Towards a Robust and Secure Decentralized Online Social Network

Dissertation

zur Erlangung des Doktorgrades
Dr. rer. nat.
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

David Koll
aus Bad Hersfeld

Göttingen
im Oktober 2014

Betreuungsausschuss:

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen

Prüfungskommission:

Referent:

Prof. Dr. Xiaoming Fu,
Georg-August-Universität Göttingen

Korreferenten:

Prof. Dr. Dieter Hogrefe,
Georg-August-Universität Göttingen
Prof. Dr. Jun Li,
University of Oregon, Eugene, USA

Weitere Mitglieder
der Prüfungskommission:

Prof. Dr. Carsten Damm,
Georg-August-Universität Göttingen
Prof. Dr. Jens Grabowski,
Georg-August-Universität Göttingen
Prof. Dr. Konrad Rieck,
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 25. November 2014

Abstract

The virtually unlimited growth of popular *Online Social Network*s (OSNs) is often accompanied by severe violations of their users' privacy and intellectual property rights. Both problems are rooted in the centralized architecture of many current OSNs, in which a single entity (the *provider* of the network) controls all globally stored but unencrypted user data. As a result, the provider is able to analyze, forward, sell, modify, or otherwise misuse the data. However, even though these practices have raised serious privacy concerns among OSN users, many providers continue to collect and analyze evermore data and introduce diverse restrictions on their users, while showing little interest in changing their behavior.

On these grounds, *Decentralized Online Social Network*s (DOSNs) have attracted the attention of both researchers and users. A DOSN can function without a central provider and aims at allowing users to control access to their data by their own rules. However, by comprehensively reviewing state-of-the-art DOSNs this thesis shows that, although the need for a competitive DOSN is obvious, several challenges for DOSNs remain unsolved, including the construction of a robust, privacy-preserving communication and data storage infrastructure.

This thesis further emphasizes the prevalence of the Sybil attack in OSNs, in which an attacker orchestrates a large number of fake accounts for various malicious intents, including vote manipulation or distributing spam messages. A detailed study of state-of-the-art proposals to defend an OSN against this particular attack reveals that none of these solutions offers efficient detection or containment of the fake nodes. As a consequence, securing DOSNs against the Sybil attack emerges as another central challenge.

In tackling these major challenges, this thesis proposes a new, comprehensive DOSN. Dubbed SOUP, the SELF-ORGANIZED UNIVERSE OF PEOPLE—among other distinctive and valuable features—in particular offers functionality to build a robust and secure DOSN.

Its infrastructure is robust in the sense that SOUP effectively replaces the central OSN provider with a substrate built by the OSN participants themselves. The substrate, even though it does not rely on any permanently available resources, makes the encrypted data of *all* users highly available.

It is further secure in the sense that it effectively guards user data from being accessed by unauthorized parties, and properly functions in the presence of large amounts of malicious users. In particular, even if an attacker can compromise large fractions of the OSN by a Sybil attack, she cannot significantly adversely affect SOUP's operation.

Zusammenfassung

Schwere Verletzungen der Privatsphäre, des Rechtes auf informationelle Selbstbestimmung und des Urheberrechtes ihrer Nutzer begleiten das nahezu unbegrenzte Wachstum von *Online Social Network*s (OSNs). Die Wurzeln dieser erheblichen Probleme liegen in der zentralisierten Architektur der OSNs, durch die der Anbieter des Netzwerkes die Kontrolle über alle (unverschlüsselten) Nutzerdaten erhält. Dies resultiert oft in der Analyse, dem Verkauf oder anderweitigem Missbrauch dieser Daten, und führt zu einem schwerwiegenden Konflikt: Einerseits protestieren immer mehr Nutzer gegen derartige Vorgehensweisen, während die OSN-Anbieter ihrerseits keinen Willen zum Entgegenkommen erkennen lassen.

Als Konsequenz aus diesem Dilemma ist die Idee der *Decentralized Online Social Network*s (DOSNs) gewachsen und erfreut sich immer größerer Beliebtheit. In einem solchen dezentralen Netzwerk existiert der zentrale Anbieter nicht mehr. Vielmehr sind die Nutzer in der Lage, den Zugriff auf ihre nun verschlüsselten Daten sehr genau selbst zu kontrollieren. Eine im Rahmen dieser Dissertation durchgeführte Studie zeigt allerdings, dass bisherige DOSN-Ansätze eine große Anzahl an Problemen aufweisen, wie zum Beispiel das Fehlen einer robusten und effizienten Alternative zur Infrastruktur des Providers.

Ein weiteres bedeutendes Problem ist die geringe Widerstandsfähigkeit gegenüber Angriffen auf das DOSN, insbesondere gegen den sogenannten *Sybil*-Angriff, der in letzter Zeit vermehrt in OSNs aufgetreten ist. Es liegt zwar eine Anzahl an Verteidigungsmechanismen gegen diesen Angriff vor; in einer gründlichen Analyse derer zeigt diese Dissertation jedoch, dass diese Systeme ihre Verteidigungsfähigkeit unter Berücksichtigung realitätsnaher Annahmen verlieren und ein DOSN daher nicht vor Sybil-Angriffen schützen können.

Um diese Probleme zu lösen, wird in dieser Arbeit ein neues DOSN vorgestellt. Das SELF-ORGANIZED UNIVERSE OF PEOPLE (kurz: SOUP) verfügt neben weiteren problemlösenden Alleinstellungsmerkmalen vor allem über zwei Eigenschaften:

SOUP ist robust, indem es die zentrale Infrastruktur effektiv durch ein Substrat ersetzt, das von den Teilnehmern des OSNs selbst errichtet ist. Insbesondere sind die verschlüsselten Daten aller Nutzer hoch verfügbar.

SOUP ist außerdem sicher, indem es effektiv gegen Datenzugriff von unautorisierten Parteien schützt und gleichzeitig seine Funktionalität auch in der Gegenwart verschiedener Angriffe gewährleistet. Dazu zählt insbesondere der Sybil-Angriff, der auch bei einer weitreichenden Kompromittierung des OSNs keine signifikanten Auswirkungen auf dessen Performanz hat.

Acknowledgements

I would like to sincerely thank my supervisor Prof. Xiaoming Fu for his constant support, his courtesy to pursue my diverse research interests and the chances he allowed me to take in visiting great research laboratories on three different continents. His efforts and guidance made this thesis possible.

I am deeply grateful to Prof. Jun Li, who also kindly agreed to be my second thesis supervisor. Jun helped to continuously improve my work through constructive criticism and reviews in hours over hours of discussions in dozens of meetings. It was also a great pleasure to work with Jun in his lab at the University of Oregon during my stay there.

I would also like to express my gratitude to Prof. Dieter Hogrefe for being the third member of my thesis committee, and for providing insightful feedback during the progress review meetings within the PhD program.

I am obliged to Prof. Carsten Damm, Prof. Jens Grabowski and Prof. Konrad Rieck for being a member of my examination committee, and to Prof. Edith Ngai and Prof. Jin Zhao for hosting me during research visits at Universitet Uppsala and Fudan University, respectively.

I am grateful to my former and current colleagues at the Computer Networks Group at the University of Göttingen, especially Dr. Mayutan Arumaithurai, Jiachen Chen, and Dr. Niklas Neumann, whose feedback also contributed to the quality of this thesis. In particular, I would like to thank Dr. Florian Tegeler, who inspired me with the very first idea of this thesis during my Master studies, acted as my first contact person during my Master's thesis, and with whom I later published my very first research paper as colleagues.

Last but definitely not least, I want to thank my parents Klaus and Barbara Koll for their never-ending support. Without them this thesis would not have been written in the first place.

Contents

List of Figures

List of Tables

Acronyms

ABE *Attribute Based Encryption*

AES *Advanced Encryption Standard*

AMSK *ABE Master Secret Key*

API *Application Programming Interface*

APK *ABE Public Key*

AS *Access Structure*

ASK *Attribute Secret Key*

BFS *Breadth First Search*

CA *Certification Authority*

CDF *Cumulative Distribution Function*

CP-ABE *Ciphertext Policy Attribute Based Encryption*

CT *Ciphertext*

DDoS *Distributed Denial of Service*

DES *Data Encryption Standard*

DHT *Distributed Hash Table*

DoS *Denial of Service*

DOSN *Decentralized Online Social Network*

DTN *Delay Tolerant Networks*

ECC *Elliptic Curve Cryptography*

ECIES *Elliptic Curve Integrated Encryption Scheme*

ES *Experience Set*

IBE *Identity Based Encryption*

ID *Identifier*

IP *Internet Protocol*

KB *Knowledge Base*

KB-ABE *Key Based Attribute Based Encryption*

KDF *Key Derivation Function*

LCC *Largest Connected Component*

LFSR *Linear Feedback Shift Register*

MAC *Message Authentication Code*

OSN *Online Social Network*

PGP *Pretty Good Privacy*

PIR *Private Information Retrieval*

P2P *Peer-to-Peer*

RSA *Rivest-Shamir-Adleman*

RTT *Round-Trip-Time*

RW *Random Walk*

SD *Sybil Detection*

SLOC *Source Lines of Code*

SNS *Social Networking Sites*

SOUP *Self-Organized Universe of People*

ST *Sybil Tolerance*

TCP *Transport Control Protocol*

TCPK *Traditional Cryptography Public Key*

TOR *The Onion Router*

TTL *Time-To-Live*

VM *Virtual Machine*

XOR *Exclusive OR*

Chapter 1

Introduction

With every new product launch, it seemed Facebook would wait for the inevitable negative reaction on privacy, then announce minimal changes without fundamentally altering the new feature.

— The Washington Post, ”Mark Zuckerberg’s theory of privacy“ [1]

1.1 The Problem

*Online Social Network*s (OSNs) have evolved from small, themed networks into ubiquitous platforms of communication over the past few years. In July 2014, Facebook[1], once a small Harvard campus network and now the world's largest OSN, counted one billion interactions related to the FIFA Football Worldcup 2014 [2]. At the same time, Twitter[2] observed 672 million status updates (*tweets*) related to the tournament, and over 35 million tweets during a single match [3]. Regardless of such prominent events, YouTube[3] provides its members with the opportunity to rate, subscribe to and comment on hundreds of hours of new video uploaded every minute [4]. The video sharing platform is now the major contributor to European Internet traffic [5].

Concurrent to the explosion of content, OSNs user numbers are continuously growing. Twitter's 225 million users are surmounted by 1.32 billion users on Facebook, and almost one billion unique users visit YouTube every month [4, 6, 7]. In 2008, the Flickr photo sharing community[4] experienced a growth of 58% in just three months [8], while Twitter even reported an even more remarkable 1,400% growth rate for 2009 [9], and has been continuously growing ever since [10].

Due to their enormous reach, OSNs can even have an influence on politics. During the political uprisings of the Arab Spring of 2011, Twitter and other social media played a major role as both communication infrastructure and dissemination channels for the demonstrators [11–13]. In 2010, protest organizers used Twitter, YouTube, and Flickr as alternative platforms for reporting during the G-20 summit in Toronto [14].

For the same reason, OSNs have also become an effective way for content producers to reach their customers, and also influence the business model of enterprises [15]. For instance, media are currently using OSNs as one major way of distributing news [16], and sales teams exploit the opportunities of viral marketing over OSNs [17, 18].

Currently, the key OSNs are organized in a centralized fashion and usually controlled by global players in information technology. Whereas Facebook and Twitter took the step to stock-markets themselves in 2012 and 2013, respectively, YouTube was acquired by Google (which additionally operates its own OSN, Google+[5]) in 2006 [19], and Flickr is owned by Yahoo [20]. These players (or *providers*), caused by the exponential growth of OSNs, deal with tremendous amounts of user information. They can obtain deep insights into

[1] `http://www.facebook.com` (all URLs have been checked on December 30th 2014)
[2] `http://www.twitter.com`
[3] `http://www.youtube.com`
[4] `http://www.flickr.com`
[5] `http://plus.google.com`

their users' personal interests, opinions, social relationships, and economical or political preferences, a situation that has raised serious privacy and security concerns [21,22]. As an example, Facebook already controls the private data of one-sixth of the worlds population.[6] Still, it is striving to obtain more user data, as demonstrated by the multi-billion dollar acquisitions of the Instagram and WhatsApp user bases in 2012 and 2014, respectively [23, 24]. With both deals Facebook obtained photos and messaging data for almost 500 million users, which were either unknown to the company before or complemented its view on the data of already-known users.

The providers may exploit user data stored at their premises for various purposes, including the resale of potentially private data or their analysis for commercial use, without notifying users [25]. This practice has already led to several class action lawsuits against OSN providers [26], without however changing their perspective towards user data privacy [1,27]. In fact, providers not only collect the data for their individual purposes; many OSNs, including Facebook, Google and Yahoo, granted full access to user data to the United States government through the PRISM program [28].

While the mere aggregation of huge amounts of data at a single instance is thus alarming in itself, OSN users often are additionally at the mercy of the OSN provider with regards to the OSNs' terms of use, which often compromise the users' data privacy and property rights [29]. For instance, Facebook and Google+ have forced their members into using their real names as user names, threatening to delete the accounts of those who would not follow that directive [30].

Moreover, many providers leave users helpless when changing, forwarding, or misusing their data [31]. In 2014, Facebook changed the location of 20 million Instagram photos from their original datacenter into Facebook's own without notifying the users [32]. Between 2007 and 2009, the Facebook Beacon application forwarded sensitive shopping information of users between Facebook and a group of partners (e.g., Amazon) without the users' consent. Beacon was only stopped in the course of a class-action lawsuit, which cost Facebook 9.5 million US dollars to settle [25]. In June 2012, an incident at LinkedIn[7] demonstrated that a central storage of private user data is also subject to external misuse, when millions of passwords were leaked from its central repository [33].

At the same time, there is little to no activity by providers to permanently fix or even improve the situation of user privacy, even though a fix might not be much of a technical challenge. A large step towards comprehensive security and privacy means to their users could be taken by, for instance, encrypting user data and letting users decide with whom they want to share what parts of their data. The major providers' conduct of not following

[6] According to `http://www.census.gov/popclock/`.
[7] `http://www.linkedin.com`

that path is however more than understandable from an economic perspective, as doing so would result in giving up the ability to analyze and sell user data and thus the loss of their main source of income [34, 35].

Consequently, the concept of *Decentralized Online Social Network*s (DOSNs) has attracted researchers and practitioners from academia and industry. The main idea of DOSNs is to build an OSN without any participation from a central provider, and thus to enable better user data security and privacy. Due to the significance of the problem, a plethora of DOSN solutions has been proposed recently [36–48]. These systems greatly differ in their approach to replace the centralized infrastructure. Whereas some DOSNs try to utilize permanently available resources—in particular storage space and processing power [36–39]—other systems relax the dependency on such resources, and let nodes cooperate with each other [40–46]. Finally, some researchers build hybrid solutions that incorporate both permanently available capacities and node cooperation [47, 48].

However, each of these solutions introduces new shortcomings, including (i) limited success in providing high availability for user data [40–43, 45, 46, 48]; (ii) a discrimination of some users based on their dependency on other nodes [40, 43, 45–48]; (iii) a dependency of all users on powerful nodes [37–39, 47, 48]; (iv) high communication or storage overhead [43–46]; (v) a low adaptivity to the user dynamics typically present in OSNs [36, 37, 41–48]; (vi) susceptibility to malicious users [37, 40–48]; (vii) a lack of data encryption and thus weakened user privacy [37, 41, 42, 48]; (viii) lack of non-consideration of mobile users [40–48]; and (ix) technical feasibility and economic deployability issues [36, 38, 39, 46, 47].

While most approaches suffer from a multitude of drawbacks, each one of these shortcomings can prevent the successful establishment of a competitive DOSN. For instance, a DOSN which does not offer a high availability of user data is unlikely to persuade a critical mass of users to join the network because the user experience will be worse than in current centralized OSNs. At the same time, if data availability comes with a usage fee, users are unlikely to join the network as well since market leading OSNs are free of charge.

Among all deficiencies, one critical drawback is susceptibility to malicious users, against which state-of-the-art DOSN solutions are not inherently protected. In fact, recent research has uncovered the existence of large numbers of malicious accounts in OSNs [49–52], which could deteriorate the performance of any unprotected DOSNs. Currently, most of the malicious accounts are used as part of a *Sybil attack* [53]. The term Sybil attack describes the creation of a multitude of fake accounts (hence the name *Sybil*) for eclectic malicious intents. Whereas some attackers try to distribute spam messages with the created Sybils [49, 54], others aim at manipulating recommendation or voting schemes by outvoting regular users [51, 55]. Fake account creation itself is now a multi-million dollar business in the underground economy, where attackers can easily buy a large number of Sybil accounts

for little money [50]. As a consequence, millions of Sybils have been observed in real-world OSNs [49, 51], which has lead researchers to try to develop automated algorithms to detect and exclude Sybils from these networks [56–63]. If these algorithms should prove to be efficient, currently proposed DOSNs could rely on them to ward off Sybil attacks on the network.

However, recent research has identified a rich set of behaviors of both attackers and honest users that calls these defenses into question [51, 52, 54, 64, 65], and it is uncertain how well they perform with regards to these behaviors. While it is critical to have effective Sybil defense solutions, it is therefore unclear what help and how much help can be obtained from existent solutions, and to what degree a DOSN, taken by itself, must be secured against Sybils.

Therefore, in essence, there exists an obvious need for decentralized online social networking, and potential users of DOSNs are currently faced with a plethora of approaches to choose from. These approaches greatly differ in their architectures and measures to replace central OSN providers, However, none of these solutions constitutes a comprehensive DOSN that is able to compete with current centralized OSNs. One particular imperfection is the lack of resiliency against malicious users, which have vigorously infiltrated OSNs in recent years. At the same time, it is unclear whether or not existing solutions to defend against malicious users could be of help when designing a new, better DOSN that is competitive with centralized OSNs.

1.2 Thesis Contributions

In this thesis, the problems stated above are addressed through the following contributions:

- To provide a clear overview of the state-of-the-art of DOSNs, a comprehensive study of existing decentralized online social networking approaches is conducted. The study reveals the absence of a full-fledged DOSN. The lessons learned from this study are applied to investigate the challenges that a new, better DOSN would face.
- Particular attention is paid to the most predominant attacks in OSNs, specifically the Sybil attack. To clarify whether or not existing techniques can help to prevent Sybil attacks, existing Sybil defenses based on OSNs are analyzed in detail. The result of the in-depth analysis suggests that currently no efficient Sybil defense exists, and that DOSNs hence need to be resilient to Sybils in their design.
- The main contribution of this thesis then is the design, implementation, and evaluation of the SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP), a novel DOSN, which addresses all the challenges emerging from the previous parts of this thesis.

1.2.1 A Comprehensive Review of DOSNs

As the first contribution, before starting to produce yet another DOSN, state-of-the-art solutions are investigated with regards to their advantages and drawbacks. The goal of the study is to hand both researchers and end users a clear overview of the specific characteristics and features of each DOSN approach. Therefore, a clear categorization into both solution classes and the particular functionality offered by each system is provided. Based on this classification, an analysis of which kind of approach would be the best design choice for a novel DOSN is conducted, followed by an investigation of the challenges this approach has to overcome.

1.2.2 An Analysis of Sybil Defenses Based on OSNs

As the second contribution, existent Sybil defense approaches are systematically analyzed, measured, and compared to find out whether or not any of these existing defense schemes can be applied to a novel DOSN, and to evaluate how well or inadequately they perform. The goal is to qualify and quantify the strengths and weaknesses of these approaches.

Two classes of Sybil defense approaches are investigated in detail: Sybil detection approaches—which try to detect Sybil nodes and exclude them from participation in a target system, and Sybil tolerance approaches—which try to limit the impact of Sybils present in the system. The former includes SybilGuard/SybilLimit [56, 66], SybilShield [63], SybilInfer [59], SybilDefender [62], GateKeeper [58], and SybilRank [57]. The latter includes Ostra [8] and SumUp [60].

Given that the Sybil defenses will face a *modern scenario*, in which a Sybil node may utilize more attack edges than traditionally assumed, this thesis' analysis pays particular attention to what a Sybil node has to achieve in order to make itself indistinguishable from honest nodes—and thereby disguise itself from the defense scheme. Different attack strategies are investigated where applicable, and for every Sybil defense solution the cost for the attacker (e.g., the number of attack edges to create) to thwart the solution is quantified.

The main finding is that current OSN-based Sybil defense approaches of both classes have difficulty identifying attack edges and Sybil nodes in the modern scenario. Surprisingly little effort is needed to deceive any existent defense scheme. Specifically, in many schemes a Sybil node only needs to create one or two attack edges to random honest nodes in order to successfully masquerade as a benign node. The attacker can further reduce the required effort if she follows more intelligent attack strategies that exploit particular weaknesses in a given defense scheme. As a consequence, when designing a new DOSN, the system needs to be designed in a way that is resilient to Sybils in the system.

1.2.3 SOUP: An Online Social Network By The People, For The People

As the third and main contribution, a novel DOSN, the SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP) is presented. Based on the analysis of related work and the Sybil defense analysis SOUP solves the drawbacks of related works and, therefore, addresses the following challenges:

(i) To achieve high data availability, SOUP proposes a new, generic approach for storing user data in a DOSN. Built on a robust, secure, and scalable mechanism, the approach mirrors a user's data at intelligently selected other OSN participants. Despite conservative assumptions on the availability of resources, the performance in terms of data availability is close to that of a centralized solution.

(ii) To not discriminate against any OSN user, SOUP ensures that regardless of participants' social relations or online probabilities, data for *all* participants is highly available.

(iii) To remove any dependencies on powerful nodes, it does not rely on permanently available or altruistically provided storage, although it can make an opportunistic use of such resources as they become available.

(iv) To limit overhead, it makes sure that there exist only as many replicas as required, and keeps the set of mirrors stable to avoid unnecessary user data retransmissions.

(v) To achieve reliability, SOUP is designed to be adaptive to the dynamics often seen in a DOSN—such as frequent node joining and departure—and it can quickly respond to changes in the system and continue to provide high performance.

