topological Characteristics of Huge Online Social Networking Services". In: *Proceedings of the 16th International Conference on World Wide Web*. ACM. 2007, pp. 835–844.

[49] D. J. Watts and S. H. Strogatz. "Collective Dynamics of 'Small-World' Networks". In: *Nature* 393.6684 (1998), p. 440.

[50] J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. "The Anatomy of the Facebook Social Graph". In: *arXiv preprint arXiv:1111.4503* (2011).

[51] J. Travers and S. Milgram. "The Small World Problem". In: *Phychology Today* 1 (1967), pp. 61–67.

[52] J. Kleinberg. "The Convergence of Social and Technological Networks". In: *Communications of the ACM* 51.11 (2008), pp. 66–72.

[53] K. Hampton, L. Sessions Goulet, L. Rainie, and K. Purcell. *Social Networking Sites and Our Lives*. Tech. rep. `http://www.pewinternet.org/2011/06/16/social-networking-sites-and-our-lives/`. Accessed: 21.6.2017. PEW Research Center, 2011.

[54] J. Bonneau, J. Anderson, R. Anderson, and F. Stajano. "Eight Friends are Enough: Social Graph Approximation via Public Listings". In: *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. ACM. 2009, pp. 13–18.

[55] J. LeBlanc. *Programming Social Applications: Building Viral Experiences with OpenSocial, OAuth, OpenID, and Distributed Web Frameworks*. O'Reilly Media, 2011. isbn: 9781449315276.

[56] H. Krasnova, S. Spiekermann, K. Koroleva, and T. Hildebrand. "Online Social Networks: Why We Disclose". In: *Journal of Information Technology* 25.2 (2010), pp. 109–125.

[57] A. De Salve, B. Guidi, and P. Mori. "Predicting the Availability of Users' Devices in Decentralized Online Social Networks". In: *Concurrency and Computation: Practice and Experience* (2017).

[58] J. C. Westland. "Critical Mass and Willingness to Pay for Social Networks". In: *Electronic Commerce Research and Applications* 9.1 (2010), pp. 6–19.

[59] J. Mander. *Internet Users Have Average of 5.54 Social Media Accounts*. Tech. rep. `http://blog.globalwebindex.net/chart-of-the-day/internet-users-have-average-of-5-54-social-media-accounts/` Accessed on 18.7.2017. Global Web Index, 2015.

[60] J. G. Palfrey and U. Gasser. *Interop: The Promise and Perils of Highly Interconnected Systems*. Basic Books, 2012.

[61] A. Peslak, W. Ceccucci, and P. Sendall. "An Empirical Study of Instant Messaging (IM) Behavior Using Theory of Reasoned Action". In: *Journal of Behavioral and Applied Management* 11.3 (2010), p. 263.

[62] European Commission. *Case M.7217 – Facebook/WhatsApp. Commission Decision Pursuant to Article 6(1)(b) of Council Regulation. No 139/2004*. `http://ec.europa.eu/competition/mergers/cases/decisions/m7217_20141003_20310_3962132_EN.pdf`. Accessed: 7.7.2017. 2014.

[63] B. A. Nardi, S. Whittaker, and E. Bradner. "Interaction and Outeraction: Instant Messaging in Action". In: *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*. ACM. 2000, pp. 79–88.

[64] A. Lenhart, O. Lewis, and L. Rainie. *Teenage Life Online*. Tech. rep. `http://www.pewinternet.org/2001/06/21/teenage-life-online/`. Accessed: 7.7.2017. PEW Research Center, June 2001.

[65] A. Lenhart, P. Hiltin, and M. Madden. *Teens and Technology*. Tech. rep. http://www.pewinternet.org/2005/07/27/teens-and-technology/. Accessed: 7.7.2017. PEW Research Center, July 2005.

[66] A. Lenhart and M. Madden. *Social Networking Websites and Teens*. Tech. rep. http://www.pewinternet.org/2007/01/07/social-networking-websites-and-teens/. Accessed: 7.7.2017. PEW Research Center, Jan. 2007.

[67] B. S. Boneva, A. Quinn, R. Kraut, S. Kiesler, and I. Shklovski. "Teenage Communication in the Instant Messaging Era". In: *Computers, Phones, and the Internet: Domesticating Information Technology* (2006), pp. 201–218.

[68] K. J. Wood. "Instant Messaging Usage and Academic and Social Integration". Master thesis. Virginia Polytechnic Institute and State University, 2007.

[69] Time Warner. *ICQ Celebrates 100 Million Registered Users*. http://www.timewarner.com/newsroom/press-releases/2001/05/09/icq-celebrates-100-million-registered-users. Accessed: 2.6.2017. May 2001.

[70] A. D. Network. *OSCAR Protocol*. https://web.archive.org/web/20080308233204/http://dev.aol.com/aim/oscar/. Accessed: 17.6.2017. 2014.

[71] D. Oleynichenko. *ICQ: 20 Years Is No Limit!* https://medium.com/@Dimitryophoto/icq-20-years-is-no-limit-8734e1eea8ea. Accessed: 29.5.2017. Nov. 2016.

[72] P. Saint-Andre, K. Smith, and R. Tronçon. *XMPP: The Definitive Guide*. O'Reilly, 2009.

[73] S. Gilbertson. *Slap in the Facebook: It's Time for Social Networks to Open Up*. https://www.wired.com/2007/08/open-social-net/ Accessed: 21.5.2017. 2007.

[74] *Most Famous Social Network Sites Worldwide as of August 2017, Ranked by Number of Active Users (in Millions)*. Tech. rep. https://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/. Accessed: 15.9.2017. Statista, 2017.

[75] E. Enge. *Hard Numbers for Public Posting Activity on Google Plus*. Tech. rep. https://www.stonetemple.com/real-numbers-for-the-activity-on-google-plus/. Accessed: 19.7.2017. StoneTemple, Apr. 2015.

[76] S. Denning. *Has Google+ Really Died?* Tech. rep. https://www.forbes.com/sites/stevedenning/2015/04/23/has-google-really-died. Accessed: 19.7.2017. Forbes, Apr. 2015.

[77] M. Seemann. *Dezentrale Social Networks – Warum sie scheitern und es gehen könnte*. http://14.re-publica.de/session/dezentrale-social-networks-warum-sie-scheitern-und-es-gehen-koennte Accessed: 14.5.2017. re:publica 14, 2014.

[78] D. S. Evans and R. Schmalensee. "Failure to Launch: Critical Mass in Platform Businesses". In: *Review of Network Economics* 9.4 (2010).

[79] M. L. Katz and C. Shapiro. "Network Externalities, Competition, and Compatibility". In: *The American Economic Review* 75.3 (1985), pp. 424–440.

[80] Independent. *Facebook may Lock-In its Internet Dominance*. `http://www.independent.co.uk/incoming/facebook-may-lock-in-its-internet-dominance-5519916.html`. Accessed: 2.6.2017. 2010.

[81] T. Berners-Lee. *Socially Aware Cloud Storage*. 2009. url: `https://www.w3.org/DesignIssues/CloudStorage.html` (visited on 07/12/2017).

[82] S. W. Waller. "Antitrust and Social Networking". In: *North Carolina Law Rev.* 90 (2011), p. 1771.

[83] *Number of Renren.com Users in China from 2009 to 2016 (in Millions)*. Tech. rep. `https://www.statista.com/statistics/227059/number-of-renren-com-users-in-china/`. Accessed: 15.9.2017. Statista, 2017.

[84] O. Malik. *Is Facebook Beacon a Privacy Nightmare?* Tech. rep. `https://gigaom.com/2007/11/06/facebook-beacon-privacy-issues/`. Accessed: 6.6.2017. 2007.

[85] C. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-Lee. "Decentralization: The Future of Online Social Networking". In: *W3C Workshop on the Future of Social Networking Position Papers*. Vol. 2. 2009.

[86] F. Stutzman, R. Gross, and A. Acquisti. "Silent Listeners: The Evolution of Privacy and Disclosure on Facebook". In: *Journal of Privacy and Confidentiality* 4.2 (2013), p. 2.

[87] A. Lewis. *User-Driven Discontent*. `http://www.metafilter.com/95152/Userdriven-discontent#3256046`. Accessed: 24.7.2017. Aug. 2010.

[88] A. Datta, S. Buchegger, L.-H. Vu, T. Strufe, and K. Rzadca. "Decentralized Online Social Networks". In: *Handbook of Social Network Technologies and Applications*. Springer, 2010, pp. 349–378.

[89] S. Buchegger, D. Schiöberg, L.-H. Vu, and A. Datta. "PeerSoN: P2P Social Networking: Early Experiences and Insights". In: *Proceedings of the Second ACM EuroSys Workshop on Social Network Systems*. ACM. 2009, pp. 46–52.

[90] M. Durr, M. Maier, and F. Dorfmeister. "Vegas - A Secure and Privacy-Preserving Peer-to-Peer Online Social Network". In: *International Conference on and 2012 International Conference on Social Computing (SocialCom) Privacy, Security, Risk and Trust (PASSAT)*. IEEE. 2012, pp. 868–874.

[91] J. A. Pouwelse, P. Garbacki, J. Wang, A. Bakker, J. Yang, A. Iosup, D. H. Epema, M. Reinders, M. R. Van Steen, and H. J. Sips. "TRIBLER: A Social-Based Peer-to-Peer System". In: *Concurrency and Computation: Practice and Experience* 20.2 (2008), pp. 127–138.

[92] C. Wilson, T. Steinbauer, G. Wang, A. Sala, H. Zheng, and B. Y. Zhao. "Privacy, Availability and Economics in the Polaris Mobile Social Network". In: *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*. ACM. 2011, pp. 42–47.

[93] Diaspora. *An Introduction to the Diaspora Source*. https://wiki.diasporafoundation.org/An_introduction_to_the_Diaspora_source, Accessed: 9.1.2017. 2015.

[94] Friendica. *Develop*. http://friendica.com/develop, Accessed: 15.1.2017.

[95] L. A. Cutillo, R. Molva, and T. Strufe. "Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-Life Trust". In: *Communications Magazine, IEEE* 47.12 (2009), pp. 94–101.

[96] A. Bielenberg, L. Helm, A. Gentilucci, D. Stefanescu, and H. Zhang. "The Growth of Diaspora - A Decentralized Online Social Network in the Wild". In: *2012 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2012, pp. 13–18.

[97] A. Bleicher. "The Anti-Facebook". In: *IEEE Spectrum* 48.6 (2011), pp. 54–82.

[98] W. Chao, Y. Guo, and B. Zhou. "Social Networking Federation: A Position Paper". In: *Computers & Electrical Engineering* 38.2 (2012), pp. 306–329.

[99] F. McCown and M. L. Nelson. "What Happens When Facebook is Gone?" In: *Proceedings of the 9th ACM/IEEE-CS Joint Conference on Digital Libraries*. ACM. 2009, pp. 251–254.

[100] P. Hu, Q. Fan, and W. C. Lau. "SNSAPI: A Cross-Platform Middleware for Rapid Deployment of Decentralized Social Networks". In: *arXiv preprint arXiv:1403.4482* (2014).

[101] D. Koll, J. Li, and X. Fu. "The Good Left Undone: Advances and Challenges in Decentralizing Online Social Networks". In: *Computer Communications* (2017).

[102] James Snell and Evan Prodromou. *Activity Streams 2.0*. http://www.w3.org/TR/activitystreams-core/. Accessed: 9.5.2017. 2017.

[103] P. Jones, G. Salgueiro, M. Jones, and J. Smarr. *WebFinger*. http://tools.ietf.org/html/rfc7033. Accessed: 1.8.2017. 2013.

[104] D. Brickley and L. Miller. *FOAF Vocabulary Specification 0.99*. Tech. rep. http://xmlns.com/foaf/spec/. Accessed: 1.8.2017. W3C, 2014.

[105] D. Appelquist, D. Brickley, M. Carvahlo, R. Iannella, A. Passant, C. Perey, and H. Story. "A Standards-Based, Open and Privacy-Aware Social Web". In: *W3C Incubator Group Report* 6 (2010).

[106] T. Paul, S. Buchegger, and T. Strufe. "Decentralized Social Networking Services". In: *Trustworthy Internet*. Springer, 2011, pp. 187–199.

[107] M. Marcon, B. Viswanath, M. Cha, and K. P. Gummadi. "Sharing Social Content from Home: A Measurement-Driven Feasibility Study". In: *Proceedings of the 21st International Workshop on Network and Operating Systems Support for Digital Audio and Video*. ACM. 2011, pp. 45–50.

[108] European Court. *Judgment of the Court of 14 February 1978. United Brands Company and United Brands Continentaal BV v Commission of the European Communities. Chiquita Bananas. Case 27/76.* http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:61976CJ0027. Accessed: 8.7.2017. 1978.

[109] European Parliament. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons With Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation)*. http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN. Accessed: 15.9.2017. 2016.

[110] T. Sperlich. "Das Recht auf Datenübertragbarkeit". In: *Datenschutz und Datensicherheit - DuD* 41.6 (June 2017), pp. 377–377. issn: 1862-2607.

[111] T. Paul, A. Famulari, and T. Strufe. "A Survey on Decentralized Online Social Networks". In: *Computer Networks* 75, Part A (2014), pp. 437–452. issn: 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2014.10.005.

[112] Wired. *Facebook Camera Means Snapping Is Officially the Future*. https://www.wired.com/2017/03/facebook-camera-means-snapping-officially-future/. Accessed: 1.6.2017. Mar. 2017.

[113] J. Heidemann. "Online Social Networks - Ein sozialer und technischer Überblick". In: *Informatik-Spektrum* 33.3 (2010), pp. 262–271.

[114] Diso Project. *Activity Streams*. http://activitystrea.ms/. Accessed: 9.5.2017. 2013.

[115] M. Halvorson. *OpenSocial Specification*. https://opensocial.atlassian.net/wiki/display/OSD/Specs. Accessed: 23.6.2017. 2013.

[116] E. Prodromou, B. Vibber, J. Walker, and Z. Copley. *OStatus 1.0 Draft 2*. Aug. 30, 2010. url: http://ostatus.github.io/spec/OStatus%201.0%20Draft%202.html (visited on 06/23/2017).