(vi) To offer resiliency, SOUP provides the means to efficiently defend itself against malicious OSN users executing both Sybil or *Denial of Service* (DoS) attacks, as it can tolerate up to half of the identities in the OSN being controlled by an adversary.

(vii) To grant data privacy, SOUP offers effective mechanisms for encrypting data and ensures only eligible users can access data.

(viii) To support mobile users, SOUP is designed to minimize data transfer and resource consumption on mobile nodes.

(ix) Finally, to demonstrate its feasibility, extensive simulation experiments with three different large-scale real-world datasets are conducted, and SOUP is shown to meet all aforementioned challenges. Further, SOUP is successfully implemented and deployed on both desktop and mobile platforms.

1.3 Thesis Overview

The remainder of this thesis is organized as follows: in Chapter 2, the basic concepts of (decentralized) online social networking are explained. To further show the importance of developing DOSNs, a case for decentralization of social networks is presented in Chapter 3. A comprehensive review of related work on DOSNs is conducted in Chapter 4. The review uncovers the need for a novel DOSN solution, for which challenges emerge in Chapter 5. As one of the challenges is the resiliency against the Sybil attack, a detailed study of existing Sybil defense solutions follows in Chapter 6.

Based on the findings of all previous chapters, Chapter 7 then introduces the SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP). Chapter 8 is entirely devoted to a critical component of SOUP: the mirror selection, which ensures that SOUP can in fact constitute a robust and secure DOSN. SOUP is then extensively evaluated based on a large-scale simulation in Chapter 9. Afterwards, the implementation of SOUP is described in Chapter 10. In Chapter 11, a deployment of SOUP and an analysis of the obtained data is presented. In discussing several aspects related to SOUP, Chapter 12 paves the way for future work, and the thesis is summarized and concluded in Chapter 13.

Chapter 2

Background

*Online Social Network*s (OSNs) are present in hundreds of millions of people's everyday lives, but their characteristics often remain only vaguely defined. This chapter starts with providing the theoretical background of these networks. In particular, OSNs typically pivot on their *social graph*, a construct that contains information about the relations between all the users in the network. The social graph is thus the principal topic of the first part of this chapter.

Afterwards, the focus is switched to more practical issues, and the basics of techniques required to build a DOSN are discussed. These include widely accepted approaches that can be of help to replace the centralized provider, as well as different cryptographic approaches to secure user data in OSNs from unauthorized access.

Contents

2.1 Online Social Networks

In a scientific context, the first use of the term social network is attributed to the anthropologist John Arundel Barnes in 1954 [67, 68]. Ever since, the interest of the scientific community to investigate social networks has been growing considerably across disciplines, with a clear manifestation in high impact research in the 1980s [67, 69, 70].

Kähler gives a basic definition of a social network as "the network of—usually social—relationships that can be observed between a defined set of single units—usually individuals" in his 1975 literature survey [71]. Approximately twenty years later, in 1994, Wasserman follows a similar path:

Definition 2.1 (Social Network) *A social network consists of a finite set or sets of actors and the relation or relations defined on them. An actor is a discrete individual, corporate, or collective social unit. A relational tie links two actors to each other. [67]*

In this classical social network perspective that is coined from a sociological perspective, a linking of actors can be established by, e.g., talking to each other or by being biologically related.

With the rise of the Internet to an ubiquitous platform of communication, the concept of social networks has been carried online in the last decade. While *Friends Reunited*, the world's first *Online Social Network* (OSN), was founded in 1999 before the millenium turned, *Friendster* was the first OSN to accumulate a large-scale user base after its opening in 2003. Rather than being confined to actual human interactions, actors in OSNs are linked over the Internet and therefore communicate online, which allows them to interact without spatial and temporal constraints. Ellison defines *Social Networking Sites* (SNS), a synonym of OSNs, as follows:

Definition 2.2 (Social Networking Sites) *We define social network sites as web-based services that allow 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. The nature and nomenclature of these connections may vary from site to site. [72]*

Hence, with regards to Definition 2.1, in *online* social networks the following interpretations apply:

- *Actor*: The actor is usually represented by a user in the OSN. The user herself is an abstraction of the original definition of the actor, i.e., of a discrete individual, corporate or collective social unit.
- *Relational Tie*: A tie is represented by an actual connection between users in the OSN. These links can be unidirectional or bidirectional and can represent a variety of social relationships. The concrete interpretation of the links between users differs among the OSNs.

An abundance of OSN services implementing this concept has been developed in the past decade, a trend which has attracted the interest of computer scientists for various reasons. To begin with, the sheer scale of OSNs results in an enormous reach of these networks. Arguably the most famous OSN is Facebook, with currently approximately 1.3 billion users [7]. By providing a scalable infrastructure to enable users to communicate with each other, OSNs are continuously growing. Twitter has reached 271 million monthly active users in August 2014, which is a remarkable growth compared to its 100 million monthly active users in 2011 [6]. These users submit 500 million *tweets* (short text messages) every day [6]. During the 2014 FIFA Football Worldcup, over 35 million tweets were posted during a single match, and 672 million tweets related to the tournament were submitted during the worldcup in total [6].

Caused by their scale, OSNs further deal with enormous amounts of data. Whereas tweets originally consisted of 140 text characters only and Twitter has just recently opened up for multimedia content, Facebook has always allowed its users to upload photos and videos, *like* content, and *comment* on (multimedia) items. Altogether, users on Facebook contributed to one billion different interactions during the first two weeks of the FIFA Football Worldcup [7]. Other OSNs such as the Flickr photo sharing community or the video portal YouTube, concentrate on one particular type of content. Nonetheless, they are growing fast as well. While the Flickr OSN grew 58% in just three months in 2008 [8], YouTube is now the dominant contributor to European Internet traffic [5], as users view six billion hours of video on the platform every month [4].

Not all of this data is generated by users in the narrower sense, as media and other content producers have discovered OSNs as one major way of distributing content based on the ability of OSNs to efficiently propagate information to a large number of users [16, 73]. For the same reason, companies use the opportunities of viral marketing over OSNs to increase the perception of their products [18]. At the same time, OSNs can even have influence on economics or politics, as shown during the Arab Spring of 2011, where Twitter and other social media were used as both communication infrastructure and dissemination channel by

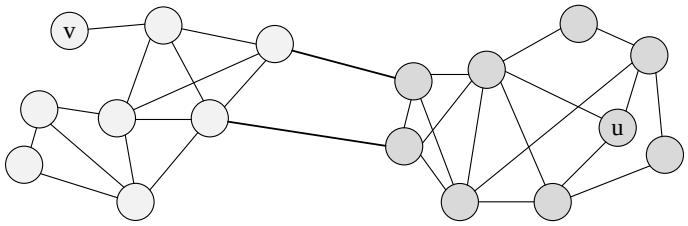


Figure 2.1: An exemplary undirected and unweighted social graph $G = (V, E)$.

the protesters [11–13].

Given the large scale, the huge amount of data to analyze, and the impact of OSNs on content propagation, researchers have thus begun to investigate and exploit these networks and to build new applications and infrastructures for them. Their research ranges from the analysis of interactions between users [74, 75], to exploiting the links between users [76], and building novel datacenter networks for large-scale OSNs [77–79].

2.1.1 The Concept of the Social Graph

Research investigating OSNs often focuses on the *social graph* of the network's users, a core structure of each OSN:

Definition 2.3 (Social Graph) *Denoted as $G = (V, E)$, the social graph of a network gives a representation of the set of users (V, the nodes in the graph) and links (E, the edges in the graph) between the users in that network. [80]*

Depending on the interpretation of the nodes in V and the links in E, the social graph yields a structural representation of the OSN. The most widespread interpretation of V and E is to let each $v \in V$ represent a user and each $e(v,w) \in E$ represent a *friendship* between v and w (i.e., implying v is *friends* with w). On Facebook, each of these friendship links is treated as mutual, which leads to an undirected social graph G which contains all friendships on Facebook. A small example of such an interpretation is shown in Figure 2.1, where each link in G is undirected.

A different interpretation, as for instance found in Twitter, where each node $v \in V$ represents a Twitter account, is that each edge $e(v,w) \in E$ represents a *follower-followee* (or

subscriber-subscribee) relation between the nodes v and w. In particular, such an edge depicts *v following w*, and is thus a directed edge. As a consequence, G is a directed graph.

For the remainder of this thesis, the terms edge, link, (social) relation, connection, or tie are interchanged when discussing an edge between two nodes. When considering undirected social graphs, the term friendship is also used in the same context.

In both directed and undirected graphs most social networks use a binary model when constructing the graph. That is, an edge between two nodes v and w either exists (then $e(v,w) = 1$), or does not (then $e(v,w) = 0$). For instance, two users on Facebook are assumed to be *friends* as soon as there is a link between them. However, such a model does not accurately represent a social network, as each tie can be of different strength [70]. Gilbert et al. found that the mere existence of an edge only contributes with approximately 4.5% to the actual strength of the tie [81]. Hence, there have been efforts to improve the social graph so that it models the strength of the ties more accurately. These models consider, e.g., the interaction frequency between nodes [74] or the intimacy of the words used in a conversation [81]. As a result, each edge $e(v,w) \in E$ can also have a weight, which describes the strength of the tie between the actors behind the users v and w.

The number of edges with which a node v is connected to other nodes is the *degree* of a node.

Definition 2.4 (Node Degree) *The degree $d(v)$ of a node v is the number of nodes in V adjacent to v in G. [80]*

The degree of node u in Figure 2.1, for instance, is 3. Various studies of OSNs have found that node degrees in most of these networks follow a power-law distribution [74, 82]. That is, most of the nodes have a relatively low node degree and only few nodes are very well connected and thus have a very high degree.

Recently, some researchers have pointed out that power-law distributions might not be the perfect fit for some OSNs as they *overestimate* the number of high-degree nodes in the network [83, 84]. Also OSNs have taken action to encourage their users to establish a certain number of links, so that the number of extremely poor connected users might not fit a power-law distribution as well [83, 84]. At the same time, other studies report that power-law distributions *underestimate* the degree of the high degree nodes [16]. For this thesis, the following facts confirmed by all kinds of studies are important: (i) most users in OSNs have a low or moderate node degree; and (ii) only few nodes with high degree exist.

Nevertheless, social graphs are well-connected. To measure the connectedness of a graph, usually the *Largest Connected Component* (LCC) (sometimes also giant component [85]) is considered.

Definition 2.5 (Largest Connected Component) *A connected component is a set of nodes for which each pair of nodes are connected by at least one path through the network. The LCC is the largest of these components. [83]*

In Figure 2.1 the LCC comprises all nodes in the graph, since every pair of nodes is connected by at least one path through the graph. Here, the node with the lowest degree, v ($d(v) = 1$), is still reachable by all other nodes. On a larger scale, a study of the complete 700-million-user Facebook social graph of 2011 found that 99.9% of the network belong to the LCC [83]. Hence, the graph is almost connected, with only a tiny fraction of the nodes in the network not being able to reach every other node by traversing edges in the graph. These nodes are often called *singletons*, i.e., users not participating in the OSN at all [85].

2.1.1.1 Communities and their detection

In addition to being well-connected, directed or undirected, and weighted or unweighted, OSN graphs have further structural properties. In particular, they typically contain *communities* [86–88].

Definition 2.6 (Community) *Communities are subsets of nodes in G within which edges between nodes are dense, but between which edges are less dense. [86]*

In Figure 2.1, there exist two communities—the membership of a node to a community is given by its coloring—in which nodes are highly interconnected but between which there exist few edges only. In real-world social networks, these communities can also be hierarchically composed [87, 89], where one larger community can contain several smaller communities.

In fact, most social networks are comprised of a large number of small communities. Averaged over different kinds of OSNs, 40% of the nodes belong to communities that are connected to the core of the network by a single edge, leading to a graph structure that can be best visualized by the term ”Octopus“ [89]. This phenomenon is also referred to as the high *modularity* of the OSN graph.

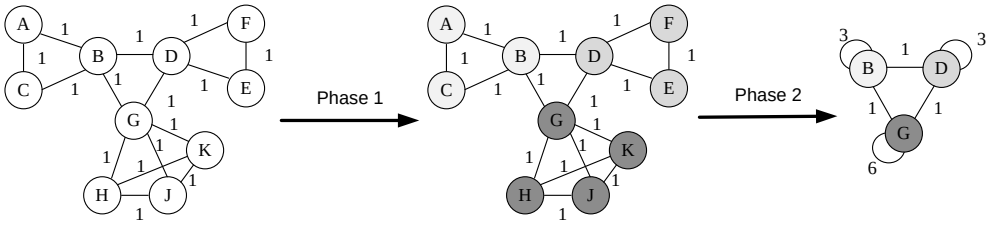


Figure 2.2: An exemplary execution of the Louvain method. Initially, for each node, a community containing only that node is created, and the weight of the links between all nodes is 1 (left). Then, in phase one, for each node the modularity gain is calculated, which results in three communities (center). In phase two, the nodes are aggregated into their communities (right), and each community is assigned the weight of the links between the community members. In this case, a second iteration yields no modularity gain, and the algorithm terminates.

Definition 2.7 (Modularity) *The modularity of a graph is defined by the fraction of the edges that fall within the communities minus the expected such fraction if edges were distributed at random. [90]*

That is, for a given division of the network's vertices into communities, modularity reflects the concentration of edges within communities compared with a random distribution of the same amount of links between all nodes in the graph. Therefore, a graph with densely connected communities, which are only sparsely connected among each other, will obtain a high modularity score.

The discovery of the modularity of social graphs has also led to an abundance of proposals to detect the communities in social networks, of which two of the most important approaches are the Girvan-Newman algorithm [86] and the Louvain method [87]. Due to its efficiency for even very large networks (the method runs in the complexity class $O(n \log n)$), the ability to detect hierarchical communities, and readily available implementations[8], the Louvain method is applied where necessary in this thesis.

An example of applying the Louvain method to a social graph is depicted in Figure 2.2. The method distinguishes between two phases:

- It starts by creating a community for each node in G, i.e., with $|V|$ communities. At the same time, based on the edges between nodes in G, a *weighted graph* is constructed, such that for each neighbor v of a node u, a link between the communities

[8] `http://perso.uclouvain.be/vincent.blondel/research/louvain.html`

representing u and v is established with weight 1. It then continues by calculating, for each node u and all its neighbors $N = v_1, v_2, ..., v_{d_u}$, the modularity gain obtained by the removal of u from its own community, while adding it to the community of a neighbor v_i. It then places u into the community that yields the highest gain or leaves u in its own community if there is no further gain. This procedure is executed until no further gain is possible and a local modularity maximization is reached.

- In the second phase, a new graph is created based on the communities found in the first phase. In this graph, the new nodes represent the communities found in the first step. The weights between these nodes are given by the sum of the weight of the links between nodes in the corresponding two communities. The method then executes the first phase (starting from calculating the modularity gain) again.

An iterative execution of this method will yield a hierarchy of the determined communities, or in other words communities of communities. One widely used metric to determine the quality of a community (i.e., the ratio of edges within the community to those edges to nodes outside of the community) is *conductance*.

Definition 2.8 (Conductance) *For a social graph $G = (V,E)$, let $S \subset V$ be a set of nodes in G with $|S| \leq \frac{1}{2}|V|$. Further, let $v = \sum_{i=1}^{|S|} d_i$ be the sum of the node degrees in S, and s the number of edges with one endpoint in S and one endpoint outside of S. Let all nodes outside of S be denoted as $\bar{S}$. The conductance of S is then defined as $\phi = s/v$. [89]*

In other words, the conductance describes a measure of the goodness of the distinction between a community S and the rest of the graph, $\bar{S}$. A small conductance usually hints at a stronger community, as then the number of edges pointing towards nodes outside of the determined community is, compared with the degree of all community members, low. Hence, the community is ought to be well connected internally, while there are only few links to other nodes.

2.1.1.2 Random Walk

Another approach to approximate communities is to execute several *Random Walk*s (RWs) on a social graph (for details, see Chapter 6). The concept of the RW was first described by Pearson in 1905 as a mathematical formalization of a path that consists of a succession of random steps [91]. RWs are used in many fields, including biology, chemistry, medicine, and first and foremost in many models in mathematics, physics and computer science [92,93]. As a consequence, they have gained significant attention from the research community [92–94].

RWs can, for instance, be conducted on a line, in the plane, in higher dimensions—or on graphs. The most simple definition of a RW is as follows:

Definition 2.9 (Random Walk) *Let $U = (U_1, U_2, ...)$ be a sequence of independent random variables with values in $\mathbb{R}^d$. Then, the stochastic process defined by*

$$X_n = X_0 + \sum_{i=1}^{n} U_i \quad n \in \mathbb{N}_0 \tag{2.1.1}$$

is a d-dimensional random walk.

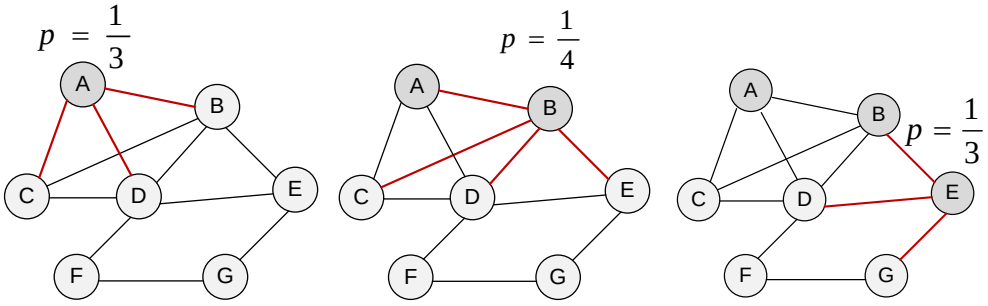
In a simple random walk on a line, each $U_i \in U$ takes the value 1 with probability $p \in [0,1]$ and -1 with probability $1-p$, respectively. One often used example to visualize the RW is the *Drunkard's Walk*, as introduced by Pearson in his 1905 article [91]: "A man starts from a point O and walks l yards in a straight line; he then turns through any angle whatever and walks another l yards in a second straight line. He repeats this process n times." In other words, at each step, the RW process chooses randomly from the options to continue available. In the case of a simple random walk on a line, the drunkard will thus step forward ($U_i = 1$) with probability p and backwards ($U_i = -1$) with probability $1-p$, respectively.

More generally, a random walk is defined by a transition function, which describes for each pair of points (x,y) the probability of transitioning from x to y. In this thesis, the random walk on a social graph $G = (V,E)$ is considered. Here, the transition function $P(x,y)$ describes the probability that the random walk transitions from a node $x \in V$ to a node $y \in V$ over the edge $(x,y) \in E$. Hence

$$0 \leq P(x,y) \leq 1, \ \sum_{i=1}^{d(x)} P(x,i) = 1 \tag{2.1.2}$$

where $d(x)$ is the degree of x. In other words, in a random walk on a graph—as executed exemplary in Figure 2.3—originating from a chosen starting point x, a neighbor y of x is chosen randomly and the walk moves to y. At y, a neighbor z of y is chosen randomly again, the walk moves to z, and so on.

More formally, such a random walk on a graph is a finite Markov chain that is time-reversible [93]. In short, this characteristic implies that the previous states of the walk are irrelevant in predicting the probability of subsequent states (for details about Markov chains and their properties, see [95]).



(a) Start at Node A (b) Second Step at B (c) Third Step at E

Figure 2.3: The first steps of a random walk on a graph. The walk starts at node A with $d(A) = 3$ in (a). Hence, each edge originating at A is traversed with probability $p = \frac{1}{3}$. In this example, the random walk chooses $e(A,B)$. In (b), the probability for the traversal of each edge originating from B is $p = \frac{1}{4}$, as $d(B) = 4$. The random walks selects $e(B,E)$. Then, analogously to previous steps, $p = \frac{1}{3}$ for each edge originating at E, and so on.

For a connected graph $G = (V,E)$ with n nodes and m edges, the random walk starts at a node v_0. At the t-th step, the walk is at node v_t and moves to a neighbor of v_t with probability $1/d(v_t)$. Here, the sequence of random nodes $(v_t : t = 0,1,...)$ is a Markov chain. The starting point v_0 can be fixed or be drawn from an initial distribution P_0. Lovasz [93] gives notations of P_T, the distribution of v_t as $P_t(i) = Prob(v_t = i)$, and the transition matrix of the Markov chain as $M = (p_{ij})$ for $i, j \in V$, such that

$$p_{ij} = \begin{cases} 1/d(i), & \text{if} \quad ij \in E, \\ 0, & otherwise \end{cases} \tag{2.1.3}$$

If A_G denotes the adjacency matrix of G and D the diagonal matrix with $D_{ii} = 1/d(i)$, then Lovasz states that $M = DA_G$, and if G is d-regular, then $M = (1/d)A_G$. As a consequence $P_t = (M^T)^t P_0$.

In other words, the probability p_{ij}^t of starting at i to reach j in t steps can be taken from the entry (i, j) of M^t. In the case of G being regular (i.e., every node has the same degree d), the Markov chain is also *symmetric*, meaning that a transition from i to j has the same probability as a transition from j to i.

In OSNs, regular graphs are unlikely to exist. For non-regular graphs, the Markov chain is instead *time-reversible*, meaning that a random walk in the backward direction is also a random walk [93]. If all random walks $(v_0, ..., v_t)$ with v_0 drawn from an initial distribution

P_0 are considered, a probability distribution P_t on v_t is obtained. Additionally, there is also a probability distribution Q on all sequences $(v_0, ..., v_t)$, and if each of these sequences is reversed, the resulting probability distribution Q'—if the chain is time-reversible—is the same as the distribution obtained by observing random walks originating from the distribution P_t.

The—for this thesis—most important probability distribution is the *stationary* distribution (sometimes also called *steady-state* distribution [96]).

Definition 2.10 (Stationary Distribution) *A distribution P_0 is called stationary for G if $P_t = P_0$ for all $t > 0$.*

Or, in a different notation, given a transition matrix M, the stationary distribution is a probability distribution π such that $\pi = \pi \cdot M$ [95]. In other words, in the long run, regardless the starting state, the proportion of time the Markov chain spends at a node v converges to π_v (the starting state is *forgotten*). Closely related to the stationary distribution is the *mixing time* of a (social) graph.