[117] A. R. Hevner, S. T. March, J. Park, and S. Ram. "Design Science in Information Systems Research". In: *MIS Quarterly* 28.1 (2004), pp. 75–105.

[118] A. R. Hevner. "A Three Cycle View of Design Science Research". In: *Scandinavian Journal of Information Systems* 19.2 (2007), p. 4.

[119] N. B. Ellison and D. M. Boyd. "Sociality Through Social Network Sites". In: *The Oxford Handbook of Internet Studies*. 2013.

[120] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. "Understanding Online Social Network Usage from a Network Perspective". In: *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement Conference*. ACM. 2009, pp. 35–48.

[121] L. Adamic and E. Adar. "How to Search a Social Network". In: *Social Networks* 27.3 (2005), pp. 187–203.

[122] G. Pallis, D. Zeinalipour-Yazti, and M. D. Dikaiakos. "Online Social Networks: Status and Trends". In: *New Directions in Web Data Management 1*. Springer, 2011, pp. 213–234.

[123] W. Kim, O.-R. Jeong, and S.-W. Lee. "On Social Web Sites". In: *Information Systems* 35.2 (2010), pp. 215–236.

[124] A. M. Kaplan and M. Haenlein. "Users of the World, Unite! The Challenges and Opportunities of Social Media". In: *Business Horizons* 53.1 (2010), pp. 59–68.

[125] A. Richter and M. Koch. "Functions of Social Networking Services". In: *Proc. Intl. Conf. on the Design of Cooperative Systems*. 2008, pp. 87–98.

[126] A. M. Kaplan and M. Haenlein. "The Early Bird Catches the News: Nine Things You Should Know About Micro-Blogging". In: *Business Horizons* 54.2 (2011), pp. 105–113.

[127] J. Heidemann, M. Klier, and F. Probst. "Online Social Networks: A Survey of a Global Phenomenon". In: *Computer Networks* 56.18 (2012), pp. 3866–3878. issn: 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2012.08.009.

[128] S. R. Chowdhury, A. R. Roy, M. Shaikh, and K. Daudjee. "A Taxonomy of Decentralized Online Social Networks". In: *Peer-to-Peer Networking and Applications* 8.3 (2015), pp. 367–383.

[129] *Entwicklung der Visits der VZ-Netzwerke von Dezember 2010 bis Dezember 2012 (in Millionen Visits)*. Tech. rep. https://de.statista.com/statistik/daten/studie/235531/umfrage/entwicklung-der-visits-der-deutschen-social-networks/. Accessed: 29.10.2017. Statista, 2013.

[130] P. Golgher. *Adeus ao Orkut*. https://brasil.googleblog.com/2014/06/adeus-ao-orkut.html. Accessed: 29.10.2017. June 2014.

[131] X. Fu, J. Luo, and M. Boos. *Social Network Analysis: Interdisciplinary Approaches and Case Studies*. CRC Press, 2017. isbn: 9781498736688. url: https://books.google.de/books?id=59WRDgAAQBAJ.

[132] S. Biedermann, N. P. Karvelas, S. Katzenbeisser, T. Strufe, and A. Peter. "ProofBook: An Online Social Network Based on Proof-of-Work and Friend-Propagation". In: *Proceedings of the 40th International Conference on Current Trends in Theory and Practice of Computer Science*. Springer. 2014, pp. 114–125.

[133] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, and A. Lysyanskaya. "Incentivizing Outsourced Computation". In: *Proceedings of the 3rd International Workshop on Economics of Networked Systems*. ACM. 2008, pp. 85–90.

[134] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. "Making P2P Accountable Without Losing Privacy". In: *Proceedings of the 2007 ACM Workshop on Privacy in Electronic Society*. ACM. 2007, pp. 31–40.

[135] L. M. Aiello and G. Ruffo. "LotusNet: Tunable Privacy for Distributed Online Social Network Services". In: *Computer Communications* 35.1 (2012), pp. 75–88.

[136] N. Kourtellis, J. Finnis, P. Anderson, J. Blackburn, C. Borcea, and A. Iamnitchi. "Prometheus: User-Controlled P2P Social Data Management for Socially-Aware Applications". In: *Proceedings of the ACM/IFIP/USENIX 11th International Conference on Middleware*. Springer-Verlag. 2010, pp. 212–231.

[137] K. Graffi, C. Gross, D. Stingl, D. Hartung, A. Kovacevic, and R. Steinmetz. "LifeSocial.KOM: A Secure and P2P-Based Solution for Online Social Networks". In: *Consumer Communications and Networking Conference (CCNC)*. IEEE. 2011, pp. 554–558.

[138] S. Jahid, S. Nilizadeh, P. Mittal, N. Borisov, and A. Kapadia. "DECENT: A Decentralized Architecture for Enforcing Privacy in Online Social Networks". In: *2012 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. IEEE. 2012, pp. 326–332.

[139] B. Guidi, T. Amft, A. De Salve, K. Graffi, and L. Ricci. "DiDuSoNet: A P2P Architecture for Distributed Dunbar-based Social Networks". In: *Peer-to-Peer Networking and Applications* 9.6 (2016), pp. 1177–1194.

[140] C. Newton. *Mastodon.social is an Open-Source Twitter Competitor that's Growing Like Crazy*. `https://www.theverge.com/2017/4/4/15177856/mastodon-social-network-twitter-clone`. Accessed: 7.8.2017. Apr. 2017.

[141] *OneSocialWeb*. Tech. rep. `http://onesocialweb.org/about.html`. Accessed on 1.8.2017 via `http://web.archive.org/web/20141230081222/http://onesocialweb.org/about.html`. Vodafone RD, 2011.

[142] P. Saint-Andre and D. Cridland. *XEP-0001: XMPP Extension Protocols*. Tech. rep. `https://xmpp.org/extensions/xep-0001.html`. Accessed: 1.8.2017. XMPP Standards Foundation, 2016.

[143] L. Eschenauer. *SocialRelationships*. Tech. rep. `http://onesocialweb.org:80/spec/1.0/osw-relations.html`. Accessed on 2.8.2017 via `http://web.archive.org/web/20141013124054/http://onesocialweb.org:80/spec/1.0/osw-relations.html`. Vodafone RD, 2011.

[144] S.-W. Seong, J. Seo, M. Nasielski, D. Sengupta, S. Hangal, S. K. Teh, R. Chu, B. Dodson, and M. S. Lam. "PrPl: A Decentralized Social Networking Infrastructure". In: *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*. ACM. 2010, p. 8.

[145] L. Schwittmann, C. Boelmann, M. Wander, and T. Weis. "SoNet – Privacy and Replication in Federated Online Social Networks". In: *33rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE. 2013, pp. 51–57.

[146] A. Famulari and A. Hecker. "Mantle: A Novel DOSN Leveraging Free Storage and Local Software". In: *Advanced Infocom Technology*. Springer, 2013, pp. 213–224.

[147] R. Baden, A. Bender, N. Spring, B. Bhattacharjee, and D. Starin. "Persona: An Online Social Network with User-Defined Privacy". In: *ACM SIGCOMM Computer Communication Review*. Vol. 39. 4. ACM. 2009, pp. 135–146.

[148] J. Bethencourt, A. Sahai, and B. Waters. "Ciphertext-Policy Attribute-Based Encryption". In: *IEEE Symposium on Security and Privacy*. IEEE. 2007, pp. 321–334.

[149] A. Shakimov, H. Lim, R. Cáceres, L. P. Cox, K. Li, D. Liu, and A. Varshavsky. "Vis-à-Vis: Privacy-Preserving Online Social Networking via Virtual Individual Servers". In: *3rd International Conference on Communication Systems and Networks (COMSNETS)*. IEEE. 2011, pp. 1–10.

[150] D. Liu, A. Shakimov, R. Cáceres, A. Varshavsky, and L. P. Cox. "Confidant: protecting OSN data without locking it up". In: *Proceedings of the 12th International Middleware Conference*. International Federation for Information Processing. 2011, pp. 60–79.

[151] R. Sharma and A. Datta. "SuperNova: Super-Peers based Architecture for Decentralized Online Social Networks". In: *4th International Conference on Communication Systems and Networks (COMSNETS)*. IEEE. 2012, pp. 1–10.

[152] J. Panzer. *Salmon Protocol*. `http://www.salmon-protocol.org/`. Accessed: 17.6.2017. Jan. 2011.

[153] R. Khare and T. Çelik. *Microformats: A Pragmatic Path to the Semantic Web*. Tech. rep. `https://commerce.net/wp-content/uploads/2012/04/CN-TR-06-01.pdf`. Accessed: 2.8.2017. CommerceNet, 2006.

[154] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Core*. Tech. rep. `https://tools.ietf.org/html/rfc6120`. Accessed: 31.7.2017. IETF, 2011.

[155] B. Fitzpatrick, B. Slatkin, M. Atkins, and J. Genestoux. *PubSubHubbub Core 0.4*. `http://pubsubhubbub.github.io/PubSubHubbub/pubsubhubbub-core-0.4.html`. Accessed: 23.6.2017. 2014.

[156] D. Strype. *A Brief History of the GNU Social Fediverse and 'The Federation'*. `https://www.coactivate.org/projects/disintermedia/blog/2017/04/01/a-brief-history-of-the-gnu-social-fediverse-and-the-federation/`. Accessed: 29.10.2017. Apr. 2017.

[157] M. Atkins, W. Norris, C. Messina, M. Wilkinson, and R. Dolin. *Atom Activity Streams 1.0*. Tech. rep. `http://activitystrea.ms/specs/atom/1.0/`. Accessed: 1.10.2017. Activity Streams Working Group, 2011.

[158] Apollic Software, LLC. *Tent.io*. `https://tent.io/`. Accessed: 1.8.2017. 2013.

[159] Pump.io. *Pump.io API*. `https://github.com/pump-io/pump.io/blob/master/API.md`. Accessed: 7.8.2017. 2017.

[160] J. Gregorio and B. de Hora. *The Atom Publishing Protocol*. Tech. rep. `https://tools.ietf.org/html/rfc5023`. Accessed: 7.8.2017. IETF, 2007.

[161] E. Hammer-Lahav. *The OAuth 1.0 Protocol*. Tech. rep. `https://tools.ietf.org/html/rfc5849`. Accessed: 7.8.2017. IETF, 2010.

[162] M. Macgirvin. *DFRN – The Distributed Friends & Relations Network*. `https://github.com/friendica/friendica/blob/master/spec/dfrn2.pdf` Accessed: 23.6.2017. 2011.

[163] M. Nottingham and R. Sayre. *The Atom Syndication Format*. Tech. rep. https://tools.ietf.org/html/rfc4287. Accessed: 5.7.2017. IETF, 2005.

[164] M. Macgirvin. *Hubzilla Documentation: Developers*. https://project.hubzilla.org/help/developer/zot_protocol Accessed: 19.10.2017. 2015.

[165] T. Çelik. *hCard*. http://microformats.org/wiki/h-card. Accessed: 17.6.2017. June 2017.

[166] J. Panzer, B. Laurie, and D. Balfanz. *Magic Signatures*. https://cdn.rawgit.com/salmon-protocol/salmon-protocol/master/draft-panzer-magicsig-01.html. Accessed: 17.6.2017. Jan. 2011.

[167] *Diaspora* Federation Protocol*. Tech. rep. https://diaspora.github.io/diaspora_federation/. Accessed: 17.5.2017. Diaspora Foundation, 2017.

[168] C. A. Webber, J. Tallon, and O. Shepherd. *ActivityPub*. Tech. rep. https://www.w3.org/TR/activitypub/. Accessed: 8.8.2017. W3C, 2017.

[169] ISO/IEC. *ISO/IEC 29115:2013: Information Technology – Security Techniques – Entity Authentication Assurance Framework*. https://www.iso.org/standard/45138.html. Accessed: 1.6.2017. Apr. 2013.

[170] L. Chen, M. A. Babar, and B. Nuseibeh. "Characterizing Architecturally Significant Requirements". In: *IEEE software* 30.2 (2013), pp. 38–45.

[171] Forbes. *Facebook Will Use Your Browsing and Apps History For Ads*. https://www.forbes.com/sites/kashmirhill/2014/06/13/facebook-web-app-tracking-for-ads/#44aff2635d58 Accessed: 22.5.2017. 2014.

[172] D. Boyd. "Why Youth (Heart) Social Network Sites: The Role of Networked Publics in Teenage Social Life". In: *MacArthur Foundation Series on Digital Learning – Youth, Identity, and Digital Media Volume* (2007), pp. 119–142.

[173] V. A. Rohani and O. S. Hock. "On Social Network Web Sites: Definition, Features, Architectures and Analysis Tools". In: *Journal of Computer Engineering* 1 (2009), pp. 3–11.

[174] B. Kneidinger. *Facebook und Co*. Springer, 2010.

[175] J. H. Kietzmann, K. Hermkens, I. P. McCarthy, and B. S. Silvestre. "Social media? Get serious! Understanding the Functional Building Blocks of Social Media". In: *Business Horizons* 54.3 (2011), pp. 241–251.

[176] K. Tserpes, G. Papadakis, M. Kardara, A. Papaoikonomou, F. Aisopos, E. Sardis, and T. A. Varvarigou. "An Ontology for Social Networking Sites Interoperability". In: *Proceedings of the 4th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*. 2012, pp. 245–250.

[177] Vincenzo Cosenza. *World Map or Social Networks*. http://vincos.it/world-map-of-social-networks/. Accessed: 8.6.2017. 2017.

[178] *Social Media Fact Sheet*. Tech. rep. http://www.pewinternet.org/fact-sheet/social-media/ Accessed on 23.7.2017. PEW Research Center, 2017.

[179] Alexa. *Top Sites in Russia*. http://www.alexa.com/topsites/countries/RU. Accessed: 8.6.2017. 2017.

[180] Kai Lukoff. *China's Top Four Social Networks: RenRen, Kaixin001, Qzone, and 51.com*. https://venturebeat.com/2010/04/07/chinas-top-4-social-networks-renren-kaixin001-qzone-and-51-com/. Accessed: 8.6.2017. 2010.