Definition 2.11 (Mixing Time) *The mixing time of a graph G indicates how fast a random walk approaches the stationary distribution. A slow mixing time means that a random walk needs to be long in order to reach the stationary distribution. [80, 97]*

Social graphs with a low mixing time often also offer a small *minimal cut*.

Definition 2.12 (Minimal Cut) *A minimal cut of a graph is a cut whose cutset (i.e., the set of edges which have to be removed to partition the graph) has the lowest number of edges among all cutsets. [97]*

For instance, the minimal cut of the graph depicted at the beginning of this Chapter in Figure 2.1 consists of the two edges connecting the left community with the right community. Note that the minimal cut is closely related to a low conductance value.

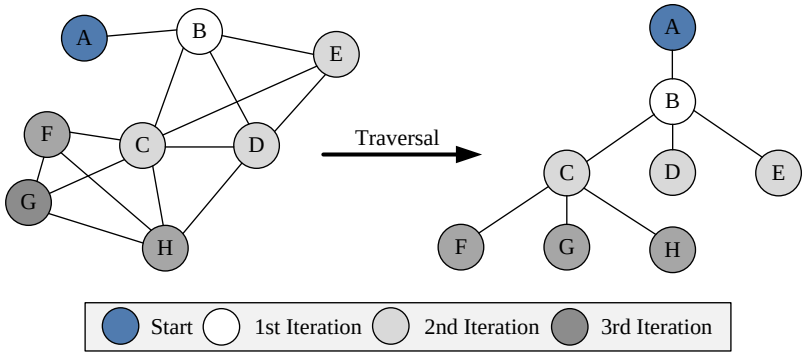


Figure 2.4: An exemplary graph traversed by BFS, starting at A. The first iteration explores only B. Moving on from B, BFS explores C, D, and E in the second iteration, and finally F, G, and H in the third iteration, upon which the algorithm terminates.

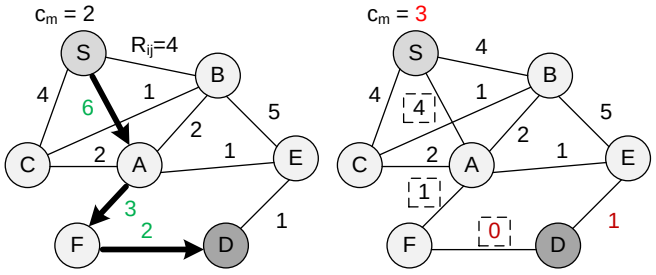
2.1.1.3 Breadth First Search

Besides a random walk, multiple algorithms can be applied to traverse a (social) graph, to, for instance, sample the graph [8, 75]. Among them, *Breadth First Search* (BFS) is one of the most-widely used algorithms. It starts from a selected node and progressively explores all neighbors [98]. Then, in each new iteration the unvisited nodes are selected in order of their exploration. As a consequence, BFS ultimately discovers all nodes that are connected to the starting node (recall that OSN graphs tend to be connected). Figure 2.4 shows a small social graph (on the left), and the order in which the nodes would be traversed by BFS (on the right).

The algorithm runs in $O(|V| + |E|)$ in the worst case, which may be reduced to $O(|V|)$ for many real-world applications [98].

2.1.1.4 Flow Network

Social graphs may additionally be more complex than the graph presented in Figure 2.1. For instance, a flow network is a directed graph, which assigns a certain capacity to each edge in the graph. Based on the capacity of each edge, *flows* (i.e., messages passing through the network) are admitted or rejected from being executed. More formally, a flow network can be defined as follows:



(a) First Message (b) Second Message

Figure 2.5: Sending of two subsequent messages in the same flow network. In (a) a message with cost $c_m = 2$ can be sent from source S to destination D, as a path $\mathbf{R}$ with $c_m \leq R_{ij} \forall R_{ij}$ exists $(S-A-F-D)$. Afterwards, the capacity is reduced by c_m. As a result, the message in (b) cannot be sent (the cost exceeds the capacity on both incoming edges of D).

Definition 2.13 (Flow networks) *A flow network is a network of entities represented by a graph $G = (V,E)$, in which a non-negative real number R_{ij} is associated with each edge $e(i,j)$—the capacity of that edge. [99]*

Of particular interest for this thesis are the capacity constraints of flow networks:

Definition 2.14 (Capacity Constraint) *In a flow network, when trying to send a message m with a certain cost c_m from a source node s to a destination node d over the network, m can only be sent if and only if there exists a path $\mathbf{R} = [R_{s,i_1},\ldots,R_{i_n,d}]$ in the network, where $c_m \leq R_{ij} \forall R_{ij} \in \mathbf{R}$, i.e., all edges on the path have sufficient capacity to forward the message. [99]*

An example flow network, in which a source tries to send two subsequent messages to a destination is given in Figure 2.5.

2.2 Distributed Hash Tables

Both centralized and decentralized OSNs often rely on a *Distributed Hash Table* (DHT) to handle large amounts of users. Whereas centralized OSNs use DHTs such as Cassandra [100] to ensure high data availability across multiple datacenters without a single point of failure [101], decentralized OSNs rather make use of DHTs as their basic infrastructure.

A DHT implements the functionality of a hash table in a decentralized manner, i.e., it provides a table that allows key-value pairs to be inserted and queried for—without storing the table at a centralized instance. For this purpose a DHT requires the participation of nodes in the network in maintaining the table. In detail, the participating nodes are interconnected in a structured overlay network and each node is responsible for a partition of the network. Here, the global *key-space* is distributed among the nodes. For those keys in the partition of the key-space for which a node is responsible, the node stores all relevant key-value pairs and answers queries (*lookups*) of other nodes towards keys within its partition.

The literature offers a number of approaches to DHTs [102–106], which have also been successfully deployed to popular *Peer-to-Peer* (P2P) networks. For instance, BitTorrent uses Kademlia [104] as a lookup directory [107]. While sharing a common name, each DHT defines the form of the structured overlay (including the ordering of nodes) and the way of routing messages between the nodes in a unique way. For instance, the CAN overlay is built on a multi-dimensional coordinate-space [105], whereas nodes form a ring structure in Chord [103]. Thus, to route a query to a certain key in the key-space, CAN maps that key to a point in the coordinate-space, while Chord numerically orders nodes according to their identifier and then matches the key to a node identifier.

Although diverse in their architecture and lookup mechanisms, all approaches are similar in their performance. Most major DHTs route a lookup request from the source to the node storing the key in $O(\log n)$ steps, where n is the number of nodes in the overlay—the exception is CAN, in which the path length grows as $O(n^{1/d})$, where d is the dimension of the overlay [108]. This property enables efficient forwarding of queries, insertions and maintenance instructions even in large scale systems like BitTorrent or OSNs. All approaches also incorporate maintenance algorithms to handle node arrivals and departures. The handling of departures caused by a node failing is of particular importance, as such failures can cause the DHT to fail on a network-wide level [108]. CAN, Chord, Kademlia, Tapestry [106] and Pastry [102] all offer mechanisms to prevail against such failures.

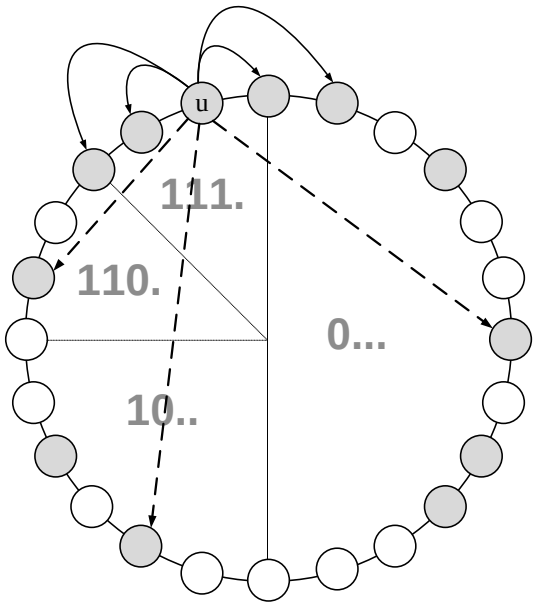


Figure 2.6: An exemplary Pastry ring. Dark colored nodes are online nodes. The routing table of node *u* is depicted by the dashed line, whereas its leaf set is represented by the solid arrows.

2.2.1 The Pastry DHT

In later parts of this thesis the Pastry DHT [102] will be used in the process of building a novel DOSN. The reason for choosing Pastry over other DHTs is that there exists an open-source JAVA Pastry implementation, which features a easy-to-use *Application Programming Interface* (API) and—more importantly—is maintained continuously.[9] Pastry further—as most other DHTs—offers the critical $O(\log n)$ performance and mechanisms to handle node failures.

In Pastry, nodes are arranged in a circular 128-bit node key (*identifier*) space, denoted as *ring*, as shown in Figure 2.6. The identifier is assigned randomly as a new node joins the system (i.e., peers with adjacent identifiers are not necessarily geographically close), and is considered a sequence of digits with base B. B is a configuration parameter usually set at $B = 2^b$ with $b = 4$.

Each peer then maintains (i) a routing table; (ii) a leaf set and (iii) a neighborhood set.

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

The *routing table* consists of $\log_B n$ rows with each row of cardinality $|B-1|$. An entry in row m shares a prefix of length m with the local node but differs in the $m+1$th digit. In other words, each row in the routing table shares a successively longer prefix with the identifier of the node maintaining the table. This prefix is then used for a longest-prefix matching routing protocol. For instance, in Figure 2.6, node u's Pastry identifier starts with 111. It will thus have a routing table entry pointing to prefixes starting with zero ($m = 0$), 10 ($m = 1$) and 110 ($m = 2$).

The *leaf set* L contains the $|L|$ ($|L| = 2^b = B$, usually) nodes with the numerically closest node identifiers, in which $|L|/2$ entries contain nodes with larger identifiers and $|L|/2$ nodes contain nodes with smaller identifiers, as shown in Figure 2.6. The *neighborhood* set M consists of node identifiers and *Internet Protocol* (IP) addresses of the $|M|$ closest peers to the local node ($|M| = 2^b = B$, usually as well). The closeness of a node is determined by using a scalar proximity metric like the IP geographic routing distance.

2.2.1.1 Routing

Based on the data structures described above, the routing itself is done as follows. When a node faces a routing request it first checks whether or not the key falls within the range of its leaf set. If so, the message is directly forwarded to the closest node to the key in the leaf set, which is the destination of the request. If not, the request is forwarded using the routing table by applying longest prefix match forwarding at each node on the path. In the case of a node failure or missing entry in the routing table, the request is forwarded to a node sharing a prefix of the same length as the local node, which at the same time is numerically closer to the key than the local node.

2.2.1.2 Node Join

If a node, say u, wants to join the overlay it has to (i) inform the other nodes of its presence and (ii) build an initial routing table. To do so, it needs to know at least one node, say v, in the network (e.g., based on some external mechanism providing a list of bootstrap nodes). It then sends a `JOIN` message to v, which is however targeted at u's own identifier. This request is routed through the network to the node w with the identifier numerically closest to u's identifier.

Upon receiving the request, all nodes on the path from u to w send their routing tables to u, and u uses that information to build its own routing table as follows: u most probably does not share a prefix with v (the bootstrap node), but may take the first row v_0 of v's routing table, since it does not need to share any prefix with these entries.

As the message is routed via longest-prefix matching afterwards, the m-th node on the path to w shares a prefix of length m with u. Therefore, u uses the m-th row of that node's routing table as its own m-th row. u also receives the leaf set of w since w is the numerically closest identifier and its leaf set is thus suitable for u as well. As u is also probably being in proximity to v, v's neighborhood set is used to initialize u's neighborhood set. Finally, u sends its routing table to each node in the leaf set, neighborhood set and routing table. The recipients update their own tables based on this information.

2.2.1.3 Node Leave

A node x is considered to have left the network if its immediate neighbors in the identifier space can no longer communicate with it. In that case, nodes for which x is a member of the routing table, leaf set or neighborhood set have to update those. To fix its routing table in case of a failing entry, a node u contacts a node in the same row as the failed entry. It asks that node for its entry at the position of the failed entry. To update its leaf set, if u was a neighbor of the failed node x, u requests the leaf set L' from the live node v with the largest index on the side of the leaf set x was on. L' partly overlaps with u's leaf set L. Node u then inserts a suitable node of L' into L after verifying that it is still alive by contacting it. To keep the neighborhood set up-to-date each node u further periodically contacts all the entries of its neighborhood set. If one of these does not respond, u asks other nodes to transmit their neighborhood sets and determines the closest distance of an entry of these sets as its new neighbor.

2.3 Cryptography Basics

As a consequence of the prevalence of computers and communication systems, a demand for mechanisms to securely exchange data in these systems has emerged [109]. The goals of a secure data exchange are manifold and range from ensuring the authenticity of a sender to enabling access control for data [109]. Modern cryptography mainly exploits two major concepts to accomplish these goals: *symmetric cryptography* and *asymmetric cryptography*.

2.3.1 Symmetric Cryptography

Symmetric cryptography uses the same key K for the encryption of plain text and for the decryption of ciphertext [109, 110]. The key is ought to be known only by the communi-

cating entities after an exchange over a supposedly secure channel. The two main types of symmetric cryptography are stream ciphers and block ciphers.

In *stream ciphers*, the digits (i.e., bytes) of a plain text message are encrypted with their corresponding digit in a pseudorandom cipher digit stream, one digit at a time. In *block ciphers*, several bits are encrypted as a single unit (e.g., in the *Advanced Encryption Standard* (AES) [111] the block size is 128 bits). Here, the plain text is padded so that its length is a multiple of the block size.

One popular stream cipher example is the A5 cipher, which is used in, for instance, cellular networks [112]. A5 is based on *Linear Feedback Shift Register* (LFSR) to keep the internal state of the cipher. LFSR are shift registers, which use *Exclusive OR* (XOR) operations as feedback to move from one internal state to the next one.

Well known examples for symmetric block cipher cryptography are the *Data Encryption Standard* (DES) (previously widely used [109], but now considered insecure), and its successor AES, a common cryptography standard.

The key advantage of symmetric schemes like A5, DES, AES, or Blowfish [113] is that XOR and most other implemented operations are basic binary operations, which makes their implementation in hardware easy. By choosing fast, hardware efficient ciphers, symmetric encryption thus offers a good user experience (i.e., short latencies) and is suitable for mobile computing because of a low energy consumption.

However, a major drawback of symmetric cryptography is that the secret key K needs to be exchanged over a possibly insecure channel before the communication starts—and secretly kept only between communication partners afterwards. In case of a compromised key, all encrypted messages can be decrypted.

2.3.2 Asymmetric Cryptography

To mitigate the problem of secure key exchanges, asymmetric cryptography (or public-key cryptography) builds on two different keys for encryption (K_E) and decryption (K_D), and each user owns one key K_E and one key K_D, respectively [109, 110]. The keys themselves are designed such that a user Alice, who owns K_E and K_D can decrypt a message encrypted with K_E by applying K_D and K_D only. Alice publishes K_E (also named *public key*) and keeps K_D secret (also called *secret* or *private key*). The most important principle of asymmetric cryptography is thus that K_D—as the only way to decrypt a message encrypted with K_E—cannot be computationally derived from K_E [109]. Hence, to ensure message confidentiality, any sender can encrypt messages for Alice, but only Alice can decrypt the messages. To ensure sender authenticity and message integrity, Alice can further digitally sign the message

with K_D, and the signature can be verified with K_E.

For the verification, it is further necessary that Alice's public key is certified, i.e., that other users can validate the binding of $K_{E(Alice)}$ to Alice's identity. Two major approaches exist to achieve this certification.

(i) A *Certification Authority* (CA) computes the keys and assigns them to Alice's identity, i.e., it issues a digital certificate to Alice that confirms the binding [114]. Users intending to communicate with Alice can then, with the help of the CA, check the validity of Alice's certificate before initiating the communication. Here, the CA is a trusted third party. Currently, the party holding the most certificates is the Symantec Group [115].

(ii) Alternatively, asymmetric cryptography can also work without a (centralized) trusted third party. In *Pretty Good Privacy* (PGP), the users themselves form a *web of trust*, and vouch for each other [116]. For instance, a friend Bob of Alice can digitally sign Alice's PGP public key to vouch for its validity. Any other user who trusts Bob's signing policies will then also accept Alice's public key as valid.

A major advantage of public-key cryptography is that it is much easier to provide authentic and secure *public* keys than it is to share a symmetric key *secretly*. On the other hand, asymmetric cryptography usually deals with larger keys and is thus considerably slower than symmetric cryptography.

The algorithms to manage the generation of keys, encryption and decryption routines, and signature procedures are usually bundled within a *cryptographic system*.

Definition 2.15 (Cryptographic System) *A cryptographic system is a set of cryptographic algorithms together with the key management processes that support use of the algorithms in some application context. [117]*

Two major asymmetric cryptographic systems are the *Rivest-Shamir-Adleman* (RSA) cryptographic system [118] and the *Elliptic Curve Cryptography* (ECC) cryptographic system [119, 120].

2.3.2.1 RSA Cryptographic System

RSA, named after its inventors Rivest, Shamir, and Adleman in 1978, is the most commonly used public-key cryptographic system [118]. The security of RSA (i.e., the inability to com-

putationally derive K_D by knowing K_E) relies on the assumption that factoring the product of two large prime numbers p and q is hard, whereas the multiplication of p and q is easy.

In RSA, the *RSA module* $N = pq$ and either an encryption exponent e or a decryption exponent d are combined to generate $K_E = (e,N)$ and $K_D = (d,N)$ respectively. Here, e is co-prime to $\phi(N)$, where ϕ is Euler's totient function. Also, d is a multiplicative inverse of e such that $ed \equiv 1 \pmod{\phi(N)}$.

To achieve message confidentiality, a sending user Bob can then encrypt a message m to a ciphertext c by using the K_E of the receiver Alice to compute $c \equiv m^e \pmod{N}$, such that Alice can decrypt m with her K_D as $m \equiv c^d \pmod{N}$, because $ed \equiv 1 \pmod{\phi(N)}$.

Next to message confidentiality, two further main goals are to ensure message integrity (i.e., that the message has not been tampered with) and to authenticate the sender. RSA uses *digital signatures* to achieve both.

Definition 2.16 (Digital Signature) *A digital signature is a value computed with a cryptographic algorithm and appended to a data object in such a way that any recipient of the data can use the signature to verify the data's origin and integrity. [117]*

To create the signature for his message to Alice, Bob will first apply a hash function H to m to obtain $H(m)$ (for details on hash functions see [121]). He will then apply his private key to $H(m)$, and attach the resulting value as a signature to the message itself. Since e is the multiplicative inversive of d, Alice, upon retrieval of the message, can apply Bob's public key to the signature.

Afterwards, Alice can compare the hash value of the signature with the hash value of the message itself. If both are equal, Alice can then verify that the message was not modified after sending (integrity), and that the author of the message was in possession of Bob's private key (authenticity).

The recent increase in computational power does not compromise the security of RSA, as the usage of longer keys counteracts the factorization of N into p and q in practical time. Currently, a key length of 2048-bit is recommended [122].

2.3.2.2 ECC Cryptographic System

Elliptic Curve Cryptography (ECC) cryptographic systems were introduced independently by Koblitz and Miller in 1985 [119, 120], and are nowadays widely implemented [123]. Their primary benefit over RSA lies within usually shorter keys, as, for instance, a 256-bit ECC key provides comparable security to a 3072-bit RSA key [124]. The security of ECC relies on more complex assumptions than RSA. Here, the main principle is that the division of two points in a cyclic group on an elliptic curve is hard, whereas the addition of the points is easy.

In the *Elliptic Curve Integrated Encryption Scheme* (ECIES), an elliptic curve is defined to be a plane curve over a finite field that consists of the points satisfying $y^2 = x^3 + ax + b$, where a and b are predefined constants. Since the definition of an elliptic curve requires an agreement between all participants of the communication on all the elements that define the curve (such as, among others, a and b), these are usually not negotiated before beginning a conversation. Participants rather agree on one of multiple pre-defined parameter settings denoted as (p,a,b,G,n,h), as, e.g., given in [125, 126]. Here, p defines the finite field over which then the curve is defined, G is the generator point of the elliptic curve, n is the cryptographic order of G, and an h is the cofactor, i.e., the ratio between the order of p and the order of G.

Then, a user Alice generates her private key $K_D = d$ and public key $K_E = dG$ (recall that deriving d from dG is computationally hard), where d is a randomly selected integer with $d \in [1, n-1]$ (n according to the parameter settings). Hence, the public key is the result of adding G together d times.

To encrypt a message, Alice uses a hybrid scheme of symmetric and asymmetric cryptography. She will first select a random number $r \in [1, n-1]$ and calculate $R = rG$. From the recipient Bob's public key, she will then derive a shared secret (a point)

$$S = P = (P_x, P_y) = rK_E^{Bob} \tag{2.3.1}$$

From S, Alice will afterwards derive both a symmetric encryption key and a *Message Authentication Code* (MAC) key with a *Key Derivation Function* (KDF)

$$k_E || k_M = KDF(S||S_1) \tag{2.3.2}$$

where S_1 is some optional shared information (for details on MACs and KDFs see [127, 128]). After encrypting the message to ciphertext $c = m_E^K$, she also attaches

$d = MAC(k_M; c||S2)$, where $S2$ is another optional shared information. She will then send $R||c||d$ to Bob.

To decrypt the message, Bob will derive S and $k_E||k_M$ in the same fashion as Alice with the help of R. He will then validate $d = MAC(k_M; c||S2)$, and, if the validation holds, decrypt the message with the symmetric key as $m = c^{k_E}$.

The principle of digital signature generation is similar to the one used in RSA. Alice will first compute the hash value $H(m)$ over the message m. She will then sign this value with her private key, whereas a recipient Bob will use Alice's public key to verify the signature. In detail, Alice will, after calculating $H(m)$, take z, an integer representation of the leftmost bits of $H(m)$, and select a random number $k \in [1, n-1]$. Afterwards, she will calculate the curve point $(x_1, y_1) = kG$ and $r = x_1 \mod n$.