[181] Facebook. *Send Money to Friends in Messenger*. https://newsroom.fb.com/news/2015/03/send-money-to-friends-in-messenger/. Accessed: 20.5.2017. Mar. 2015.

[182] Facebook. *Reactions Now Available Globally*. https://en.facebookbrand.com/assets/reactions. Accessed: 20.6.2017. Feb. 2016.

[183] M. Ku. *Introducing Marketplace: Buy and Sell With Your Local Community*. https://newsroom.fb.com/news/2016/10/introducing-marketplace-buy-and-sell-with-your-local-community/. Accessed: 28.10.2017. Oct. 2016.

[184] S. Kairam, M. Brzozowski, D. Huffaker, and E. Chi. "Talking in Circles: Selective Sharing in Google+". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM. 2012, pp. 1065–1074.

[185] D. Schiöberg, F. Schneider, H. Schiöberg, S. Schmid, S. Uhlig, and A. Feldmann. "Tracing the Birth of an OSN: Social Graph and Profile Analysis in Google+". In: *Proceedings of the 4th Annual ACM Web Science Conference*. ACM. 2012, pp. 265–274.

[186] K. Baran and W. Stock. "Facebook has Been Smacked Down. The Russian Special Way of SNSs: Vkontakte as a Case Study". In: *Proceedings of the 2nd European Conference on Social Media (ECSM 2015), 9.-10. July 2015, Porto, Portugal*. 2015, pp. 574–582.

[187] S. Khveshchanka and L. Suter. "Vergleichende Analyse von profilbasierten sozialen Netzwerken aus Russland (VKontakte), Deutschland (StudiVZ) und den USA (Facebook)". In: *Information–Wissenschaft und Praxis* 61.2 (2010), p. 2010.

[188] A. Rosen and I. Ihara. *Giving You More Characters to Express Yourself*. https://blog.twitter.com/official/en_us/topics/product/2017/Giving-you-more-characters-to-express-yourself.html. Accessed: 30.11.2017. Sept. 2017.

[189] M. Zarro and C. Hall. "Pinterest: Social Collecting for #Linking #Using #Sharing". In: *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries*. ACM. 2012, pp. 417–418.

[190] D. Boyd. "Friends, Friendsters, and Top 8: Writing Community Into Being on Social Network Sites". In: *First Monday* 11.12 (2006). http://www.firstmonday.org/ojs/index.php/fm/article/view/1418. Accessed: 7.7.2017. issn: 13960466. doi: 10.5210/fm.v11i12.1418.

[191] A. Monitzer. *XEP-0224: Attention*. Tech. rep. https://xmpp.org/extensions/xep-0224.html. Accessed: 2.6.2017. XMPP Standards Foundation, 2008.

[192] Life Before Us, LLC. *What is Yo?* http://docs.justyo.co/ Accessed: 3.5.2017. 2017.

[193] K. O'Rourke. *Goodbye, Like Button*. https://blog.pinterest.com/en/goodbye-button. Accessed: 9.6.2017. Apr. 2017.

[194] Facebook. *Facebook Graph API Reference v2.9*. https://developers.facebook.com/docs/graph-api/reference/v2.9/. Accessed: 17.6.2017. June 2017.

[195] B. Goode. "Voice Over Internet Protocol (VoIP)". In: *Proceedings of the IEEE* 90.9 (2002), pp. 1495–1517.

[196] James Snell and Evan Prodromou. *Activity Vocabulary*. https://www.w3.org/TR/activitystreams-vocabulary/. Accessed: 9.5.2017. 2017.

[197] Facebook Brand Resource Center. *News Feed*. https://en.facebookbrand.com/assets/newsfeed, Accessed: 21.5.2017. 2017.

[198] T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. https://www.tools.ietf.org/html/rfc3986. Accessed: 3.6.2017. 2005.

[199] P. Mockapetris. *Domain Names - Concepts and Facilities*. Tech. rep. https://tools.ietf.org/html/rfc1034. Accessed: 23.8.2017. IETF, 1987.

[200] P. Resnick. *Internet Message Format*. https://tools.ietf.org/html/rfc5322. Accessed: 5.6.2017. 2008.

[201] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Address Format*. Tech. rep. https://tools.ietf.org/html/rfc6122. Accessed: 31.7.2017. IETF, 2011.

[202] A. Sambra, S. Corlosquet, H. Story, and T. Berners-Lee. *WebID 1.0: Web Identity and Discovery*. Tech. rep. http://dvcs.w3.org/hg/WebID/raw-file/tip/spec/identity-respec.html. Accessed: 1.8.2017. W3C, 2017.

[203] R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride. *RDF 1.1 Concepts and Abstract Syntax*. Tech. rep. http://www.w3.org/TR/rdf11-concepts/. Accessed: 1.8.2017. W3C, 2014.

[204] H. Story, B. Harbulot, I. Jacobi, and M. Jones. "FOAF+SSL: Restful Authentication for the Social Web". In: *Proceedings of the First Workshop on Trust and Privacy on the Social and Semantic Web (SPOT2009)*. 2009.

[205] H. Story, S. Corlosquet, A. Sambra, T. Inkster, and B. Harbulot. *WebID-TLS: WebID Authentication over TLS*. Tech. rep. https://www.w3.org/TR/auth-webid/. Accessed: 1.8.2017. W3C, 2017.

[206] N. Sakimura, J. Bradley, M. Jones, and E. Jay. *OpenID Connect Discovery 1.0 Incorporating Errata Set 1*. Tech. rep. http://openid.net/specs/openid-connect-discovery-1_0.html. Accessed: 2.8.2017. OpenID Foundation, 2014.

[207] D. Reed and D. McAlpin. *Extensible Resource Identifier (XRI) Syntax V2.0*. https://www.oasis-open.org/committees/download.php/15377. Accessed: 3.6.2017. 2005.

[208] G. Wachob, D Reed, L Chasen, W Tan, and S Churchill. *Extensible Resource Identifier (XRI) Resolution Version 2.0*. http://docs.oasis-open.org/xri/2.0/specs/xri-resolution-V2.0.html. Accessed: 2.6.2017. 2008.

[209] D. Recordon and D. Reed. "OpenID 2.0: A Platform for User-Centric Identity Management". In: *Proceedings of the Second ACM Workshop on Digital Identity Management*. DIM '06. Alexandria, Virginia, USA, 2006, pp. 11–16. isbn: 1-59593-547-9.

[210] P. Leach, M. Mealling, and R. Salz. *A Universally Unique IDentifier (UUID) URN Namespace*. http://tools.ietf.org/html/rfc4122. Accessed: 5.6.2017. 2005.

[211] B. Demir. *Twitter Snowflake*. https://github.com/twitter/snowflake. Accessed: 12.2.2017. 2010.

[212] D. Gardner and L. Vasconcelos. *Cruftflake*. https://github.com/davegardnerisme/cruftflake. Accessed: 12.2.2017. 2015.

[213] D. Featherston, S. Debnath, T. Nyman, A. Veres-Szentkirályi, and M. Countryman. *Boundaryflake*. https://github.com/boundary/flake. Accessed: 12.2.2017. 2015.

[214] M Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*. Tech. rep. http://www.ietf.org/rfc/rfc2251.txt. Accessed: 23.8.2017. IETF, 1997.

[215] K. Zeilenga. *Lightweight Directory Access Protocol (LDAP) Transactions*. Tech. rep. http://tools.ietf.org/html/rfc5805. Accessed: 23.8.2017. IETF, 2010.

[216] J. Sermersheim. *Lightweight Directory Access Protocol (LDAP) The Protocol*. Tech. rep. http://tools.ietf.org/html/rfc4511. Accessed: 23.8.2017. IETF, 2006.

[217] *X.500: Information technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*. Tech. rep. http://www.itu.int/rec/T-REC-X.500/en. Accessed: 23.8.2017. International Telecommunication Union (ITU-T), 2012.

[218] P. Mockapetris. *Domain Names - Implementation and Specification*. Tech. rep. https://tools.ietf.org/html/rfc1035. Accessed: 23.8.2017. IETF, 1987.

[219] A. Oram et al. "Peer-to-Peer: Harnessing the Benefits of a Distruptive Technology". In: *O'Reilly & Associate Inc.* (2001), pp. 67–72.

[220] C. Douligeris and A. Mitrokotsa. "DDoS Attacks and Defense Mechanisms: Classification and State-of-the-Art". In: *Computer Networks* 44.5 (2004), pp. 643–666.

[221] X. Yue, X. Qiu, Y. Ji, and C. Zhang. "P2P Attack Taxonomy and Relationship Analysis". In: *11th International Conference on Advanced Communication Technology (ICACT)*. Vol. 2. IEEE. 2009, pp. 1207–1210.

[222] J. Frankel and T Pepper. *Gnutella Protocol Specification v0.4*. http://web.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf Accessed: 4.5.2017. 2007.

[223] J Liang, R Kumar, and K Ross. "Understanding KaZaA". In: (2004).

[224] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A Scalable Content-Addressable Network*. Vol. 31. 4. ACM, 2001.

[225] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In: *ACM SIGCOMM Computer Communication Review* 31.4 (2001), pp. 149–160.

[226] A. Rowstron and P. Druschel. "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems". In: *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2001, pp. 329–350.

[227] P. Maymounkov and D. Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. isbn: 3-540-44179-4.

[228] V. Ramasubramanian and E. G. Sirer. "Beehive: O(1) Lookup Performance for Power-Law Query Distributions in Peer-to-Peer Overlays". In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*. Vol. 4. 2004, pp. 8–8.

[229] V. Ramasubramanian and E. G. Sirer. "Beehive: Exploiting Power Law Query Distributions for O(1) Lookup Performance in Peer to Peer Overlays". In: *Proceedings of the 1st Symposium on Networked Systems Design and Implementation*. 2004.

[230] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson. "Profiling a Million User DHT". In: *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*. ACM. 2007, pp. 129–134.

[231] Y. Chawathe, S. Ramabhadran, S. Ratnasamy, A. LaMarca, S. Shenker, and J. Hellerstein. "A Case Study in Building Layered DHT Applications". In: *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '05. New York, NY, USA: ACM, 2005, pp. 97–108. isbn: 1-59593-009-4. doi: 10.1145/1080091.1080104.

[232] V. Ramasubramanian and E. G. Sirer. "The Design and Implementation of a Next Generation Name Service for the Internet". In: *ACM SIGCOMM Computer Communication Review* 34.4 (2004), pp. 331–342.

[233] R. Cox, A. Muthitacharoen, and R. T. Morris. "Serving DNS using a peer-to-peer lookup service". In: *International Workshop on Peer-To-Peer Systems*. Springer. 2002, pp. 155–165.

[234] V. Pappas, D. Massey, A. Terzis, and L. Zhang. "A Comparative Study of the DNS Design with DHT-Based Alternatives". In: *Proceedings of the 25th IEEE Conference on Computer Communications (INFOCOM)*. IEEE. 2006, pp. 1–13.

[235] D. Crocker and P. Overell. *Augmented BNF for Syntax Specifications: ABNF*. https://tools.ietf.org/html/rfc5234. Accessed: 21.7.2017. 2008.

[236] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Token (JWT)*. Tech. rep. http://tools.ietf.org/html/rfc7519. Accessed: 23.6.2017. IETF, 2015.

[237] S. Josefsson. *The Base16, Base32, and Base64 Data Encodings*. Tech. rep. https://tools.ietf.org/html/rfc4648. Accessed: 1.8.2017. IETF, 2006.

[238] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polkk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. https://tools.ietf.org/html/rfc5280. Accessed: 5.6.2017. 2008.

[239] A. K. Lenstra, E. Tromer, A. Shamir, W. Kortsmit, B. Dodson, J. Hughes, and P. Leyland. "Factoring Estimates for a 1024-bit RSA Modulus". In: *Asiacrypt*. Vol. 2894. Springer. 2003, pp. 55–74.

[240] Patarin, J and Montreuil, A. *Benes and Butterfly Schemes Revisited*. Cryptology ePrint Archive, Report 2005/004. 2005.

[241] N. Farrington and A. Andreyev. "Facebook's Data Center Network Architecture". In: *Optical Interconnects Conference, 2013 IEEE*. IEEE. 2013, pp. 49–50.

[242] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. "Inside the Social Network's (Datacenter) Network". In: *ACM SIGCOMM Computer Communication Review*. Vol. 45. 4. ACM. 2015, pp. 123–137.

[243] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. C. Li, et al. "TAO: Facebook's Distributed Data Store for the Social Graph". In: *USENIX Annual Technical Conference*. 2013, pp. 49–60.

[244] D. Austin, A. Barbir, C. Ferris, and S. Garg. *Web Services Architecture Requirements*. Tech. rep. http://www.w3.org/TR/wsa-reqs. Accessed: 29.7.2017. W3C, Nov. 2002.

[245] I. G. Alonso et al. *Future Internet Design Principles*. Tech. rep. http://www.future-internet.eu/uploads/media/FIArch_Design_Principles_V1.0.pdf. Accessed: 4.8.2017. Future Internet Architecture (FIArch) Group, 2012.

[246] A. De Salve, B. Guidi, and L. Ricci. "An Analysis of Ego Network Communities and Temporal a Affinity for Online Social Networks". In: *International Conference on Smart Objects and Technologies for Social Good*. Springer. 2016, pp. 135–144.

[247] H. Haas and A. Brown. *Web Services Glossary*. Tech. rep. https://www.w3.org/TR/ws-gloss/. Accessed: 29.7.2017. W3C, Feb. 2004.

[248] M. Richards. *Software Architecture Patterns*. O'Reilly Media, 2015. isbn: 9781491925409.

[249] A. Pathak, G. Rosca, V. Issarny, M. Decat, and B. Lagaisse. "Privacy and Access Control in Federated Social Networks". In: *Engineering Secure Future Internet Services and Systems*. Springer, 2014, pp. 160–179.

[250] R. Sandhu, D. Ferraiolo, and R. Kuhn. "The NIST Model for Role-Based Access Control: Towards a Unified Standard". In: *ACM Workshop on Role-based Access Control*. Vol. 2000. 2000, pp. 1–11.