She finally calculates

$$s = k^{-1}(z + rd) \mod n \tag{2.3.3}$$

The signature pair (r,s) is then attached to the message itself. Bob first verifies that the signature can be a correct ECC signature (e.g., r and s have to be in $[1, n-1]$), and computes $H(m)$ himself. Afterwards, similarly to RSA, he calculates an inverse $w = s^{-1} \mod n$ and thereafter $u_1 = zw \mod n$ and $u_2 = rw \mod n$. Based on the results, Bob can now determine the curve point

$$(x_1, y_1) = u_1 G + u_2 K_{E_{Alice}} \tag{2.3.4}$$

The signature is then valid if and only if $r \equiv x_1 (\mod n)$.

2.3.3 Attribute Based Encryption

In traditional, asymmetric cryptography, a message is encrypted for a specific receiver using the receiver's public key, and the public key is usually verified by a CA or other users (as in PGP). The public key usually does not contain any further information about its owner.

On the contrary, *Identity Based Encryption* (IBE), an ECC based encryption scheme, changed the perspective of public key encryption in that it allowed the public key of a user to be a publicly known, arbitrary string identifying that user, such as her email address [129, 130]. This allows users to encrypt messages to a publicly known identity without access to

the recipient's public key certificate, and therefore without requiring the assistance of a CA or other users for looking up certificates.

Subsequently, Fuzzy-IBE introduced some fault-tolerance to IBE [131]. Fault-tolerance is useful for, e.g., biometric cryptography applications, where two samples taken at different times usually differ to a small extent. Fuzzy-IBE lays the foundations for *Attribute Based Encryption* (ABE), as it is the first cryptographic approach to view the identity of a user as a set of descriptive *attributes*. Here, a user is able to access a message if and only if her identity is in a certain range of the public key used to encrypt the message.

ABE itself is a relatively recent approach towards public-key cryptography [36, 132]. It generalizes the concept of (Fuzzy-)IBE and defines the identity of a user as a set of attributes, and messages can be encrypted with respect to those attributes (*Key Based Attribute Based Encryption* (KB-ABE) [133]) or policies defined over a set of attributes (*Ciphertext Policy Attribute Based Encryption* (CP-ABE) [132]). In this thesis, wherever ABE is mentioned, it refers to CP-ABE. The main advantage of CP-ABE over KB-ABE is that, in the latter, the encrypter cannot control who accesses the encrypted data by other means than assigning attributes to the *data*. In CP-ABE however, the encrypter is additionally able to grant access to the data by assigning attributes to *identities*, which then—on a high level—have to match with those assigned to the data to enable the decryption of the data [132].

In detail, access to an encrypted message is granted by an *Access Structure* (AS) over a defined universe of attributes within the system.

Definition 2.17 (Access Structure) *Let $P_1, P_2, ..., P_n$ be a set of attributes. A collection $\mathbb{A} \subseteq 2^{P_1,P_2,...,P_n}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$ then $C \in \mathbb{A}$. An access structure is a (monotone) collection $\mathbb{A}$ of non-empty subsets of $P_1, P_2, ..., P_n$, i.e., $\mathbb{A} \subseteq 2^{P_1,P_2,...,P_n} \setminus \emptyset$. The sets in $\mathbb{A}$ are called the authorized sets and the sets not in $\mathbb{A}$ are called the unauthorized sets. [132]*

Here, a user will only be able to decrypt the message, if and only if her ASK is associated with the attributes to satisfy the AS of the message. ASs can be specified over (groups of) attributes using (k,n) threshold gates, where k out of n attributes have to be present in the ASK of the requester to decrypt the message.

The logical operations `AND` ($\wedge$) and `OR` ($\vee$) can be derived from these gates as (n,n) and $(1,n)$ threshold gates respectively [133]. Theoretically, ABE is also able to express non-monotonic ASs by including the logical `NOT` ($\neg$), which however would be computationally expensive [132].

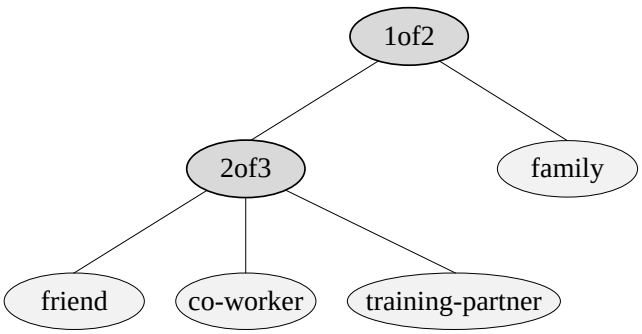


Figure 2.7: The ABE access tree of the AS 'friend co-worker training-partner 2of3 family 1of2'. Two (k,n) threshold gates (*2of3*, *1of2*) are applied to a total of four attributes.

ABE will process the attributes defined in the AS based on an *access tree*. The access tree is usually traversed in post order, i.e., first the left part of the tree, then the right part of the tree and finally the root is visited. Figure 2.7 shows the access tree for the example AS 'friend co-worker training-partner 2of3 family 1of2'.

Nonetheless, even without a `NOT` operation, ABE is highly expressive. The expressiveness however comes at the price of high complexity, since *all* operations in ABE are asymmetric.

2.3.3.1 Cryptographic Operations

Based on these high-level concepts and on the concepts introduced in the discussion of the ECC cryptographic system (cf. Section 2.3.2.2), Sahai et al. define the following cryptographic operations in [132]:

Let S be a set of elements in $\mathbb{Z}^+$ under the Lagrange coefficient Δ (for details on Lagrange polynomials and their coefficients, see [134]).

Setup. Given a bilinear group $\mathbb{G}_0$ of order n with a generator g (for details regarding the bilinearity, see [135]), the setup algorithm chooses two random exponents $\alpha, \beta \in \mathbb{Z}$. Then, the *ABE Public Key* (APK) of a user is

$$APK = \mathbb{G}_0, g, h = g^{\beta}, f = g^{1/\beta}, e(g,g)^{\alpha} \tag{2.3.5}$$

where e is the bilinear map $\mathbb{G}_0 \times \mathbb{G}_0 \to \mathbb{G}_1$. At the same time, the *ABE Master Secret Key* (AMSK) is computed as (β, g^{α}).

Encryption. Let τ be an access tree. Then, the encryption algorithm encrypts a message M under the access structure of τ. For each node of τ, it selects a polynomial q_x that is of a degree that is one less than the threshold value k of that node, and that is defined by $k-1$ points randomly selected from $\mathbb{Z}^+$.

That is, starting at the root node R, $q_R(0) = s = \text{rand}(\mathbb{Z}^+)$, and for any other node x, $q_x(0) = q_{\text{parent(x)}}(\text{index(x)})$. The remaining points to define q_R and q_x are chosen randomly. Then, if Y is the set of leaf nodes in τ, the *Ciphertext* (CT) is computed as

$$\begin{aligned} CT = (\tau, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s, \\ \forall v \in Y : C_y = g^{q_y(0)}, C'_y = H(att(y))^{q_y(0)}). \end{aligned} \tag{2.3.6}$$

Here, H is a collision resistant hash function.

Key Generation. The input for the key generation algorithm is a set of attributes S. The algorithm again chooses a random $r \in \mathbb{Z}_+$, and for each attribute $j \in S$ it chooses another random $r_j \in \mathbb{Z}_+$. Then, the ASK is generated as

$$\begin{aligned} ASK = (D = g^{(\alpha+r)/\beta}, \\ \forall j \in S : D_j = g^r \cdot H(j)^{r_j}, D'_j = g^{r_j}). \end{aligned} \tag{2.3.7}$$

Decryption. Finally, the decryption algorithm takes the CT, an ASK associated with a set S of attributes and a node x from the access tree τ as input. If x is a leaf node, then $i = att(x)$ and if $i \in S$, then the decryption algorithm computes

$$\begin{aligned} \text{DecryptNode(CT, SK, x)} &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= \frac{e(g^r \cdot H(i)^{r_i}, h^{q_x(0)})}{e(g^{r_i}, H(i)^{q_x(0)})} \\ &= e(g,g)^{rq_x(0)}. \end{aligned} \tag{2.3.8}$$

Otherwise, if $i \notin S$, then the algorithm returns DecryptNode(CT,SK,x) = $\perp$.

In other words, if the ASK contains the attribute describing the tree node (i.e., it contains *i*), the node is satisfied, whereas if the ASK does *not* contain the attribute describing the node, the node is not satisfied.

Then, for a non-leaf node x the decryption algorithm computes DecryptNode(CT,SK,z), for all child nodes z of x. The node x is then satisfied if there exists a set of child nodes S_x of x of arbitrary size for which $\forall z \in S_x$: DecryptNode(CT,SK,z) $\neq \perp$. In other words, depending on the definition of the AS $\mathbb{A}$, a number of child nodes have to be satisfied for x to be satisfied.

To decrypt the ciphertext, the decryption algorithm then executes DecryptNode(CT,SK,R) for the root node R of τ, which recursively calls DecryptNode(CT,SK,z) on all child nodes z of R, until it reaches a leaf node. Then, if the tree is satisfied by the ASK associated with S, it sets

$$A = \text{DecryptNode(CT, SK, r)} = e(g,g)^{rq_R(0)} = e(g,g)^{rs} \tag{2.3.9}$$

and decrypts the ciphertext with

$$\tilde{C}/(e(C,D)/A) = \tilde{C}/\left(e\left(h^s, g^{(\alpha+r)/\beta}\right)/e(g,g)^{rs}\right) = M. \tag{2.3.10}$$

Chapter 3

Why Do We Need Decentralized Online Social Networking?

All the seemingly trivial details we reveal about ourselves online every day can be cross-referenced and correlated often to startling effect.

— Tom Chatfield on BBC, "Do we reveal too much about ourselves online?" [136]

All major OSNs are currently orchestrated by a single provider. This provider usually offers a sophisticated centralized infrastructure, which is used by an ever-growing number of OSN participants to exchange data. However, concomitant with these networks' tremendous growth are increasing concerns from users about their privacy and the protection of their data. As both the infrastructure and user data management are centralized, the provider has the unprecedented privilege to access every user's private data.

In this chapter the case for a different approach towards online social networking is made. Instead of relying on a central provider, *decentralized* OSNs can enable social networking services, in which the access to and management of user data is controlled by the users themselves. Additionally, such decentralization approaches can also help to increase the usability of OSN applications.

Contents

3.1 Issues with Centralized OSNs

The remarkable growth of OSNs has inherently led to tremendous amounts of user information being part of these networks. At the same time, this information is maintained by a single instance—the provider of the OSNs (e.g., Facebook, Google, or Yahoo). This situation has raised severe privacy and security concerns [21, 22].

First and foremost, the control over huge amounts of user data without restriction of any kind is worrisome itself, because the providers can obtain a deep insight into their users' personal interests, opinions, social relationships, and economical or political preferences. For instance, recent lawsuits against Facebook and other OSN providers (e.g., Google and Yahoo) complain about the practice of tapping into the users' private messages for the purpose of content analysis [137, 138]. Moreover, both Facebook and Google have introduced a clear-name policy, which makes the use of real names as user names mandatory; not following the directive will result in an exclusion of the user from the OSN [30].

The providers have good reasons for their actions: For instance, Facebook is currently creating 85% of its annual income from personalized advertisements [35], which can be customized better—and therefore sold with greater revenue—if precise user profiles are available. By extracting interests or product preferences from user data such as messages, the profile precision can be increased; the clear-name policy further eases the linking of existing OSN profiles with all sorts of information available elsewhere [139].[10]

These problems even affect persons who do not have an account in the OSN, let alone uploaded any data to its servers [140]. Providers have started to collect information about non-members from data uploaded by the members of the OSN, a practice which has been coined *shadow profiling* [141]. For instance, Facebook has crawled its users' email or phone address books for such information [141]. As a consequence, even though a particular person might not own any OSN account, significant fractions (e.g., a phone number or address) of her data can still be in the hands of a provider.

The profiling of users does not even stop at the boundaries of the OSN providers. Between 2007 and 2009, Facebook and a group of partners (among them, e.g., Amazon, eBay and Sony [142]) implemented the Beacon application, which forwarded sensitive shopping information of users along the partners without the users' consent. Beacon was only stopped in the course of a class-action lawsuit, which cost Facebook 9.5 million US dollars to settle [25]. Additionally, a large group of major OSN providers—including Facebook, Google and Yahoo—granted full access to user data to government agencies within the PRISM program without any knowledge of their users [28]. Extended cooperation or collaboration of providers with government institutions could thus ultimately result in the "transparent

[10] For further issues with the policy see, for instance, `http://mynameisme.org`

user", where all available information about each single user is available in a bundle at a single instance, without the user's knowledge and thus also without her consent.

From a different perspective, data misuse does not only happen at the will of the provider. Storing all user data at a single entity also increases the risk of a major external privacy breach. Attackers seeking to obtain user data only need to compromise that entity to gain access to all data. The reality of this danger was demonstrated in June 2012, when eight million users of LinkedIn[11] saw their passwords leaked from the company's central repository [33]. Earlier, attackers were able to obtain access to millions of accounts on the now shut down German OSN SchülerVZ in multiple attacks in 2009 and 2010 [143]. This particular leak drew considerable public interest, as most members of SchülerVZ are children [144].

Further, users of OSNs, even those familiar with current networks' limited privacy settings, tend to underestimate their audience. As a result, private user data is often visible to a larger audience than intended by the data owner herself [145, 146].

Finally, the providers do not only endanger user privacy. The terms of use are often difficult to process and at the same time invalidate property rights [29]. For instance, for every photo uploaded to Facebook, the user grants a simple usage right for that picture to the company, and the photo will remain on Facebook indefinitely (see Section 2.1 of [147]). Additionally, the central provider might at some point introduce a usage fee to a previously free-of-charge service. Users would then face the ostensible choice to either lose their social network or to pay the fee to continue using the service.

In summary, there exists an obvious and urgent need for increased privacy in OSNs. However, if OSN providers would indeed aim at protecting user data, this would require them to forfeit access to those data. In addition to not longer being able to cooperate with government institutions, such a concession would be tantamount to giving up a number of economical advantages, including (i) the opportunity to analyze the data for personalized advertisement; (ii) the possibility to link external publicly available information with the OSN profiles of their users; (iii) the option to exchange data with other providers to complete their own view on the data; and (iv) usage rights on the content. As a consequence, it is unlikely for OSN providers to allow users to apply comprehensive security and privacy means to their data. Current OSN users can thus not expect any provider efforts to drastically improve the current privacy situation [1].

[11] http://www.linkedin.com

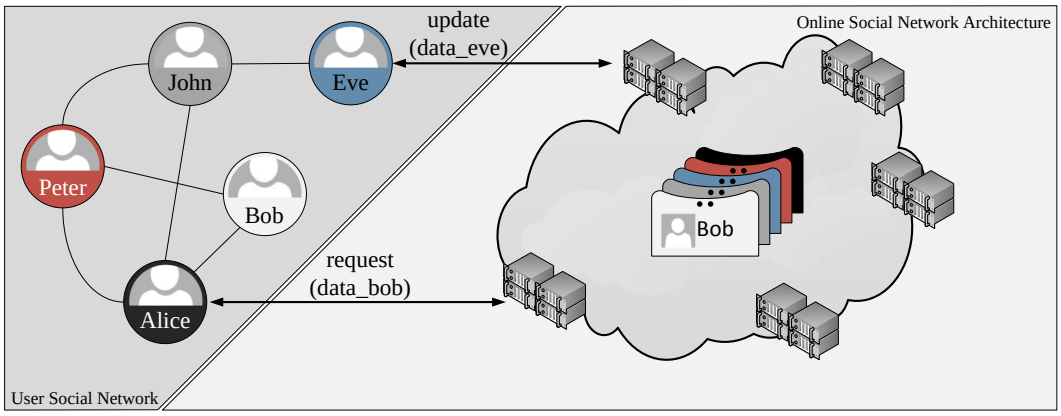


Figure 3.1: An exemplary centralized OSN. User data is stored across multiple interconnected datacenters, which are controlled by a central entity. Encryption for user data is non-existent.

3.2 The Advantages of Decentralized Online Social Networks

The organization of a centralized OSNs usually comprises several inter-connected datacenters to host unencrypted user data, as shown in Figure 3.1. Google, for instance, operates thirteen different datacenters across the globe [148]. User data is replicated among these datacenters either with full replication or geographical replication schemes [149, 150]. In full replication, each user's data is available at every datacenter, while geographical replication schemes try to save storage space by replicating a user's data only to datacenters in geographic proximity of the user. Although the inter- and intra-datacenter communication might thus be complex itself, connecting a user to the data is relatively easy in a traditional client-server fashion. For instance, in Figure 3.1, Eve can update her data directly at the corresponding datacenter(s), and Alice can request Bob's data from there as well. In fact, the central provider deals with all technical challenges of social networking, such as the massive scale of OSNs, averting malicious attacks, and so on. The downside however is, as described above, that the provider also has full control over the network and any data stored in the datacenters.

A *Decentralized Online Social Network* (DOSN) on the other hand is an OSN that is *not* controlled by a provider, but is rather maintained by a multitude of *distributed* entities. As a consequence, a DOSN usually does not benefit from a sophisticated centralized infrastructure, but has to construct a substrate for that infrastructure itself. The construction approach is a matter of system design, and may greatly differ from one DOSN to another.

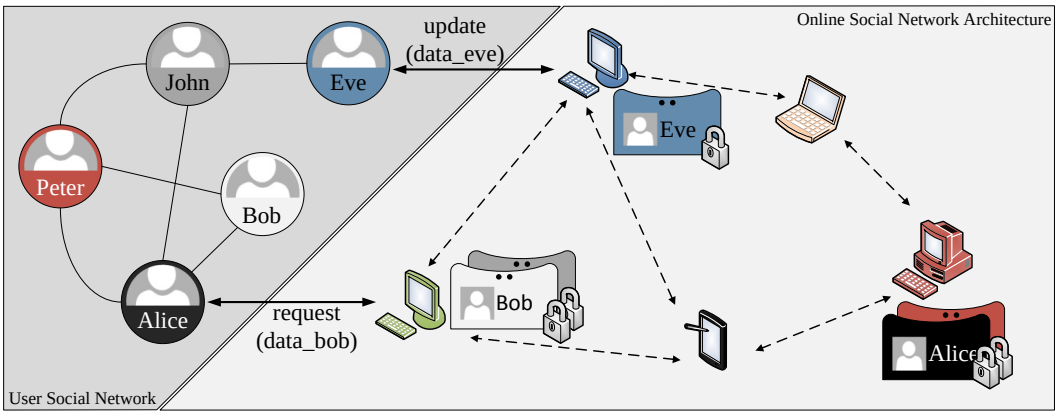


Figure 3.2: An exemplary *de*centralized OSN. The central provider is removed and encrypted user data is stored among the users themselves.

As an example, Figure 3.2 shows a decentralized OSN, in which the users themselves absorb the absence of the central provider by storing encrypted data across the participants' devices, and these devices control access to the data. Another option to realize the substrate is to make use of permanently available resources. For instance, a DOSN can delegate data storage to paid servers in cloud environments, or to personal, dedicated servers.

Connecting a user to data is, however, more difficult. For instance, as there exists no central user data directory, users first need to locate data of interest in the network before they can request it. The location, at which the data is located, might further be a node that was previously unknown to the requesting user. As an example, in Figure 3.2, Alice requests Bob's data from a user she does not necessarily know. At the same time, Eve can update her data at her own machine, but has to somehow communicate the information about the update to other users.

Although removing the central provider is technically challenging, by building a DOSN on either of these construction approaches, users can regain control over their data. In both forms of decentralization, they can determine the location of their data themselves. Additionally, and more importantly, users are able to encrypt their data such that it is only accessible by eligible users. For instance, in Figure 3.2, Bob's data is encrypted, and Alice can only access it, if she owns the proper decryption credentials. In particular, the storage provider itself is not able to analyze any data (unless access is granted by the user).

Further, a decentralized approach is also a natural fit for online social networking for several reasons.

First, the communications paths in OSNs do not require the presence of a provider. Although the provider of current centralized OSNs offers the infrastructure to distribute and access the content, it does not significantly contribute to the content itself. In fact, online social networking is rather peer-to-peer in nature, as the users of an OSN generate content for other users (e.g., their friends) and not for any third party like a provider. In a DOSN, both the infrastructure and access to user data would be maintained by those entities that produce and consume the content anyway—the users (or their surrogates) themselves.

Second, a decentralized system without a single, central data repository also limits the risk of large-scale privacy breaches [22]. Since such a system would rely on a multitude of distributed storage repositories for user data, a breach at a single one of them would not expose all user data to the attacker.

Third and finally, DOSNs can—without decreasing the users' privacy—remove one of the big barriers of todays OSNs, the non-existing interoperability between several of these networks [151]. Currently, a user needs one account for each OSN she is a member of (e.g., Facebook, Twitter, Flickr, etc.), and additionally inevitably shares her data redundantly with many providers at the same time. Worse, if there was a single-sign-on for all OSNs the user is registered with, the provider of this service could connect all data of users from several OSNs. A generically designed DOSN however can allow users to accurately control a single set of encrypted data, and to reduce the hassle of handling multiple accounts, while still granting access to a multitude of OSN applications on a fine-grained basis.

Chapter 4

Related Work

The previous chapter discussed that, even though centralized OSNs are extremely successful and home to billions of users, *Decentralized Online Social Network*s (DOSNs) offer a promising alternative for both better user data privacy and improved usability. As a consequence, researchers have started to propose approaches to decentralize OSNs with increasing frequency over the past years. These approaches greatly differ in the way the central provider is replaced by a distributed architecture and the features they offer to their users.

This chapter conducts a comprehensive review of state-of-the-art DOSNs. The review is started with an analysis of whether or not classical P2P or distributed storage solutions can be reused to create the storage substrate in DOSNs. Afterwards, existing DOSNs are systematically grouped into three broad categories of working principles—(i) systems, which rely on permanently available resources; (ii) approaches, which rely on the cooperation of users; and (iii) hybrid systems, which contain elements from both other categories. Within each of these categories, the existing approaches are introduced and discussed in detail.