[251] J. Martin. *Managing the Data Base Environment*. A James Martin book. Pearson Education, Limited, 1983. isbn: 9780135505823.

[252] R. T. Fielding. "Architectural Styles and the Design of Network-based Software Architectures". https://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf. Accessed: 22.7.2017. PhD thesis. University of California, Irvine, 2000.

[253] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*. Tech. rep. https://tools.ietf.org/html/rfc7231. Accessed: 4.8.2017. IETF, 2014.

[254] A. Tanenbaum. *Computer Networks*. 5th. Pearson, 2012. isbn: 978-3868941371.

[255] R. Daigneau. *Service Design Patterns: Fundamental Design Solutions for SOAP/WSDL and RESTful Web Services*. Addison-Wesley, 2011.

[256] J. E. White. *A High-Level Framework for Network-Based Resource Sharing (RFC 707)*. https://tools.ietf.org/html/rfc707. Accessed: 23.6.2017. 1976.

[257] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. Tech. rep. http://www.w3.org/TR/soap12-part1/. Accessed: 4.8.2017. W3C, 2007.

[258] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Tech. rep. http://www.w3.org/TR/xml/. Accessed: 31.7.2017. W3C, 2008.

[259] R. Fielding and J. Reschke. *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. Tech. rep. https://tools.ietf.org/html/rfc7230. Accessed: 4.8.2017. IETF, 2014.

[260] S. Tilkov, M. Eigenbrodt, S. Schreier, and O. Wolf. *REST und HTTP*. 3rd ed. dpunkt.verlag, 2015. isbn: 978-3-86490-120-1.

[261] B. Mulloy. *Web API Design - Crafting Interfaces that Developers Love*. Tech. rep. https://apigee.com/about/cp/api-design-best-practices. Accessed: 22.7.2017. Apigee, 2013.

[262] T. Espinha, A. Zaidman, and H.-G. Gross. "Web API Growing Pains: Loosely Coupled yet Strongly Tied". In: *Journal of Systems and Software* 100 (2015), pp. 27 –43. issn: 0164-1212. doi: http://dx.doi.org/10.1016/j.jss.2014.10.014.

[263] S. Faulkner, A. Eicholz, T. Leithead, A. Danilo, and S. Moon. *HTML 5.2*. Tech. rep. https://www.w3.org/TR/html52/. Accessed: 3.8.2017. W3C, Aug. 2017.

[264] S. McCarron and M. Ishikawa. *XHTML 1.1 - Module-based XHTML - Second Edition*. Tech. rep. http://www.w3.org/TR/xhtml11/. Accessed: 3.8.2017. W3C, 2010.

[265] K. Bals. *Extensible Stylesheet Language (XSL) Requirements Version 2.0*. Tech. rep. http://www.w3.org/TR/xslfo20-req/. Accessed: 3.8.2017. W3C, 2008.

[266] M. Kay. *XSL Transformations (XSLT) Version 3.0*. Tech. rep. https://www.w3.org/TR/xslt-30/. Accessed: 3.8.2017. W3C, 2017.

[267] J. Robie, M. Dyck, and J. Spiegel. *XML Path Language (XPath) 3.1*. Tech. rep. https://www.w3.org/TR/xpath-31/. Accessed: 3.8.2017. W3C, 2017.

[268] *ECMA-404: The JSON Data Interchange Format*. Tech. rep. https://www.ecma-international.org/publications/standards/Ecma-404.htm. Accessed: 9.7.2017. ECMA, 2013.

[269] T. Bray. *The JavaScript Object Notation (JSON) Data Interchange Format*. Tech. rep. https://buildbot.tools.ietf.org/html/rfc7158. Accessed: 9.7.2017. IETF, 2013.

[270] A. Wright and H. Andrews. *JSON Schema: A Media Type for Describing JSON Documents draft-wright-json-schema-01*. Tech. rep. http://json-schema.org/latest/json-schema-core.html. Accessed: 9.7.2017. IETF, 2017.

[271] T. Berners-Lee. *Linked Data*. July 27, 2006. url: https://www.w3.org/DesignIssues/LinkedData.html (visited on 08/01/2017).

[272] M. Sporny, D. Longley, G. Kellogg, M. Lanthaler, and N. Lindström. *JSON-LD: A JSON-based Serialization for Linked Data*. Tech. rep. https://www.w3.org/TR/json-ld/. Accessed: 1.6.2017. 2014.

[273] P. Siriwardena. "JWT, JWS, and JWE". In: *Advanced API Security: Securing APIs with OAuth 2.0, OpenID Connect, JWS, and JWE*. Berkeley, CA: Apress, 2014, pp. 201–220. isbn: 978-1-4302-6817-8.

[274] M. Jones, J. Bradley, and N. Sakimura. *JSON Web Signature (JWS)*. Tech. rep. http://tools.ietf.org/html/rfc7515. Accessed: 1.8.2017. IETF, 2015.

[275] M. Jones and J. Hildebrand. *JSON Web Encryption (JWE)*. Tech. rep. http://tools.ietf.org/html/rfc7516. Accessed: 1.8.2017. IETF, 2015.

[276] S. Perreault. *vCard Format Specification*. Tech. rep. https://tools.ietf.org/html/rfc6350. Accessed: 3.8.2017. IETF, 2011.

[277] P. Kewisch. *jCard: The JSON Format for vCard*. Tech. rep. https://tools.ietf.org/html/rfc7095. Accessed: 3.8.2017. IETF, 2014.

[278] S. Perreault. *xCard: vCard XML Representation*. Tech. rep. https://tools.ietf.org/html/rfc6351. Accessed: 3.8.2017. IETF, 2011.

[279] J. Smarr. *Portable Contacts 1.0 Draft C*. 2008.

[280] OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Tech. rep. https://www.w3.org/TR/owl2-overview/. Accessed: 1.8.2017. W3C, 2012.

[281] T. Çelik, M. Mullenweg, and E. Meyer. *Xhtml Friends Network*. Tech. rep. http://www.gmpg.org/xfn/11. Accessed: 13.7.2017. GMPG, 2003.

[282] Facebook. *The Open Graph Protocol*. http://ogp.me/. Accessed: 3.1.2017. 2010.

[283] *Dublin Core Metadata Element Set, Version 1.1*. Tech. rep. http://dublincore.org/documents/2006/12/18/dces/. Accessed: 2.8.2017. Dublin Core Metadata Initiative, 2006.

[284] B. Adida, M. Birbeck, S. McCarron, and I. Herman. *RDFa Core 1.1 - Third Edition. Syntax and Processing Rules for Embedding RDF Through Attributes*. Tech. rep. http://www.w3.org/TR/rdfa-core/. Accessed: 2.8.2017. W3C, 2015.

[285] Facebook Developers. *OpenGraph*. https://developers.facebook.com/docs/opengraph/. Accessed: 12.6.2017. 2013.

[286] J. Snell. *Atom Threading Extensions*. Tech. rep. https://tools.ietf.org/html/rfc4685. Accessed: 4.6.2017. IETF, 2006.

[287] P. Saint-Andre. *Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence*. http://tools.ietf.org/html/rfc6121. Accessed: 31.7.2017. 2011.

[288] P. Saint-Andre. *XEP-0045: Multi-User Chat*. Tech. rep. https://xmpp.org/extensions/xep-0045.html. Accessed: 1.8.2017. XMPP Standards Foundation, 2016.

[289] P. Millard, P. Saint-Andre, and I. Paterson. *XEP-0020: Feature Negotiation*. Tech. rep. https://xmpp.org/extensions/xep-0020.html. Accessed: 1.8.2017. XMPP Standards Foundation, 2006.

[290] P. Saint-Andre and J. Hildebrand. *XEP-0184: Message Delivery Receipts*. Tech. rep. https://xmpp.org/extensions/xep-0184.html. Accessed: 1.8.2017. XMPP Standards Foundation, 2011.

[291] *Matrix Specification*. Tech. rep. https://matrix.org/docs/spec/. Accessed: 1.8.2017. Matrix.org, 2017.

[292] *Matrix FAQ*. Tech. rep. https://matrix.org/docs/guides/faq.html. Accessed: 1.8.2017. Matrix.org, 2017.

[293] J. Genestoux, A. Parecki, B. Fitzpatrick, B. Slatkin, and M. Atkins. *WebSub*. Tech. rep. https://www.w3.org/TR/websub/. Accessed: 1.8.2017. W3C, 2017.

[294] M. N. Ko, G. P. Cheek, M. Shehab, and R. Sandhu. "Social-Networks Connect Services". In: *Computer* 43.8 (2010), pp. 37–43.

[295] Google. *Google+ API*. https://developers.google.com/+/web/api/rest/. Accessed: 17.6.2017. 2017.

[296] D. Hardt. *The OAuth 2.0 Authorization Framework*. Tech. rep. https://tools.ietf.org/html/rfc6749. Accessed: 23.6.2017. 2012.

[297] G. Papadakis, K. Tserpes, E. Sardis, M. Kardara, A. Papaoikonomou, and F. Aisopos. "Social Media Meta-API: Leveraging the Content of Social Networks". In: *Proceedings of the 21st International Conference on World Wide Web*. ACM. 2012, pp. 271–274.

[298] M. Mostarda, D. Palmisano, F. Zani, and S. Tripodi. "Towards an OpenID-Based Solution to the Social Network Interoperability Problem". In: *W3C Workshop on the Future of Social Networking*. 2009, pp. 15–16.

[299] G. Gouriten and P. Senellart. "API Blender: A Uniform Interface to Social Platform APIs". In: *International World Wide Web Conference (WWW)*. Lyon, France, 2012.

[300] J. Jonsson and B. Kaliski. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. Tech. rep. https://tools.ietf.org/html/rfc3447. Accessed: 4.6.2017. IETF, 2003.

[301] M. Bellare and P. Rogaway. "The Exact Security of Digital Signatures - How to Sign with RSA and Rabin". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, pp. 399–416.

[302] D. J. Bernstein. *RSA Signatures and Rabin-Williams Signatures: The State of the Art*. https://cr.yp.to/papers.html#rwsota. 2008.

[303] J. Postel and J. Reynolds. *File Transfer Protocol (FTP)*. Tech. rep. https://tools.ietf.org/html/rfc959. Accessed: 26.8.2017. IETF, 1985.

[304] P. Hethmon and R. Elz. *Feature Negotiation Mechanism for the File Transfer Protocol*. Tech. rep. https://tools.ietf.org/html/rfc2389. Accessed: 26.8.2017. IETF, 1998.

[305] E. Kohler, M. Handley, and S. Floyd. *Datagram Congestion Control Protocol (DCCP)*. Tech. rep. https://tools.ietf.org/html/rfc4340. Accessed: 26.8.2017. IETF, 2006.

[306] A. Paro. *ElasticSearch Cookbook*. Packt Publishing Ltd, 2015.

[307] L. Masinter. *The "data" URL Scheme*. Tech. rep. https://www.ietf.org/rfc/rfc2397.txt. Accessed: 21.7.2017. IETF, 1998.

[308] D. Calhoun. *When to Base64 Encode Images (and when not to)*. Aug. 28, 2011. url: http://davidbcalhoun.com/2011/when-to-base64-encode-images-and-when-not-to/ (visited on 07/21/2017).

[309] P. Deutsch. *GZIP File Format Specification Version 4.3*. Tech. rep. https://tools.ietf.org/html/rfc1952. Accessed: 21.7.2017. IETF, 1998.

[310] D. Longley, M. Sporny, S. McCarron, and C. Allen. *Linked Data Signatures 1.0*. Tech. rep. https://w3c-dvcg.github.io/ld-signatures/. Accessed: 10.10.2017. W3C, 2017.

[311] T. Bocek. *TomP2P - A Distributed Multi Map*. 2009.

[312] Jason Evans. *The HipHop Virtual Machine*. https://www.facebook.com/notes/facebook-engineering/the-hiphop-virtual-machine/10150415177928920/, Accessed: 16.1.2017. 2011.

[313] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. "Design Patterns: Abstraction and Reuse of Object-Oriented Design". In: *European Conference on Object-Oriented Programming*. Springer. 1993, pp. 406–431.

[314] S. M. H. Sharhan. "Design and Implementation of Sonic Compliant Online Social Network Platform for Home Servers". MA thesis. Technische Universität Berlin, 2017.

[315] S. Buchegger and A. Datta. "A Case for P2P Infrastructure for Social Networks - Opportunities & Challenges". In: *Sixth International Conference on Wireless On-Demand Network Systems and Services, 2009. WONS 2009.* IEEE. 2009, pp. 161–168.

[316] B. Carpenter. *Architectural Principles of the Internet*. Tech. rep. `https://tools.ietf.org/html/rfc1958`. Accessed: 23.8.2017. IETF, 1996.

[317] R. Braden. *Requirements for Internet Hosts - Communication Layers*. Tech. rep. `https://tools.ietf.org/html/rfc1122`. Accessed: 23.8.2017. IETF, 1989.

[318] *ISO/IEC 27000:2016: Information Technology — Security Techniques — Information Security Management Systems — Overview and Vocabulary*. Tech. rep. ISO/IEC, Feb. 2016.

[319] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021*. Tech. rep. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf`. Accessed: 28.8.2017. Cisco, Feb. 2017.

[320] R. Darwish and K. Ghazinour. "A Novel Approach for Studying Privacy Behavior in Social Media". In: *Proceedings of the 4th Annual Conference on Computational Science & Computational Intelligence (CSCI)*. ACSE. 2017.

[321] A. Koopmans. "Decentralized Social Networking Site". Bachelor Thesis. Universiteit van Amsterdam, 2015.

[322] D. Koll, D. Lechler, and X. Fu. "SocialGate: Managing Large-Scale Social Data on Home Gateways". In: *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. IEEE. 2017, pp. 1–6.

[323] H. Hebbo. "Integration of the Sonic Protocol into Existing Online Social Networks to Facilitate Seamless Inter-Platform Communication". MA thesis. Technische Universität Berlin, 2015.

[324] M. Beckmann. "Design and Implementation of a Protocol for Feature Negotiation between Platforms with Different Featuresets in the Sonic Online Social Network Federation". MA thesis. Technische Universität Berlin, 2016.