To conclude the review, a comparative analysis of the strengths and weaknesses of each DOSN is provided. The analysis emphasizes the key features that each DOSN provides as well as the properties that may be missing to achieve competitiveness with centralized OSNs.

Contents

4.1 Classical Distributed Storage Solutions

In the course of providing an efficient storage substrate, classical P2P or distributed data storage approaches might offer a solution (for a detailed survey see, for instance [152]). However, these approaches are generally designed for supporting traditional decentralized applications such as file sharing, which are often characterized by long durations of user online time, typically spanning from multiple hours up to days. For instance, BitTorrent users have been shown to exhibit long segments of online time during file seeding, and desktop machines at workplaces have been shown to be online almost permanently [153–155]. At the same time, orthogonal to such applications, users' online patterns in social networking sites show high activity peaks with large gaps of offline time [82, 156]. The main contributing factor to these patterns is that OSN content is often uploaded and accessed from mobile devices, which may be disconnected most of the time.

Moreover, the goals of replication differ between classical P2P storage and storage in OSNs. Whereas files in traditional P2P systems often have a limited required lifetime due to their exponential decrease in popularity [157], data in OSNs should be available even after the first interest in the data abates. For instance, users will want to keep their conversations with others for later review or may want to look at photos that might have been taken several years ago, even though the vast majority of the content might not be accessed frequently after its initial exchange.

Another OSN characteristic is the inherent relation between the participants, which can imply storage incentives among the users of the OSN [158]. Intuitively, a user will prefer to store the data of a friend to that of a stranger, and in case of storage exhaustion, a user would prefer to help friends to achieve a better performance, and thus would rather drop data of the stranger.

Furthermore, in contrast to traditional P2P systems, the tit-for-tat strategy is not as desirable for OSNs. In a tit-for-tat strategy, a user's performance depends on the contributions she makes towards the network. For instance, in BitTorrent, users will be able to download files faster if they contribute more bandwidth for uploading files themselves. Applied to the OSN scenario, in terms of data availability, users would achieve high availability for their data if they are often online and host data themselves.

However, users rather need the OSN *as a whole* to be robust, with each user's data accessible at any given time. Otherwise, even highly contributing users may find data of interest unavailable. For instance, consider a highly contributing user Alice who wants to access the newest photo album of her friend Bob. If Bob, for whatever reasons, did not contribute enough to the network, Alice—in spite of her earlier performance—might not be able to see Bob's photos.

Table 4.1: A chronologically ordered categorization of DOSN approaches.

Permanent Resources			Node Cooperation			Hybrid		
Approach	Year	Ref.	Approach	Year	Ref.	Approach	Year	Ref.
Persona	2009	[36]	Friendstore	2008	[40]	Confidant	2011	[47]
Diaspora	2010	[37]	PeerSoN	2009	[41]	SuperNova	2012	[48]
Vis-à-Vis	2011	[38]	Safebook	2009	[43]			
Contrail	2011	[39]	Cachet	2012	[44]			
			MyZone	2013	[45]			
			Proofbook	2014	[46]			

4.2 Specific DOSN Solutions

With these reasons in mind, researchers have suggested a wide range of solutions that are specifically tailored to decentralize OSNs [36–48]. These solutions can be divided into three categories that describe both their approach to replace the centralized infrastructure and thereby inherently also the assumptions under which this replacement occurs.

The first category are systems that exploit permanently available resources, which are built upon the assumption that every user has access to a device that can be online constantly. To relax this assumption, approaches that work without such resources but rather expect a cooperation of users constitute the second category. The third class of DOSN solutions then consists of hybrid systems, which contain elements of both previous categories. Such systems typically require that there are both permanently available resources *and* the cooperation of users.

Table 4.1 shows a chronologically ordered classification of existing DOSNs into these categories. It is striking that hybrid systems are rather existing in a niche, while most DOSNs do not assume any permanent resources, but build on the cooperation of users instead. Chronologically, the trend is that after an initial burst of node cooperation schemes (2008-2009), with the advent of cloud computing, researchers were more attracted towards hybrid and resourceful solutions (2010-2012). Thereafter however, cooperation schemes have—at least with regards to their numbers—become more popular again. The reasons for this latest development are twofold. First, the reputation of resourceful storage providers with regards to privacy has suffered recently, which has led researchers to avoid these providers. Second, there exist technical reasons to opt for node cooperation schemes. Both reasons will be discussed in detail in the following sections.

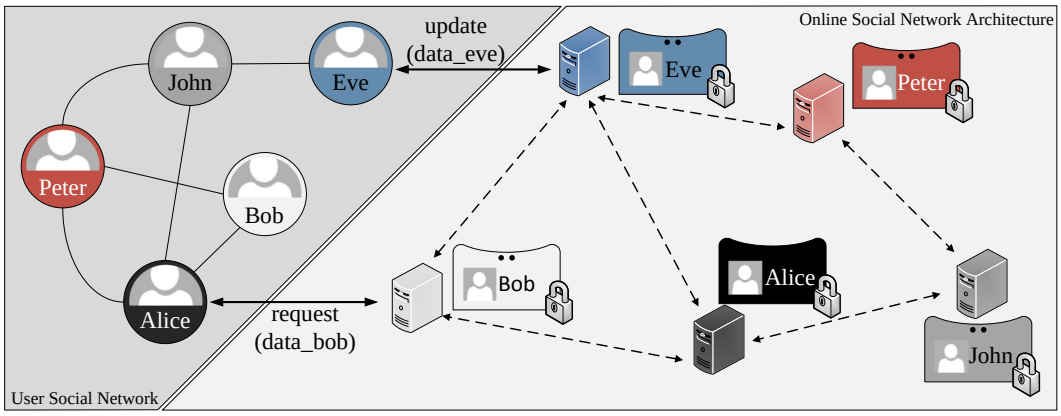


Figure 4.1: A decentralized OSN that exploits permanently available storage. The central provider is removed and each user provides a permanently available server to store her user data, as in, e.g., Persona. When Alice wants to retrieve Bob's data, she contacts the server where Bob has stored his data. When Eve wants to update her own data, she does so by manipulating it on her server.

4.2.1 Solutions Built on Permanently Available Resources

When assuming permanently available resources, the usual approach is to distribute data control and storage to a limited number of permanently online storage locations, which are shared by multiple users [36–39], as shown in Figure 4.1. The storage and control facilities might either be altruistically provided, or supplied based on economic incentives such as user payments.

In Persona [36], the approach is to ask each user to provide a permanently available storage space (e.g., a personal web server) for their own encrypted profile. This approach does provide high data availability and low overhead, since the data is permanently available at the server and no replication is required. However, it also requires *all* users to be technically able to provide and configure their own permanently available data storage, which is impractical. The issue might be mitigated by incentivizing storage and configuration providers, which however results in monetary costs for the user.

Diaspora [37] extends this solution by not only allowing each user to set up her own server, but also by relying on a limited number of altruistically provided servers. As an alternative to setting her own server, a user can also select one of the altruistically provided servers to store her data on. Diaspora itself then offers a server-overarching search function to find other users. To run her own server, a user would have to be able to first setup the

server physically and then to install Ruby, SQLite3, OpenSSL, and several other libraries required to run a Diaspora server [159]. Since this is usually too much effort required from a typical OSN user, most users have to rely on the altruistically provided servers.

Vis-à-Vis [38] gets rid of altruistically provided servers, but rather lets users store their private data on cloud services like the Amazon EC2 cloud [160] or Microsoft Azure [161]. Each user operates an independent server—a *Virtual Machine* (VM) running in a commercial cloud that hosts the user's data—and all independent servers in turn form the network. Content is then shared within groups of user servers, for each of which membership is administered by the user who created that group. As the operation of a VM is not free of charge, Vis-à-Vis—like Persona—requires the operation of a paid resource by each user.

Contrail [39] builds on a cloud storage substrate as well, and additionally relies on the cloud to act as a relay for messages between (mobile) users. Here, users do not interact directly between each other, but send and receive messages from so called *cloud relays* in a publish-subscribe fashion. A result of this extended cloud usage is a higher cost for the user.

In summary, while requiring each user to set up their own server is impractical, altruistic provisioning, usually from a limited set of volunteers, is unlikely to meet the demand of a large-scale social network with as many as several hundred million users. At the same time, motivating server providers with user payments will most likely prevent a large-scale transition from current centralized OSNs, which do not impose fees on their users.

The dependency on both altruistic and paid servers is also a concern, as data loss can occur when such servers become disengaged abruptly. Additionally, cloud providers are not necessarily a better alternative than current OSN providers with regards to user privacy. Dropbox, for instance, was also involved in the PRISM program and has been criticized for other privacy breaches [162]. Further, in Diaspora, since users are not able to encrypt their data, full privacy of data is also not achieved; the danger of misusing user data is rather shifted from one central provider to several quasi-central providers (e.g., server administrators in Diaspora).

Common among all the approaches are further drawbacks, especially with regards to malicious users. If a user stores her data at a dedicated server, this server might suffer from *Distributed Denial of Service* (DDoS) attacks at any time. None of the state-of-the-art systems provide means to react to such an attack. One exception is Vis-à-Vis, in which a user could boot additional VMs to manage the increased load, but the procedure itself is undefined and would also result in higher monetary cost. At the same time, these solutions are not adaptive in the sense that a user, who wants to contribute more than her own machine, is not able to do so easily.

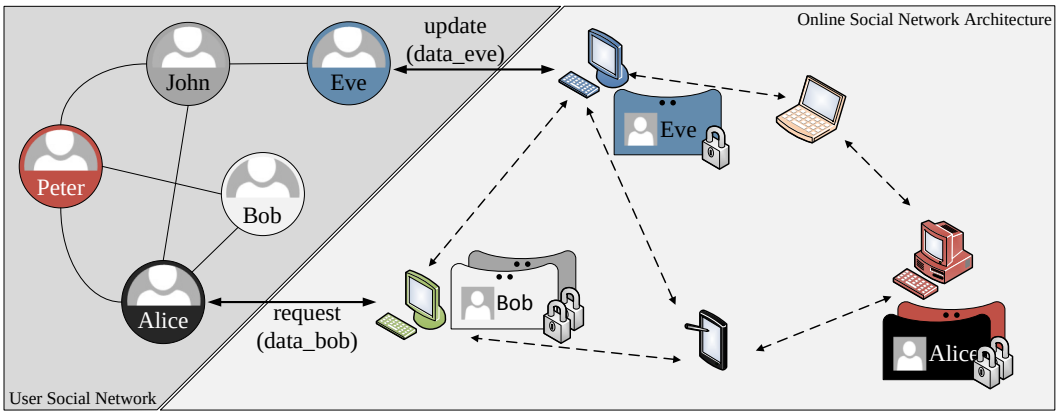


Figure 4.2: A decentralized OSN that exploits user cooperation. The central provider is removed and users provide each other temporary storage to host their data, as in, e.g., PeerSoN. In this example, Alice requests Bob's profile from another, possibly unknown user, as Bob himself is not online. Eve updates her data at her own machine.

4.2.2 Solutions Built on the Cooperation of Users

The second category of approaches lets users cooperate and provide temporarily available storage to each other [40–46], as shown in Figure 4.2. With the mutual cooperation of nodes and flexible data storage locations, users can be independent of dedicated servers and their drawbacks. Moreover, as the OSN functions with resources that are exclusively contributed by users, it can operate without additional costs. Due to the lack of dedicated servers, the major challenge of this approach, however, is to provide a storage substrate with high data availability to the users, while limiting the overhead induced by that substrate.

The earliest approach into social cooperation was Friendstore [40], an online backup system for user data in which users store data at their friends' machines, which achieved high data availability in a real-world deployment. However, since Friendstore is a storage substrate only, it lacks any OSN functionality. Further, recent studies suggest that the online time of OSN users is much less than assumed in Friendstore [82, 156]. Ultimately, this would lead to a reduction of data availability. Finally, users can only store their data at friends' devices. Thus, as many OSN users only have few friends (see Chapter 2.1.1), their data will suffer from low availability due to a lack of suitable storage locations.

PeerSoN proposes an optimized node selection algorithm in which nodes with mutual agreements store data for each other [41, 42]. Here, the mutual agreements are formed

between nodes with similar online times in a tit-for-tat manner. That is, nodes with high online times will match with other high performance nodes, and low performance nodes will team up with other nodes with little online time. The main issue of this approach is its inability to construct a robust OSN. Users with an online time of less than eight hours a day achieve less than 90% data availability. Given the power-law distribution of online time in OSNs [82, 156], the majority of users will thus be unable to achieve high levels of data availability. The outcome is a frail OSN where even highly contributing users may not be able to find data they want.

Cachet [44] replicates the data of users within a DHT. While the data replication feature of the DHT ensures high data availability, it also increases the communication overhead between nodes. As OSNs usually experience high churn rates [82], data often has to be transferred from departing nodes to other DHT members. This is particularly the case for mobile nodes. Also, Cachet does not minimize the number of replicas, which can increase the overhead to keep all replicas of a user's data up-to-date.

Similar to Friendstore, but designed as comprehensive OSNs, Safebook [43], MyZone [45] and ProofBook [46] mirror each user's data at a subset of their direct friends. Unfortunately, as in Friendstore, in those systems a user depends on her social contacts for data storage, as she needs enough suitable friends that qualify as a mirror (in MyZone a mirror is even more trusted than a friend). This is difficult for many users in OSNs who maintain few social relations [75]. As a consequence, such systems typically achieve low data availability rates (e.g., 90% in Safebook and MyZone). Additionally, in Safebook the data is only accessible through a path of so called shells. The mirrors form the innermost shell, friends of mirrors form the second shell, and so on. Retrieving a node's data requires traversing the shells along the path of nodes that befriend each other. Hence, *all* nodes on the same path towards the innermost shell need to be online simultaneously in order to provide access to the data, which is unlikely considering the typical online time patterns of OSN nodes [74].

Finally, some additional deficits exist across all approaches, with some exceptions to that rule. First, none of the above schemes *explicitly* consider mobile (i.e., smartphone) devices, which have become one major way of using OSNs. In cooperation schemes some tasks (e.g., maintenance of a DHT) can be difficult for mobile devices with limited bandwidth capacity and energy resources. Second, both defense algorithms against any kind of attack and adaption mechanisms for extended user contributions are missing. One slight exception is MyZone, which considers some specific attacks in its design, but does not efficiently prevent a DDoS attack (see Chapter 6 for details). Third, with the sole exception of the optimized approach of PeerSoN, all other solutions are not making an effort to limit their storage and communication overhead.

4.2.3 Hybrid Systems

Some DOSN solutions try to combine elements from both above categories. Confidant [47] lets users cooperate to provide each other with storage space for their data, while the name resolution service for the data is built by using cloud services. Thus, Confidant requires a lower level of cloud interaction than Vis-à-Vis, as the (paid) cloud service only runs the name resolution service for data that is otherwise stored on user machines. Still, a monetary effort is required by the user. At the same time, data availability from storing the data on user machines tends to be low for weakly connected users, as Confidant also requires these machines to be trusted (i.e., friends).

SuperNova [48] relaxes the dependency on dedicated servers, but builds on the existence of so called super peers, i.e., nodes with increased resources. These super peers act as a replacement for the dedicated servers, and thus have to implement a variety of functionality to be used by the remaining users, including maintenance of the directory of the network and acting as a storage node for others. At the same time, the regular nodes in the network are supposed to cooperate by acting as *storekeepers* for each other, i.e., to mirror each others data. However, SuperNova does not provide any information on how to choose storekeepers, and suffers from low data availability similar to Safebook or MyZone. Additionally, the super peers are supposed to be economically motivated as well. Finally, SuperNova does not provide any encryption means, so that super peers and storekeepers can always inspect the data other users store at their facilities.

4.3 Summary

A summary of state-of-the-art DOSNs is given in Table 4.2. Here, each approach is listed with regards to its strengths and weaknesses as discussed above with the goal of providing a clear overview of the profileration of DOSN solutions. The main finding is that each investigated system has deficiencies in multiple properties, which would be important to fulfill in order to enable the adaptation of the DOSN by a critical mass of users from centralized OSNs. As a consequence, a competitive DOSN is currently lacking.

On the one hand, systems, which rely on permanently available resources can enable high availability for encrypted data with low overhead. However, their main unsolved challenge is to provide these features in a technically and economically feasible fashion, without depending on some sort of (paid-for) centralized entity as, for instance, cloud services or web servers.

On the other hand, systems, which exploit cooperation among users are able to mitigate

Table 4.2: A summary of the state-of-the-art DOSNs.

Category	Approach	High Data Availability	Robustness	No Dependency	Low Overhead
Permanent Resources	Persona [36]	✓	✓	✗	✓
	Diaspora [37]	✓	✓	✗	✓
	Vis-a-Vis [38]	✓	✓	✗	✓
	Contrail [39]	✓	✓	✗	✓
User Cooperation	Friendstore [40]	○	✗	✓	✓
	PeerSoN [41,42]	✗	✗	✓	✓
	Cachet [44]	✓	✗	✓	✗
	Safebook [43]	✗	✗	✓	✗
	MyZone [45]	✗	✗	✓	✗
	Proofbook [46]	✗	✗	✓	✗
Hybrid	Confidant [47]	✓	✓	✗	✓
	SuperNova [48]	✗	✗	✗	○

Category	Approach	Adaptivity	Resiliency	User Data Privacy	Mobile Aware	No Cost for Users
Permanent Resources	Persona [36]	✗	○	✓	-	✗
	Diaspora [37]	✗	○	✗	-	✓
	Vis-a-Vis [38]	○	○	○	-	✗
	Contrail [39]	✓	○	✓	✓	✗
User Cooperation	Friendstore [40]	✗	✗	○	✗	✓
	PeerSoN [41,42]	✗	✗	✗	✗	✓
	Cachet [44]	✗	✗	✓	✗	✓
	Safebook [43]	✗	✗	✓	✗	✓
	MyZone [45]	✗	○	✓	✗	✓
	Proofbook [46]	✗	✗	✓	✗	✗
Hybrid	Confidant [47]	✗	✗	○	✗	✗
	SuperNova [48]	✗	✗	✗	✗	✓

Legend: ✓ feature supported; ✗ feature not supported; ○ feature partially supported; - feature not applicable

these challenges, but have limited success in providing high and robust data availability without introducing a lot of overhead, and do not consider mobile devices.

Moreover, hybrid systems, while trying to extract the best from both worlds, in fact suffer from their combined drawbacks as well. For instance, in Confidant, the usage of cloud services as lookup service still introduces a monetary cost, while the data storage at user nodes is still problematic with regards to availability.

Finally, arching over all existing solutions resiliency and adaptivity is lacking. The reason for shortcomings in these features is mainly that providing a storage substrate for a DOSN is already challenging, and researchers have focused on providing this substrate for a normal operation until now. Thus, both security (besides the privacy of user data) and adaptivity have—although important—not been considered as issues yet.

Chapter 5

Challenges for Decentralized OSNs

Linking the observations of Chapter 3 with the results of the study of the state-of-the-art in Chapter 4 shows that while there is a substantial need for decentralized OSNs, a comprehensive solution is currently lacking.

This chapter lays the foundation for the design of such a solution as it clarifies the challenges which a DOSN must address to emerge as a comprehensive alternative to current centralized OSNs. The discussion of these challenges benefits from the analysis and discussions in both Chapter 3 and Chapter 4 by considering the opportunities of decentralized online social networking and by learning from the drawbacks of existing works.

Contents

5.1 Robust Data Availability without User Payments

A primary challenge for the new DOSN is to achieve a level of data availability that is very close to that of centralized OSNs while decentralizing the control over the data and its storage. In particular, when removing the central provider, its sophisticated communication infrastructure is also no longer available. As discussed in Chapter 4, there are several options to build the substrate for that infrastructure.

The first option is to rely on permanently available resources. However, as previously argued, taking such an approach would (i) result in a dependency on the resource provider (e.g., a cloud provider as Amazon); (ii) require to obtain enough trustworthy storage locations; and (iii) incur costs on the users. As a consequence, a large-scale dissemination of the DOSN would be unlikely, as it requires the acquisition of a critical mass of users. The same reasoning can be applied to hybrid systems, where a dependency on super nodes or cloud providers still exists.

As a consequence, a DOSN that exploits the cooperation of users emerges as the better choice. However, the design of such an approach is more difficult:

In leveraging resources of participating nodes, the new approach must address the significantly differing characteristics of OSN users. First, users run diverse hardware configurations. While desktop devices are still a major way to use OSNs, social networking on mobile devices has become much more popular in recent years. For instance, Facebook was accessed by almost equal amounts of mobile and desktop users in June 2014 [7]. Second, the online time patterns of OSN nodes can differ substantially from each other. OSNs usually experience a power-law distribution of online times in OSNs [74, 82, 156]. As a result, the majority of users are seldomly online. Further, once users are online, they remain online for a short time only, i.e., their sessions are usually short and bursty [74, 82, 156]. Hence, a major challenge of a DOSN building solemnly on the cooperation of users is that it will also have to deal with churn among possible storage locations.

Moreover, the DOSN must recognize that these storage locations are usually not servers and only offer limited storage capacities. Thus, even though storage is a relatively cheap resource, the DOSN must use the capacity each node supplies efficiently. Further, as a node exhausts its capacity, it must be able to decide which data to keep and which to drop.

Finally, while facing these challenges, it must scale to the dimensions of OSNs and be easily deployable in large-scale scenarios. The latter requires a quick convergence to a stable system state; even when many nodes join the system at the same time and each node has little information to begin with, SOUP must provide effective means to quickly reach high data availability to each joining node.

5.2 Low Overhead

Achieving high data availability in an unfavorable scenario as described above is difficult, especially as the DOSN has to limit the overhead which is introduced by maintaining the DOSN in the absence of a central controller.

The overhead introduced is mainly two-fold. First, there is a possible *storage* overhead caused by the absence of dedicated servers. Because most non-server devices are not constantly powered on and thus not always connected to the DOSN, user data has to be replicated across multiple storage locations to enable high data availability. Second, each added replica of user data also requires maintenance, which introduces further *communication* overhead between users. As an example, a user that uploads a new photo album to the DOSN will have to distribute this album to *all* of her storage locations.

Another major challenge is thus to keep the number of storage locations low to ensure that there is no excess in both storage and communication overhead.

5.3 Adaptivity

Although the system must be able to achieve high and robust data availability with low overhead by using user machines only, it should also be open to the opportunities of altruistically provided resources and exploit them if they become available.

The most preferred resource selfless users may provide is probably highly available storage with high capacity. For instance, some users could decide to provide storage on servers they are operating anyway. At the same time, smaller contributions to the resource pool (e.g., a desktop machine intentionally left online for a relatively long time by its owner) should also be exploited.

5.4 Resiliency

While the new approach *should* exploit altruistically provided resources, it *must* be resilient when facing more unfavorable situations as well. That is, its performance must not significantly suffer even if resources in the network become unavailable.

For instance, a DDoS attack can disable a large number of machines, or a targeted DoS attack can be launched at the storage locations of a particular user to make her data unavail-

able. Both types of attacks have been investigated thoroughly by the research community, but—as attackers evolve—still constitute major problems today [163]. Similarly, network failures can also reduce the resources available for the DOSN and need to be handled. As a consequence, the new approach must be resilient in the sense that it has to be able to dynamically adjust to these scenarios.

However, not only external influences such as DoS attacks or network outages pose a problem for OSNs. In particular, OSNs have recently been infiltrated from the inside by an increasing number of fake identities [51]. Many such fake identities are often orchestrated by a single attacker, who uses them collaboratively in a Sybil attack inside the OSN for various purposes. For instance, Sybils may try to rig the outcome of a voting system by outvoting honest users [164]. A new DOSN must also pay respect to such an attack in its design.

5.5 Data Privacy

While current, centralized OSN platforms are—due to their sophisticated, redundant infrastructure—usually well equipped to defend against external attacks, their major problem is that they do not grant privacy for user data, but rather tend to collect, analyze and misuse the data as discussed in previous chapters of this thesis. Simply removing the central provider and shifting the data to distributed storage locations across the OSN does not entirely prevent such a behavior by parties interested in user data, as these parties could set up powerful storage facilities for DOSN users to regain access to user data.

Hence, a new DOSN must provide users with means to hide the contents of their data from others, including the storage location owners themselves. User content in OSNs is often targeted at a specific audience only. For instance, while a user might decide to share her vacation pictures with her close friends, she does not want her work colleagues to see those pictures as well. At another time, she might want to share a photo of the last company meeting with her colleagues, but not with her family. As a consequence, the new system must allow each user to control access to *every single* data item, and to do so on a very fine-grained basis.

5.6 Mobile Awareness

OSN users have been using social applications from their mobile phones with increasing intensity (see Chapter 4). In addition to acknowledging that not every device used in the

network is a device with sufficient resources in terms of storage, mobile devices also have other limitations. For instance, they experience an even higher churn rate than stationary devices (including laptops), suffer from limited connectivity in terms of bandwidth, and are often restricted in using that bandwidth due to expensive data plans.

In contrast to existing works, a full-fledged DOSN must therefore also be able to distinguish between such devices and others that are not subject to those limitations. In particular, when designing the system, a careful consideration of mobile devices is necessary to not only prevent an overburdening of these devices, but also to increase the stability of the system itself.

5.7 Genericness

One of the big opportunities of decentralized online social networking is to remove the need for each user to maintain one set of data for each social networking application (see Chapter 3). Thus, a new DOSN should also be generic in the sense that it allows a multitude of applications to operate on a single set of shared data.

Moreover, these applications should be allowed to be diverse in the sense that they, while operating on a single set of shared data, can implement arbitrary functionality. In particular, they should not be limited to current OSN functionality. Instead, to ensure the DOSN's future competitiveness, the functionality of the DOSN and its applications has to be extensible.

5.8 Exploitation of Social Relations

So far, the design of a DOSN seems to be subject to a lot of challenges. However, there is also help at hand. The new solution can and should exploit the potential of social relations within the OSN. In fact, multiple applications have leveraged these relations to improve their performance (e.g., [76, 165, 166]).

Also, some research suggests that social relations can be the usually required incentive for self-interested nodes to cooperate with other nodes in an OSN and discourage freeriding (recall that users need a robust OSN, see Chapter 4) [167]. If utilized properly, such cooperation can, for instance, help every user to distribute her data within the OSN more efficiently, or to defend against certain attacks.

However, to not violate the requirement of robust data availability, such cooperation must not hinder users with weak social connections from obtaining high availability for their data. Instead, the data of *every* user must be highly available in the OSN at all times. Also, the cooperation must not be influenced by possibly manipulative attackers.

5.9 Summary of Challenges

In summary, the following challenges have to be addressed successfully to provide a novel, full-fledged DOSN:

(i) the DOSN must provide data availability for *all* users without requiring user payments;

(ii) it must do so without introducing excessive overhead;

(iii) the new solution must be both adaptive to additionally provided resources and resilient against malicious activity;

(iv) each user must have a way to grant fine-grained access control to her data, while excluding illegible parties from accessing it;

(v) sharing of the encrypted data by multiple independent applications with arbitrary functionality should be enabled;

(vi) the limited capabilities of mobile devices must be considered; and

(vii) the opportunities of exploiting social links between users of the OSN should be exploited, if possible.

Chapter 6

Defending against the Sybil Attack

The resiliency against malicious users has emerged as one of the critical challenges for the construction of a new DOSN. Defending against DoS and DDoS attacks is still an unsolved problem [163], which is why future DOSNs need to be able to alleviate such attacks to a certain extent themselves.

Moreover, the research community has lately observed another manipulative strategy—the *Sybil attack*—in many real-world networks, including OSNs [51, 53, 168]. Here, the adversary forges multiple identities in order to compromise a target system. In particular, when considering DOSNs, each user could be a Sybil herself and aim at breaking the system. As a consequence, various attacks on the DOSN may be possible, especially when each user's actions or opinions are influential.

However, due to the potentially critical effect of the attack, researchers have investigated defense schemes against it, many of them based on OSNs [56–63]. The new DOSN could thus delegate the control of Sybil identities to one of these defense solutions, if that solution can efficiently protect OSNs against Sybils. On the contrary, if none of the systems can be applied to (D)OSNs, the system itself needs to be Sybil-resilient.

This chapter thus investigates state-of-the-art Sybil defense schemes to determine their eligibility for preventing Sybils to affect the operation of both OSNs and DOSNs. In particular, it answers the following questions:

(i) What are possible Sybil defense solutions for (D)OSN?

(ii) What is the scenario that these solutions would face in (D)OSN?

(iii) How do they perform when facing this attack scenario?

(iv) Ultimately, can they protect a new DOSN from Sybils?

Contents

6.1 The Sybil Attack and OSNs

In the past decade, the research community has extensively studied a new kind of malicious behavior. In the *Sybil attack*—the term Sybil originates from a pseudonym for a patient with multiple personality disorder—the attacker forges multiple identities in order to subvert a target system [164].

Sybil attacks are not only a threat in theory, but have also been observed in many real-world networks, including Ebay and OSNs [51,168]. In OSNs, Sybil attacks—next to DDoS attacks—are a major menace as they are used for spam or malware propagation [54]. Matters get worse when considering DOSNs built on user cooperation, as each participating user could be a Sybil herself and attack the cooperation schemes. Here, the attack schemes can be diverse, ranging from manipulating recommendation schemes to maliciously exploiting friendship relations of the target DOSN. While existing user-cooperation DOSN solutions do not acknowledge such attacks and are thus not constructed to be Sybil-resilient, a novel DOSN approach must be able to tolerate the attack.

At the same time, the Sybil attack, due to its pervasiveness, has also attracted researchers to build defense schemes against it. Among the various schemes that researchers have proposed to defend a target system, the most popular approaches in recent years leverage the OSNs of users in the target system, and inspect the structure of their social relations [56–63].

The main hypothesis of the existing defense schemes is that OSN identities controlled by the attacker will have difficulties establishing social relations (i.e., edges in the OSN graph) with honest users. These approaches reason that there must be some sort of trust between both ends of a social relation, which should not be the case for relations between honest users and forged identities. Thus, although there may be many relations both among the Sybil identities themselves and among the benign users themselves, there should be very few connections from Sybils to the community of benign users. These isolated links, i.e., links between a Sybil node and a benign node, are called *attack edges* [97]. As a result, the social network graph will have a clear partition between a Sybil region and a non-Sybil region, except for the few attack edges between them. In other words, there is supposed to be a *small cut* between both regions, as removing a small number of attack edges would result in two disconnected graphs [166]. To summarize, OSN-based Sybil defenses have been predominantly investigating the following scenario:

Although an attacker can create an arbitrary number of Sybil identities, she cannot establish an arbitrary number of attack edges to the non-Sybil identities.

However, as described in detail in Section 6.3, recent research has identified a rich set of behaviors of both attackers and honest users that invalidate the above hypothesis. In fact,

attackers can easily create links to benign users by simply sending out link-establishing requests (e.g., friendship requests on Facebook). The success rate can reach an astonishing 90% for specifically forged profiles or engineered bots [51,52,54]. At the same time honest users can also be easily tricked to establish the link and even initiate communication with forged identities [64, 65]. In addition, contrary to what all Sybil defense approaches have suspected, Sybils do not create numerous links mostly between themselves and thereby form a dense Sybil community; instead, almost 75% of links originating at a Sybil are connected to honest users and thus attack edges [51]. Therefore, the scenario that matches the assumptions made by current defense approaches is referred to as the *classical scenario*, while a *modern scenario* emerges:

Rather than predominantly connecting with other Sybils, an attacker is able to establish an increasing amount of social relations to benign users, and becomes more and more integrated within the community of benign users.

Hence, it is uncertain how well all existent OSN-based Sybil defense solutions, which were designed for the classical scenario, would perform under the new, modern scenario. While it is critical to have effective Sybil defense solutions for building a new DOSN, it is unclear what help and how much help can be obtained from existent solutions. In this chapter, the focus is to systematically analyze, measure, and compare how well or inadequate existent OSN-based approaches perform, with the goal to determine whether or not any approach might be applied to protect a DOSN from a Sybil attack. In doing so, this chapter qualifies and quantifies the strengths and weaknesses of these approaches under both the classical scenario and the modern scenario.

This thesis investigates *two* classes of Sybil defense approaches: *Sybil Detection* (SD) approaches—which try to detect Sybil nodes and exclude them from participation in a target system, and *Sybil Tolerance* (ST) approaches—which try to limit the impact of Sybils present in the system. The former includes SybilGuard/SybilLimit [56, 66], SybilShield [63], SybilInfer [59], SybilDefender [62], GateKeeper [58], and SybilRank [57]. The latter includes Ostra [8] and SumUp [60]. Given that a Sybil node may obtain more attack edges than traditionally assumed, the analysis in this thesis pays particular attention to what a Sybil node has to achieve in order to make itself indistinguishable from honest nodes—and thereby disguise itself from the defense scheme. Different attack strategies are investigated where applicable, and for every Sybil defense solution the cost for the attacker (e.g., the number of attack edges to create) to thwart the solution is quantified.

The main finding is that current OSN-based Sybil defense approaches of both classes have difficulty identifying the attack edges and the Sybil nodes in the modern scenario. Although they perform well in the classical scenario, surprisingly little effort is needed to deceive any existent defense scheme. Specifically, in many schemes a Sybil node only needs to create one or two attack edges to random honest nodes in order to successfully masquerade as

a benign node. The attacker can further reduce the required effort if she follows more intelligent attack strategies that exploit particular weaknesses in a given defense scheme.

As a consequence, a new DOSN cannot efficiently detect or tolerate Sybils by relying on one of the existing schemes. It rather needs to actively protect itself against a Sybil attack.

6.2 Related Work

Prior to the survey presented in the following, there have been other studies analyzing OSN-based Sybil defenses. In the work closest to this thesis' analysis, Viswanath et al. revealed that Sybil detection schemes could be abstracted to community detection algorithms [166]. In another study, Yu provided a concise summary of Sybil defenses existent prior to that study, and described their working principles under the classical scenario [97]. Viswanath et al. further explored the design space for OSN-based defenses [169]. Finally, Boshmaf et al. presented a framework for the evaluation of graph-based Sybil detection [170].

This thesis substantially differs from such studies: First, the performance of Sybil defenses in a completely new scenario, i.e., the modern scenario (Section 6.1) is analyzed. For instance, research in [166] requires Sybils to reside outside a distinguishable community, which is not true in the modern scenario. Second, this thesis includes more recent defense schemes that were not studied before. In particular, it studies approaches that can handle *modular* OSN graphs [57], which did not exist at the time of previous studies. Third, to determine what conditions cause each defense system to fail, not only qualitative analysis is provided, but also a *quantitative* approach to providing concrete results is taken. This complements studies such as [170], which do not analyze particular approaches, but rather develop guidelines on how they should be evaluated or designed.

Besides research focusing on the analysis of Sybil defenses, some works further rethought social graph properties with regards to Sybil defense. Alvisi et al. [171] pointed out that Sybil defense schemes tend to assume a minimal cut between Sybils and honest users (see Chapter 2.1.1.2), which however may not hold. The authors also analyzed SybilLimit [56] and show the sensitivity of the scheme to the existence of this cut. The analysis of this thesis goes one step further: It provides an in-depth analysis of *all* relevant Sybil defense approaches and their performance. Unlike Alvisi et al., it also considers Sybil tolerance approaches (a listing and categorization of the subjects under investigation is given in Section 6.4), whose working principles greatly differ from those of Sybil detection approaches [169].

Researchers have also attempted to alter social graphs to help Sybil defense schemes work better. Mittal et al. proposed introducing noise to a social graph to hide its true structure

to attackers [172]. However, doing so only hides the exact structure of the graph from the attacker, but does not constrain her in creating attack edges.

6.3 Revisiting Assumptions for Sybil Defenses

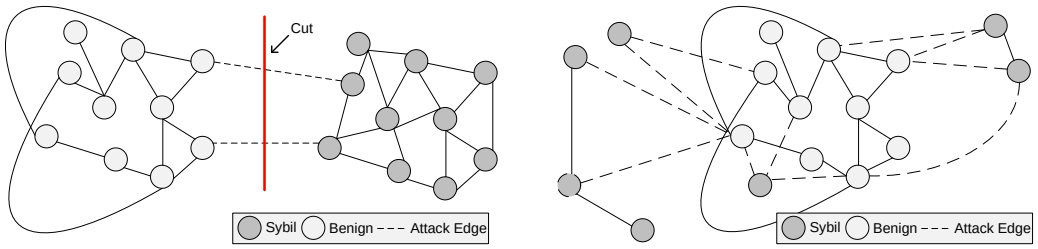
This thesis emphasizes that attackers employing a Sybil attack operate differently than previously thought. The following section explains *why* a new attacker behavior is proposed and *what* this behavior looks like, and thereby lays the foundation of the in-depth study reported in the later parts of this chapter.

6.3.1 Troubling Observations

Recently, researchers have discovered a multitude of behaviors of both honest and Sybil nodes that have a significant impact on the structure of a social graph. These behaviors can be observed across different OSNs, independent from their working principle and graph structure [51, 54, 64, 65]. The most salient observation is that a large fraction of OSN users are credulous and attackers are easily able to create links to honest nodes. Yang et al. found close to 11 million attack edges distributed among roughly 65,000 Sybil nodes—an average of 170 attack edges per Sybil [51]. Attack edges can be created by simply sending out link-establishing requests in OSNs (e.g., friendship requests on Facebook). The success rates of this simple attack range between 26% and an astonishing 90% of requests for specifically forged profiles or engineered bots [51, 52, 54].

Moreover, Bilge et al. show that 50% of users with whom the attacker could create an attack edge also clicked on links which the attacker sent in a message [54]. This implies that honest users even *trust* the forged profiles to a certain extent—even though the content of the message is possibly malicious (e.g., a link to malware).

It is also easy to trick benign users into sending link-establishing requests to Sybils [64, 65]. Using simple attacks (e.g., manipulating the friend recommendation scheme on Facebook), an attacker can obtain hundreds of friend requests *per day* with a single fake profile. Sybils are thus able to obtain thousands of attack edges to benign users without initiating any contact. Furthermore, once a Sybil has established some attack edges to honest users, it will be recommended to more users due to the existence of mutual friends, thus increasing the number of attack edges constantly. Sridharan et al. show that spammers on Twitter can effectively trick benign users into following them, i.e., creating a link to attackers [65]. Even with a simple attack, one third of the spamming accounts could accumulate more than 100 honest followers.



(a) Classical Scenario (b) Modern Scenario

Figure 6.1: Juxtaposition of Scenarios. In (a), a clear distinction of the Sybil region and the honest region is possible, whereas such a distinction has become difficult in (b).

Further, what has changed is not only how Sybils are able to interact with honest users, but also how many attack edges Sybils can create. In fact, according to [51], almost 75% of links originating from Sybils are connected to honest users, instead of connecting to other Sybils as once thought, meaning the Sybil community structure is not as densely connected as thought before.

As a consequence, Sybils can be observed in a multitude of OSNs, ranging from regional OSNs like the Chinese network RenRen [51] and subscription-based OSNs like Twitter [65] to the world's largest OSN, Facebook [54, 64].

6.3.2 Modern Scenario versus Classical Scenario

As a result of these findings, this thesis proposes a new structure of the OSN graph, with the following characteristics: (i) the number of attack edges k increases; and (ii) Sybils create most links to honest nodes, not to other Sybils [51]. Figure 6.1 shows a juxtaposition of the original classical scenario (Figure 6.1a) and the modern scenario (Figure 6.1b). The modern scenario has two major distinguishing features:

1) No minimal cut exists. Figure 6.1a provides a clear distinction between a benign region and a Sybil region based on a small minimal cut in a social graph. Recall that a minimal cut of a graph is a cut whose cutset has the lowest number of edges among all cutsets (see Chapter 2.1.1.2). In the modern scenario, the existence of such a minimal cut between the honest and the Sybil nodes is no longer assumed (note that such cuts may exist between communities sparsely connected to each other [57]).

Some approaches abstract the existence of a minimal cut in a graph to the *mixing time* of

the graph [97]. Recall, that the mixing time indicates how fast a random walk on a graph G approaches the stationary distribution (see Chapter 2.1.1.2). Defense schemes then compare the mixing times of different subsets of G. If G contains a minimal cut between honest and Sybil nodes, it should have a slower mixing time than the subset of honest nodes, which is well connected [97]. On the other hand, if there is no minimal cut, the mixing time for G will then be faster. Applied to the modern scenario, the mixing times of different subsets of G are not easily distinguishable anymore.

2) There is not a single, densely connected Sybil community. In the classical scenario, Sybils form a single community, which is densely connected internally. Since Sybils create links to honest nodes in larger counts than to other Sybils, this dissertation advocates that Sybils do not form any specific community structure.

6.4 Analysis of OSN-based Sybil Defenses

In this section, the working principle and effectiveness of the well-known Sybil defense approaches are analyzed under the modern scenario. Both Sybil detection approaches and Sybil tolerance approaches are investigated. In particular, this section provides a qualitative analysis of the weaknesses of these approaches. A quantitative analysis and comparison is then presented in Section 6.5.

6.4.1 Sybil Detection Approaches

To identify Sybils in an OSN graph G, a primary methodology has been leveraging random walks of limited lengths on the graph (see Chapter 2.1.1.2), starting at an *a priori* known trusted node. The main idea is that Sybil nodes experience a *low reachability* from the trusted node since it is highly unlikely for a length-constrained random walk originating from the trusted node to cross one of the few attack edges. Instead, the random walk is expected to stay within the honest region with very high probability.

Different methods are employed in leveraging the random walk concept. SybilGuard/SybilLimit use *random routes*, a random walk modification that uses pre-computed random routing tables to determine the next hop of a random walk, and check if the random route from a verifier will intersect with that from a suspect node. SybilShield follows the same concept; in addition, it performs random walks to find agent nodes for the verification of the random routes. SybilInfer conducts random walks—called *traces*—from all nodes, where a particular *a priori* honest node computes the probability that a subset G' of G is composed of entirely honest nodes based on the mixing time of the subset. SybilDefender first uses

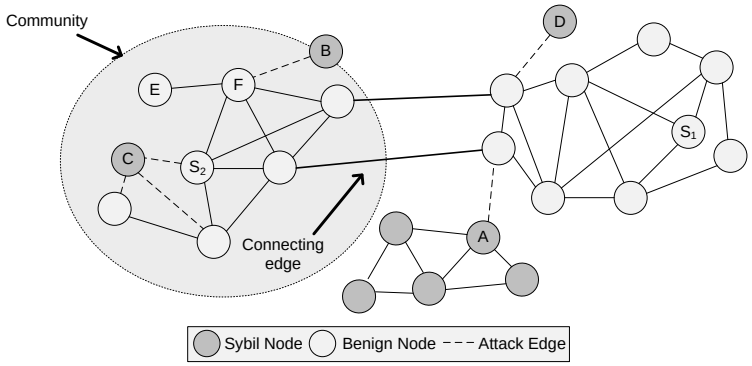


Figure 6.2: An exemplary graph with Sybil nodes attached to honest nodes in different scenarios. The honest nodes are organized into two communities.

random walks originating from an *a priori* known honest node to identify starting points of further random walks, which in turn identify certain Sybil nodes based on the frequency with which these walks traverse them. It then uses random walks again to detect additional Sybils in the neighborhood of the already detected Sybils. Finally, SybilRank relies on the landing probabilities of short random walks that start from a non-Sybil node.

The only exception is GateKeeper. Rather than using random walks, it employs a *Breadth First Search* (BFS) ticket distribution strategy to detect Sybils (see Chapter 2.1.1.3). However, like approaches using random walks, it also exploits the *low reachability* of the assumed Sybil region from the honest region. The idea behind GateKeeper is that since Sybils are not as well connected to the ticket sources, they will obtain substantially less BFS distributed tickets than benign nodes.

All these approaches were designed primarily toward the classical scenario. For instance, consider a random walk starting at node S_1 in Figure 6.2. The walk has a low probability of traversing the attack edge to the Sybil community attached to node A. In the following, the impact of the modern scenario on each of these detection schemes is discussed.

6.4.1.1 SybilGuard/SybilLimit

Because of the similarity of SybilGuard and SybilLimit (the latter is the successor of the former), these two approaches are studied together.

Working principle. In SybilGuard, an honest node (the verifier) accepts a suspect node only if the random route (i.e., the modified random walk) from the verifier intersects with the random route from a suspect node. If there are only a few attack edges, it is highly unlikely

that the random route from a Sybil will intersect with the route of a verifier, causing the Sybil to be unlikely to be accepted. SybilLimit modifies SybilGuard, where every suspect and verifier initiate multiple random routes. For a Sybil to be accepted by a verifier, it needs to come up with a random route intersecting with a random route of the verifier. Route intersections are no longer related to nodes on the routes, but to the tails (i.e., the final traversed edge before a route halts). Every random route is also shorter, leading to less possibilities for attackers to fake random routes that cross attack edges.

Effectiveness analysis. SybilGuard provides guarantees on the number of Sybils admitted per attack edge. If the number of attack edges k in a system with n benign nodes is at most $O(\sqrt{n}/\log n)$, SybilGuard will admit at most $O(\sqrt{n}\log n)$ Sybil nodes *per attack edge*. If however k increases, SybilGuard is not able to limit the number of Sybil nodes at all [56]. SybilLimit improves on SybilGuard to the point where it accepts at most $O(\log n)$ Sybils *per attack edge*, no matter how many attack edges exist. It does not suffer from SybilGuard's limitation that the number of attack edges must stay below a certain threshold to have an effect. Nonetheless, SybilLimit was designed for a particular scenario: A densely connected Sybil community connected to the honest region by only a *few* attack edges. In this (classical) scenario admitting $O(\log n)$ Sybils per edge is a good effort, since many more Sybils may be connected to the honest region by these edges only (see Figure 6.1a or node A in Figure 6.2). However, in an inversion of the argument, this also means that every Sybil that can obtain a single attack edge can be admitted with high probability.

6.4.1.2 SybilShield

Working principle. SybilShield extends the working principle of SybilGuard to also cover modular OSNs, i.e., OSNs that consist of multiple distinct communities. In SybilShield, a verifier does not immediately reject a suspect node if no intersections exist among random routes. Instead, the verifier will perform random walks towards nodes outside of its own community, and use the endpoints of these walks as its *agents*. The verifier will then still admit the suspect, if the suspect's random routes have enough intersections with the random routes of a number of agents.

As in SybilGuard, if Sybils are gathered in a distinct community, the probability for random routes from an honest verifier and those from a Sybil node to intersect is low. Also, it is unlikely that a Sybil node can obtain enough intersections with random routes from a number of agents. On the other hand, since verifiers can now reach out to multiple communities, the extension with agents allow well-connected honest nodes to become accepted by more verifiers.

Effectiveness analysis. SybilShield works as effectively as SybilGuard in its first step, as the approaches are basically the same. With the addition of the agent walks, SybilShield can even admit more honest nodes, while not admitting noticeably more Sybils. However, the principle only works if the number of attack edges is indeed small, and the Sybils are indeed gathered in a distinct community. When these assumptions are dismantled—as in the modern scenario—it might in fact be *easier* for a Sybil to be admitted than for a benign node. As Sybils are not bound to the community structure found in OSNs, they can attach their attack edges to multiple communities and become integrated into those. As a consequence, compared to an honest node that belongs to a single community, a Sybil can more easily create more intersections with the agents.

6.4.1.3 SybilInfer

Working principle: SybilInfer assumes OSN graphs to be fast-mixing, which is a dubious assumption in itself [56, 80]. The basic principle of SybilInfer is that with a small number of attack edges k—i.e., the existence of a small cut in G—the mixing time of G is slower than the mixing time of just the benign region of G. The reason is that traces originating from a benign node are more likely to remain in the benign region than to traverse one of the few attack edges and end on a Sybil node.

Effectiveness analysis: SybilInfer would work well if k is indeed small *and* the benign region is fast-mixing. However, if the Sybil nodes become more integrated into the graph, the cut will become less distinct (consider nodes B and C in Figure 6.2). The degree to which the cut is detectable also depends on the structure of the Sybil region. In SybilInfer, an attacker should not introduce many interconnected Sybils (e.g., node A in Figure 6.2), as otherwise all attacker nodes can be detected due to a slower mixing time of G. The attacker will be most successful in disguising Sybil nodes as benign nodes with a sparse community structure and many attack edges, an attacking behavior which is observed in the modern scenario.

6.4.1.4 SybilDefender

Working principle: In addition to the minimal cut, SybilDefender also assumes that the Sybil region is substantially smaller than the honest region. It works in multiple stages. In its first stage, it offers a set of algorithms to detect Sybil nodes. In the second stage, originating from these Sybils, it tries to further detect the supposedly existing Sybil community surrounding the previously detected Sybils. Both stages rely on random walks.

The first stage executes a number of random walks originating from an *a priori* known honest node and ending at a number of *judges*. In turn, the judges and the honest node each start a number of random walks of varying lengths in order to learn the threshold for identifying Sybils; in particular, they count how often each node is traversed by these walks—i.e., the *frequency* of a node. Then, for each suspect node, SybilDefender performs random walks from the suspect to determine if it is a Sybil. The key principle is that a long enough random walk from an honest node will traverse more different nodes than a walk that is trapped inside the small Sybil region. In other words, the frequency of nodes in walks originating from an honest node will be lower than that in walks originating from Sybils.

The second stage starts a series of random walks from a previously detected Sybil, and every random walk terminates once it reaches a dead end (i.e., a node whose neighbors have already been visited). It sorts the nodes traversed by all random walks based on their *frequency* into a list in decreasing order. From this list, it adds nodes one by one into the Sybil region (initially empty), until the conductance of the Sybil region is not getting smaller. The nodes on the list then constitute the Sybil region discovered from the second stage. Recall that the conductance measures the quality of the cut between a region and the rest of a graph, and is the ratio between the number of edges in the cut and the sum of the degrees of all the nodes in the region (see Chapter 2.1.1.1). The reasoning is that, in presence of the minimal cut, the walks will remain within the Sybil region, and nodes with the high frequency are very likely Sybil nodes as well.

Effectiveness analysis: In the presence of one honest and one Sybil community separated by a minimal cut, the random walks started from the judges will remain within the honest community with high probability. Hence, honest nodes will—compared to Sybils—obtain higher frequencies during the random walks. Also, the Sybil community can be detected in the same way when starting at a known Sybil node within that community. Thus, SybilDefender works well within the classical scenario and adds an efficient second stage to previous approaches. If, however, one of the assumptions does not hold (and neither of them holds in the modern scenario), SybilDefender's performance deteoriates. If there are multiple honest communities, the false positive rate will rise, as judges are likely to stay within one honest community only. Or, as Sybils rather attach themselves to different benign communities, not only does the detection step become ineffective, but the false positive rate will again increase. The reason is that starting random walks from a presumably identified Sybil node to detect its community will also cover a wide selection of honest nodes.

6.4.1.5 SybilRank

Working principle: Like SybilLimit and SybilInfer, SybilRank also employs random walks as a means to detect Sybil nodes. In particular, SybilRank uses *logn power iterations*

to determine the fate of a node. In short, power iterations are a succession of random walk transition matrix multiplications, which compute the probability that an early-terminated random walk would land on a node (for details see [173]). The higher the probability is, the more likely the node is benign.

The random walk starts at a seed inside a Louvain-detected honest community (see Chapter 2.1.1.1), and each major community in the OSN can have a manually chosen seed that is guaranteed to be benign. This method allows SybilRank to deal with the highly modular OSN graph structure. Under the assumption that there are few attack edges connecting Sybil nodes with an honest community, a short random walk is unlikely to finish at a Sybil node since the traversal of an attack edge is needed. Consider node A in Figure 6.2. As there is only one attack edge towards A, the random walk from seed S_2 is unlikely to land on any node in the Sybil community around A.

Effectiveness analysis: SybilRank heavily relies on the number of attack edges that Sybil nodes can establish (it admits $O(\log n)$ Sybils per attack edge), but is roughly independent of the Sybil community structure. Consider Sybil nodes A and D in Figure 6.2, where A is part of a Sybil community but D is attacking as a single node. Since a random walk only has one attack edge to traverse to reach either A or D, both A and D will obtain a low ranking, and are thus likely to be categorized as Sybil nodes.

If the number of attack edges k increases, random walks will be more likely to land on Sybils adjacent to attack edges, thus increasing the ranking of these Sybils and even mislabeling them as benign nodes. The Sybil node C in Figure 6.2, for example, is completely disguised in the non-Sybil community. In fact, when starting random walks at S_2, node C will receive one of the highest rankings due to its position in the graph.

Finally, the introduction of trusted seeds enables more sophisticated attacks. By putting one seed in each major honest community (e.g., S_1, S_2 in Figure 6.2), honest users should be ranked higher than Sybils since they are well connected to those seeds, whereas Sybils have to rely on few attack edges. However, if Sybils can create links to nodes near the seed, they can increase their rankings. Although the seeds themselves might be careful with accepting link requests, nodes near them may be not.

6.4.1.6 GateKeeper

Working principle: In contrast to all previously discussed approaches, GateKeeper does not (directly) leverage random walks for Sybil detection. Similar to SybilRank, a central authority (called *admission controller*) selects a number of seeds (called *ticket sources*). Whereas the selection of ticket sources is performed via random walks originating at the

controller, the actual Sybil detection algorithm operates differently.

Each ticket source obtains a number of tickets from the admission controller. In a *Breadth First Search* (BFS) approach the ticket source then distributes the tickets equally among its neighbors, and each neighbor again distributes the tickets equally among its neighbors, and so on (see Chapter 2.1.1.3). To be admitted into the system, a node must obtain tickets from a fraction of the ticket sources (e.g., a node is required to collect a ticket from 20% of all sources in [58]). The reasoning here is that if Sybils are only connected to the honest region by very few attack edges, they will rarely obtain tickets from the requisite fraction of sources, whereas honest users will easily do so.

Effectiveness analysis: GateKeeper is strongly dependent on the number of attack edges k, and accepts $O(\log k)$ Sybils per attack edge, improving from the magnitude of $O(\log n)$ of the previous schemes. If the Sybil community is well-connected among itself and there are few attack edges, the system will only admit $k \log k$ Sybils from the Sybil region. The more attack edges (i.e., the further k is increased), the more Sybil nodes will be admitted. In the case of the modern scenario, it becomes easier for Sybils to obtain the required fraction of tickets: each Sybil owns more attack edges and—similar to SybilRank—may be able to place them within a small distance to a ticket source.

6.4.2 Sybil Tolerance Approaches

In contrast to Sybil detection approaches, Sybil tolerance schemes aim at limiting the influence of Sybils that may reside in a system without necessarily detecting them. For instance, these Sybils could—if there are no countermeasures—flood a DOSN with spam data or manipulate its cooperation schemes. Another major difference between the two classes is that Sybil tolerance approaches are usually designed for a particular purpose, while Sybil detection approaches are meant to provide a more universal solution.

This thesis focuses on two major Sybil tolerance approaches, Ostra and SumUp, which may act as a countermeasure against both spam and vote manipulation in cooperation schemes. Both schemes are built upon a flow network where each link is granted a certain capacity (see Chapter 2.1.1.4), called *credit*. Nodes can exchange messages (Ostra) or cast votes (SumUp) only if they find a path with enough credit on every link of the path. Both schemes also feature feedback mechanisms in order to penalize links used for spam (Ostra) or bogus votes (SumUp).

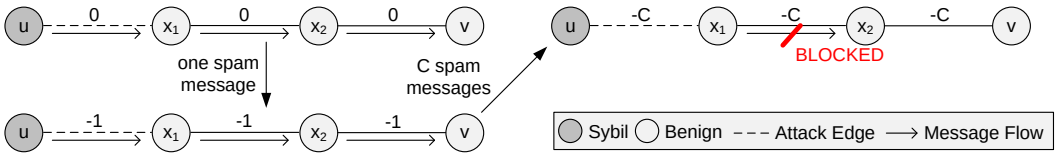


Figure 6.3: Credit reduction in Ostra due to spam. As one credit is reduced for every spam, eventually every link between u and v—including benign links—has no available capacity for message delivery.

6.4.2.1 Ostra

Working principle: Ostra aims to limit spam issued by malicious users in a social network. It provides two modes of operation. One mode which requires strong user identities (i.e., a user is guaranteed to have only one digital identity) and a second mode where this requirement is relaxed. As a DOSN should not enforce strong user identities (see Chapter 3), Ostra is studied with regards to the latter mode.

It assigns credits to links between users, where each link has two dependent credit values, one for each direction. If Ostra finds a path with available credit from a sender to its receiver, the sender's message can be sent; otherwise, the message will be blocked. As a message traverses along a path, starting at the source of the message, credit will be deducted from each traversed link in the direction of message transmission, while the same amount of credit is added in the opposite direction. If the recipient classifies the received message as wanted (i.e., not spam), Ostra will reverse the credit operations previously performed, such that only messages classified as spam will have an effect on the credit available on each link. The main idea limiting the influence of Sybils is that the classification of messages will quickly deplete the capacity on attack edges, leaving Sybils unable to distribute any spam afterwards.

As benign users may inadvertently send out unwanted messages from time to time, Ostra also provides a mechanism to *forgive* a link over which spam was sent previously. That is, Ostra increases the credit on each link periodically, even if it has been depleted before.

Effectiveness analysis: In Ostra, more attack edges will lead to more spam that Sybils can send. Additionally, the credit forgiving scheme effectively makes sure that an attack edge never dies completely. Furthermore, sending spam reduces not only the credit on an attack edge of a spammer, but also the credits on the entire path to the destination. For instance, in Figure 6.3, with each spam message sent from a Sybil node u to an honest node v that traverses the path $p = u, x_1, x_2, v$, Ostra will penalize benign edges (x_1, x_2) and (x_2, v) as well. If there is no other path, eventually it may not be possible for x_1 to send a regular

message to x_2 or v anymore, as the capacity C has been depleted. An intelligent attacker may therefore be able to isolate some honest nodes by depleting credits on many edges in the network. Ostra suggests honest users obtain "a sufficient number of trust links" [61] to reduce the probability of becoming isolated, which further lowers the bar for the attacker, as users are even more willing to accept link requests.

Even worse, the attacker could specifically target the few links that connect the communities in a highly modular OSN graph G. Consider Figure 6.2 again. Sybils A and D can block any communication towards the community to the left (located in the shaded area) if each can send their spam toward S_2 over one of the two links for reaching the community.

6.4.2.2 SumUp

Working principle: SumUp aims at limiting the number of bogus votes that Sybil nodes can cast in a system. It chooses an *a priori* trusted vote collector, which then distributes tickets in a BFS manner downstream along the OSN graph. Every node will keep one ticket, and distribute the remaining tickets equally among its next BFS-level neighbors. The capacity of every link is set to be the number of tickets distributed along the link plus one. SumUp defines all links with a capacity larger than one to be within the *voting envelope*. The main idea is to keep Sybil nodes outside of the envelope and limit the voting capacity per attack edge to one.

SumUp employs two main mechanisms to modify an OSN graph and reduce the impact of attack edges. Before distributing tickets, SumUp employs a *pruning mechanism* to reduce the number of attack edges available to Sybils (therefore limiting their attack capability) and speed up vote computation with fewer edges. Basically, the number of edges from a node at a BFS-level of $i+1$ can only have at most d edges going to BFS-level i, where the vote collector is at BFS-level 0. It also has a *feedback mechanism*: the vote collector can provide negative feedback to paths through which bogus votes have been cast. Once an edge has too much negative feedback, it is eliminated and no vote can be cast along that edge any more.

Effectiveness analysis: As in Ostra, the damage from Sybil nodes will be directly proportional to the number of attack edges. Moreover, more attack edges would lead to a higher likelihood for a Sybil to be close to a vote collector (e.g., Sybil node C in Figure 6.2 would be close to S_2). As nodes near the collector can issue more tickets, such a Sybil would gain an increased capacity. Furthermore, as the number of attack edges increases, SumUp's feedback mechanism may penalize links adjacent to benign nodes too, causing collateral damage to benign edges while penalizing attack edges (similar to Ostra). If there is a sufficient number of Sybils with a path through a benign node, it may even cause all edges of the node to be eliminated. To mitigate this negative effect, after SumUp removes certain

Table 6.1: The datasets used for evaluating Sybil defenses.

OSN	Nodes	Edges	Avg. Degree
Facebook	90,269	3,646,662	40.40
Synthetic	1,000	2,048	2.05

edges due to penalties caused by Sybil votes, it reintroduces pruned edges into the graph to replace the penalized edges in order to maintain the required number of incoming edges per node. The reintroduced edges, however, could be attack edges, leaving the efficacy of this feature questionable.

6.5 Are OSN-based Sybil Defenses Still Working?

After providing a qualitative analysis of the weaknesses of Sybil defense approaches under investigation in Section 6.4, in this section a quantitative study of these approaches is conducted. In particular, this study answers the following questions:

(i) How serious are the weaknesses of Sybil defense solutions?

(ii) Can OSN-based Sybil defense schemes still work?

(iii) What is the actual cost for the attacker (e.g., the number of attack edges to create) to thwart a defense solution.

6.5.1 Evaluation Methodology

All aforementioned Sybil defense approaches have been implemented and simulated to test their behavior when faced with different attack strategies. The simulations are based on both a real-world Facebook graph [174] and a synthetic graph as shown in Table 6.1. The degree distribution of the synthetic graph follows the OSN-typical power-law distribution (see Chapter 2.1.1), which can also be observed in the Facebook graph. Sybil nodes,which—as suggested in [51]—are sparsely interconnected are added to these existing graphs in a second step.

In evaluating every Sybil defense approach, the parameters of the original evaluation of each approach are implemented if possible. For instance, the evaluation of Ostra uses the same amount of messages sent in the system and follows the original evaluation with 93% of these messages sent to direct neighbors.

The parameters which are varied mainly include the number of attack edges and the strategy for placing attack edges. By default the attacker is assumed to place attack edges completely randomly, but for some approaches the attacker can place them close to specific nodes to gain certain advantages as explained in Section 6.4. To prevent biased results due to the specificity from one attack edge placement, for each parameter setting 100 different attack edge placements are simulated.

The evaluation of approaches which use any form of seed placement may require performing community detection beforehand. Following SybilRank, the Louvain method is used to detect communities in these cases. Recall, that the Louvain method iteratively merges two existing communities if a merge would result in a higher modularity of the graph with low computational cost (see Chapter 2.1.1.1).

Since Sybil detection and Sybil tolerance schemes have different goals and working principles, they are treated separately in their evaluation as well. To ensure that only the change in the structural properties of OSN graphs is evaluated, the attacker is *not* allowed to deviate from the protocol of an approach being studied. For instance, in SumUp, a Sybil node obtaining a number of tickets will not try to favor other Sybils but follow the SumUp design and distribute tickets further downstream, even if the recipients are honest nodes. The Sybils are also *not* allowed to be selected as seeds where applicable. The results are very similar for both datasets; for the ease of exposition results are reported for the Facebook set unless stated otherwise.

6.5.2 Sybil Detection Approaches

The goal for evaluating Sybil detection approaches is to find out what an attacker needs to achieve in order to disguise a Sybil node, i.e., to disable a detection approach from distinguishing between the Sybil node and honest users. Thus, there are two main questions which need to be answered:

(i) How many attack edges does a Sybil node need to establish in order to disguise itself as an honest node?

(ii) Does the location of Sybils on an OSN graph make a difference?

To determine whether a Sybil detection approach is able to distinguish between Sybils and honest nodes, both classes of nodes are compared with regards to their the relative performance in each detection scheme. In an ideal detection approach, all benign users should perform far better than all the Sybil nodes, thus leading to a clear distinction between both classes without any false positives or negatives. In the following, the ability of each scheme to differentiate between Sybils and honest nodes is called its *distinguishing ability*.

To illustrate the relative performance of benign users and Sybils, *Cumulative Distribution Function* (CDF) graphs are used for the majority of the SD approaches. Generally speaking, the further to the right a CDF curve appears, the better the nodes in this class perform. Thus, the CDF describing the Sybil nodes' performance would ideally be located on the far left in each graph, whereas the CDF for the performance of the benign nodes would ideally be at the far right. Then, the distinguishing ability of the defense solution would be sustained, as a clear distinction between the performance of Sybils and honest nodes is possible. However, the more both CDFs approach each other, the harder it becomes for the system to distinguish nodes in both classes. In the worst case, the Sybil CDF would 'overtake' the benign CDF, which indicates that Sybils performed better than honest nodes.

6.5.2.1 SybilLimit

The distinguishing ability of SybilLimit depends on the number of intersecting tails that a Sybil has with a verifier. Recall that in the original SybilLimit design, a Sybil node only needs *one* tail to intersect with that of a verifier in order to be verified. Preliminary experiments revealed that a Sybil can become verified with high probability if it can place one attack edge to a random honest node. This is not hard to achieve given that $O(\log n)$ Sybils could be admitted per attack edge (see Section 6.4.1.1).

However, with some modification, SybilLimit might still be able to distinguish honest nodes from Sybils by looking at the *number* of intersecting tails. In particular, the worst performing benign node might obtain significantly more intersecting tails with the verifiers than the best performing Sybil. In that case, SybilLimit might still able to work with different parameters. Therefore, it is more important to ask how many attack edges a single Sybil node would have to create, denoted as k, in order to be indistinguishable from a benign node in terms of the number of intersecting tails.

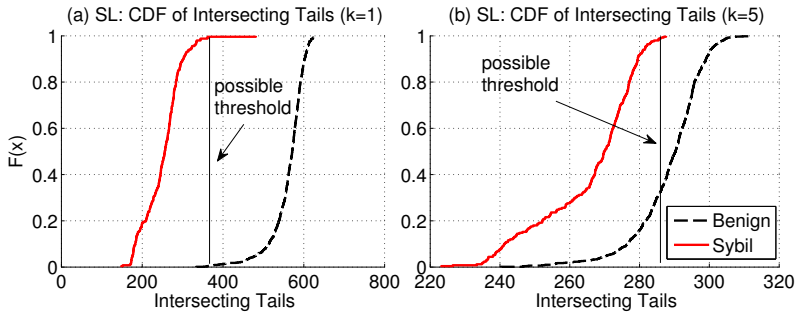


Figure 6.4: Performance of SybilLimit (SL). k is the number of attack edges per Sybil. SybilLimit is not able to detect Sybils with k increasing.

Figures 6.4a and 6.4b show the CDF of the number of intersecting tails with verifiers in SybilLimit. The attack edge parameter k increases from one attack edge per node up to a point at which SybilLimit is no longer able to distinguish both classes of nodes. When a Sybil node can obtain one randomly placed attack edge (Figure 6.4a), the distinguishing ability of SybilLimit remains good. However, as the number of attack edges increases, the distinguishing ability is reduced. The results are exemplified with 5 attack edges per node ($k = 5$) in Figure 6.4b. The main observation is that a possible admission threshold, which rejects the vast majority of Sybils, would also classify 30% of the honest nodes as Sybils. These results have to be seen in the light of recent discoveries discussed earlier in this section, after which attackers can gain hundreds of attack edges per day.

6.5.2.2 SybilShield

As SybilShield is based on random routes as well, the issues with SybilLimit also apply in principle. In particular, SybilShield's distinguishing ability also lies in the ability to differentiate between the numbers of intersections that Sybils and honest nodes achieve with the verifiers, respectively. As SybilShield does not consider tail intersections but rather route intersections—which are less difficult to obtain, as it is easier to obtain an intersection on an arbitrary node rather than on a specific edge—it already breaks at one attack edge per Sybil node (Figure 6.5a). Here, the number of intersections between verifiers and benign nodes is not clearly distinguishable from the intersections between verifiers and Sybils.

One distinguishing feature in SybilShield is the agent walk, by which additional nodes are selected to perform verification on behalf of a verifier. This should allow more honest nodes to be accepted even if their random routes reach communities without a verifier.

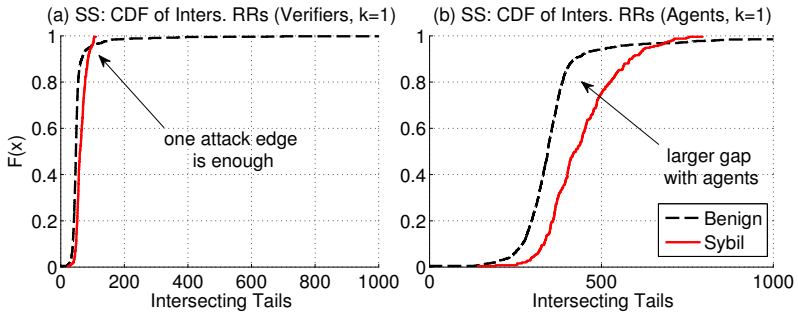


Figure 6.5: Performance of SybilShield (SS). k is the number of attack edges per Sybil. SybilShield is compromised with a single attack edge per Sybil (a), and its performance is worse in the agent phase (b).

However, as Figure 6.5b shows, compared with benign nodes, Sybil nodes can clearly obtain more random route intersections with these agents. The reason is that most benign nodes belong to a single community within the OSN, whereas Sybils will randomly attach their attack edges to possibly multiple communities, thus remaining in the reach of more agents.

6.5.2.3 SybilInfer

The distinguishing ability of SybilInfer lies within the landing probability of its modified random walk, i.e., the trace. Originating at a benign node, the vast majority of traces should end at another benign node—only then can gaps between the mixing times of different subgraphs be detected. Figure 6.6a shows the number of traces that end at benign and Sybil nodes, normalized by the number of benign and Sybil nodes in the system, respectively. All traces originate from a benign node, and therefore are called benign traces. As observed in SybilLimit, Sybils cannot obtain a sufficient amount of traces to end at a Sybil node with a single attack edge. However, Sybils succeed as more attack edges are added. As seen in Figure 6.6a, even with two randomly placed attack edges per Sybil node, i.e., $k = 2$, SybilInfer is no longer able to distinguish between benign and Sybil nodes, because more traces now end on Sybils than on benign nodes.

Another finding of this experiment is that when traces end at a Sybil, they do not concentrate at a few Sybils, but instead are widely distributed. If $k = 2$ and a trace starts from every benign node, altogether the traces can hit 75% of the Sybils. An equivalent amount of Sybils might be admitted.

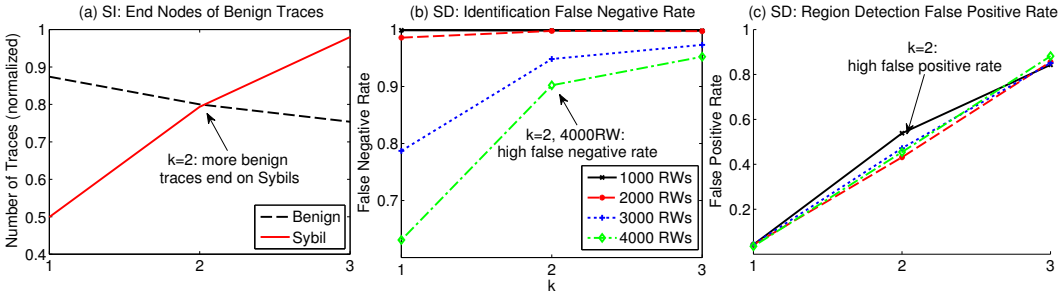


Figure 6.6: Performance of SybilInfer (SI) and SybilDefender (SD). k is the number of attack edges per Sybil. SybilInfer suffers from a low distinguishing ability if $k \geq 2$ (a), and the same is valid for SybilDefender, which exhibits high false negative rates (b) and false positive rates (c).

6.5.2.4 SybilDefender

The distinguishing ability of SybilDefender's first stage lies in its ability to detect deviations in the frequencies of nodes on the random walks originating from Sybils, which should be significantly larger than the precomputed frequency thresholds. In contrary, the frequencies of nodes on the random walks that start from an honest node should not deviate much. Figure 6.6b shows that, as the number of attack edges increases, SybilDefender's ability to identify Sybil nodes degrades. As a Sybil community does no longer exist and Sybils are more integrated in the honest communities, the random walks originating at a Sybil node result in a frequency deviation similar to those originating at an honest node. Hence, many Sybils are treated as honest nodes (*false negatives*). The severity of this issue is inversely correlated to the number of random walks originating at the judge nodes. By increasing the number of walks the judges execute, the average frequencies become more stable, and SybilDefender is more reliable in detecting Sybils. However, arbitrarily increasing the number of random walks from the judges is not effective, as the improvements in false negative rates rely on the small size of the cut, which is no longer given in the modern scenario. Even with 4000 such random walks, two attack edges per Sybil are more than enough to confuse SybilDefender.

Given the Sybils detected in the first stage, the second stage of SybilDefender identifies all nodes within the Sybil region. Figure 6.6c shows that as the number of attack edges increase, the number of honest nodes that are mistakenly added to the Sybil region also increases. The reason for that is that as Sybils become more integrated into the honest community the cut between the two regions becomes larger. Instead, as the cuts between honest communities tend to be small [87], SybilDefender could detect an honest community as a Sybil community.

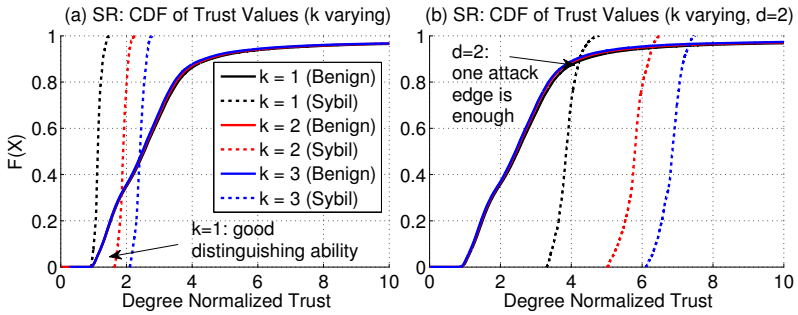


Figure 6.7: Performance of SybilRank (SR). k is the number of attack edges per Sybil. Whereas a random attack strategy requires two attack edges for a Sybil to disguise itself (a), a more intelligent attacker can reduce the effort to one attack edge, if she is able to place that edge close to a trust seed (b).

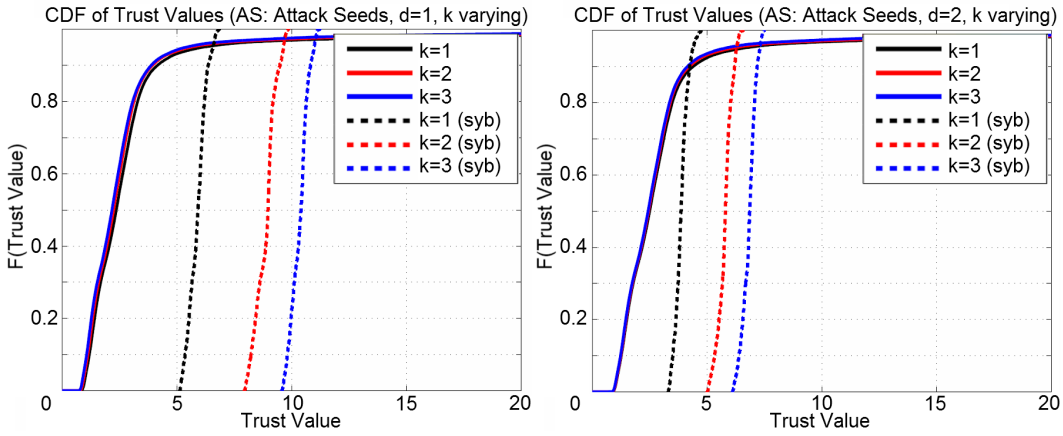
6.5.2.5 SybilRank

SybilRank distinguishes nodes according to their normalized trust ranking. The lower the ranking of a node is, the more likely it should be a Sybil. Thus, to determine the distinguishing ability of SybilRank, the difference of the rankings of benign nodes and Sybil nodes should be evaluated.

In its original evaluation, SybilRank places 50 seeds, with one chosen from ten nodes with the highest degree in the OSN and the other 49 randomly chosen [57]. Initial experiments showed that this strategy becomes increasingly flawed as the size of the OSN graph grows. The reason is that due to the modular structure of OSN graphs, in many cases the distance of the honest users to the seeds is larger than that of the Sybil nodes to the seeds, resulting in higher rankings of Sybils than many benign nodes. Therefore, to improve SybilRank's distinguishing ability, one seed is placed in each honest Louvain-detected community in another experiment, for which results are shown in Figure 6.7.

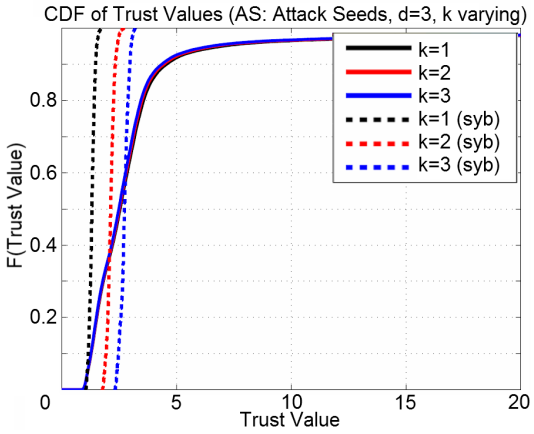
While SybilRank retains a good distinguishing ability if a Sybil can place only one attack edge randomly ($k = 1$), a Sybil can already obtain a higher ranking than 30% of the honest nodes with two such edges, leaving SybilRank with either a very high false positive rate (30% of honest nodes ranked as Sybils) or ineffective at detecting Sybils.

More worrisome, as shown in Figure 6.7b, if the attacker can place attack edges two hops away from a seed (i.e., $d = 2$), a single attack edge is sufficient for Sybils to outperform the majority of honest nodes.



(a) Sybils Attached Directly to Seed

(b) Sybils Attached to Seeds Neighbors



(c) Sybils Attached Further Away

Figure 6.8: Performance of SybilRank when attacking seeds. k is the number of attack edges per Sybil. If directly attached to a seed, a Sybil needs one attack edge to succeed (a). For each hop further away from the seed, Sybils need one additional edge to become indistinguishable to SybilRank (b,c).

Further, an observation drawn from additional experiments in Figure 6.8 is that, as a rule of thumb, if placing attack edges one more hop away from the seed, a Sybil will only need to add one more attack edge in order to achieve the same effect. In case of $d = 3$ (Figure 6.8c), SybilRank performs similarly to the scenario in which attack edges were placed randomly. This is a reasonable result considering that the average path lengths in OSN graphs is usually around 5 [83], and thus a distance of 3 towards a certain seed is as effective as a random placement of the edge. At the same time, the CDFs of benign nodes stay virtually the same for different number of attack edges.

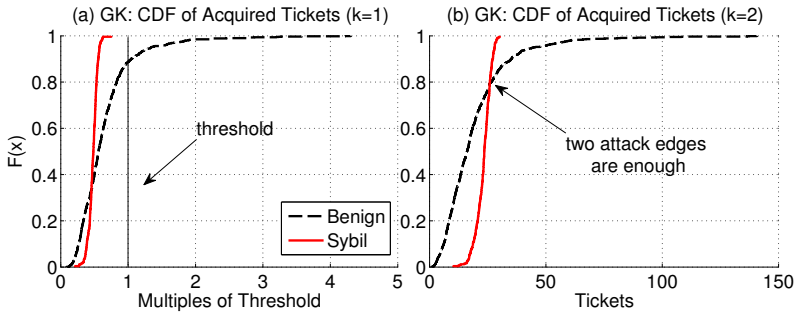


Figure 6.9: Performance of GateKeeper (GK). k is the number of attack edges per Sybil. Threshold = 35 tickets. When considering the threshold approach, most nodes, both Sybil and honest, do not get admitted, because GateKeeper is not able to work with a modular graph (a). A modification of the approach is only successful to limited extent (b).

6.5.2.6 GateKeeper

The distinguishing ability of GateKeeper depends on how many tickets Sybil nodes can obtain relative to honest users. Figure 6.9a shows two CDF curves of acquired tickets for Sybil nodes and benign nodes, respectively, with one randomly placed attack edge per Sybil node ($k = 1$). Clearly, one randomly placed attack edge per Sybil is sufficient to make the two CDFs cross. About 35-40% of the honest nodes obtain fewer tickets than Sybil nodes. If all Sybil nodes are excluded from being admitted, about 90% of the honest nodes will suffer the same fate. This is caused by the modular structure of the OSN (i.e., multiple distinct benign communities), which GateKeeper does not consider. With only few edges connecting different communities, most ticket sources selected by the admission controller via random walk will be in the same community as the controller, and nodes from other communities will only acquire at most a trickle of tickets. Here, the same concept that should protect GateKeeper from Sybils backfires. Further experiments with more, randomly placed attack edges show that Sybils will not gain much further advantage. As long as attack edges are randomly attached to a different community than that of the ticket source, Sybils will not receive much more tickets by increasing attack edges.

To see whether modifying GateKeeper may help, GateKeeper can be altered so that it can reach more benign nodes in modular networks. An admission controller is placed in *each* Louvain-detected community. The results are shown in Figure 6.9b, which shows that virtually all honest nodes are admitted, since they only have to be admitted by one controller, and there is one in each community. However, for the same reason virtually all Sybil nodes are admitted as well. If more attack edges are added, the Sybils outperform honest users. In fact, if a Sybil is able to obtain two random attack edges, it can collect

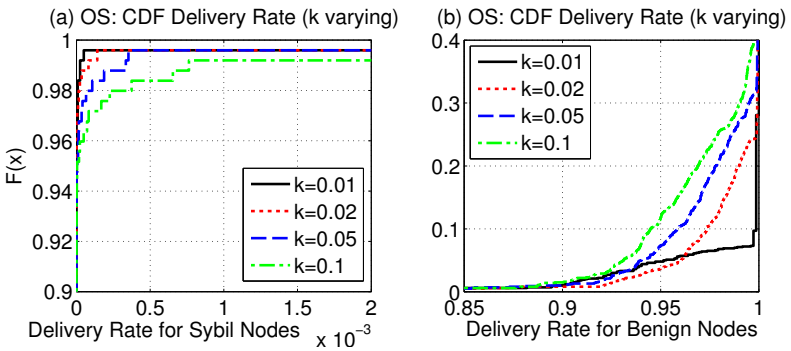


Figure 6.10: Performance of Ostra (OS). k is the ratio of attack edges in the system. Ostra can mitigate spam in the system (a), but also blocks honest users content from being sent (b).

more tickets than 80% of the honest nodes. This is because benign nodes have most links within one community, whereas Sybils have a good chance to place attack edges to multiple communities, and therefore in reach of multiple ticket sources.

6.5.3 Sybil Tolerance Approaches

The goal in evaluating Sybil tolerance approaches is to find out to what extent these approaches are able to limit the impact of the Sybil nodes in the modern scenario. In contrast to Sybil detection approaches, it is important to consider the number of attack edges relative to the number of honest edges in a Sybil tolerance system, i.e., the ratio of attack edges to honest edges, also denoted as k.

Therefore, the focus in ST experiments is on to which extent the impact of Sybils may grow with a higher ratio of attack edges or intelligent attack strategies.

6.5.3.1 Ostra

Figure 6.10 provides an overview of Ostra's performance for a varying number of attack edges. On one hand, Ostra does a good job in mitigating spam from Sybils. While the amount of spam messages that can go through does grow proportionally with the number of attack edges in the system, as shown in Figure 6.10a, Ostra is able to block a large amount of spam messages and keeps the delivery ratio for Sybils quite low. However, the true impact of an increasing number of attack edges lies in Figure 6.10b, where the amount of benign messages that are blocked due to the credit depletion on the path between a source and a

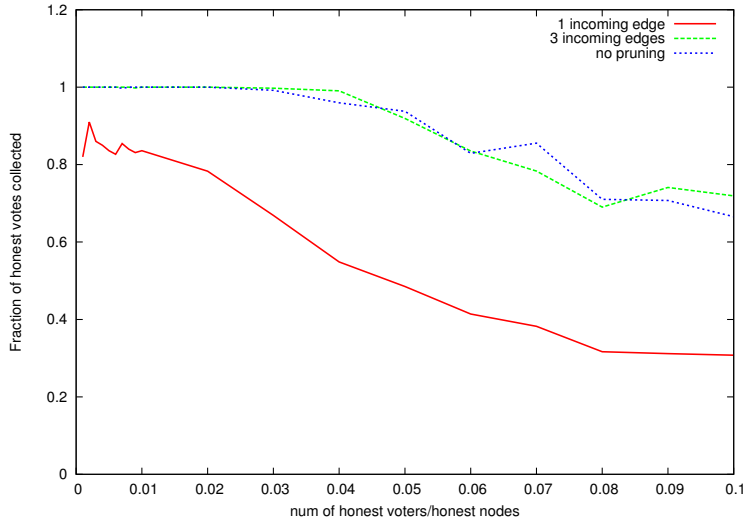


Figure 6.11: Performance of SumUp (SU). k is the number of attack edges per Sybil. Pruning to one incoming edge has a negative impact on the voting capabilities of honest nodes.

destination is evaluated. Recall that when a Sybil node sends spam to a destination, all links on the path—including edges between honest users—can be penalized by Ostra's feedback mechanism (see Section 6.4). For example, with 1% ($k = 0.01$) edges in the entire system being attack edges, about 5% of the benign nodes will have 5% of their messages blocked. Note that since a message that traverses one attack edge may traverse multiple benign edges, every newly added attack edge can multiply the negative impact described here.

6.5.3.2 SumUp

Among all evaluated approaches, SumUp is probably the most complex. It employs two mechanisms which modify an OSN graph—namely *pruning* and *feedback-based link elimination*. Both mechanisms are investigated in the following.

Pruning mechanism: Pruning the OSN graph affects both honest and Sybil users in their voting options. For *honest* users, as a successful pruning should have limited or no impact on them. This is the case if a node has three or more incoming edges after pruning as shown in Figure 6.11. Pruning to a single incoming edge, however, has an adverse impact since it depletes vote capacities on edges that could otherwise be routed around.

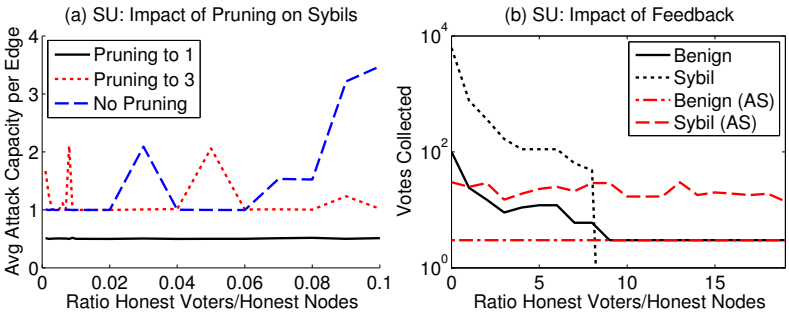


Figure 6.12: Performance of SumUp (SU). k is the number of attack edges per Sybil. Pruning has little impact on Sybils (a), feedback reduces the number of collected honest votes, and attackers can cycle through Sybils to outvote benign users (b).

Figure 6.12a illustrates how pruning may affect *Sybils'* voting capacity. Specifically, it shows how the capacity of attack edges varies when the pruning has a different level of aggressiveness and when more honest voters vote. The results in Figure 6.12a demonstrate that pruning has minimal impact on Sybil users, if any. Only pruning to a single incoming edge drops the average capacity per attack edge significantly. Unfortunately, doing so comes at the price of a reduced ability of collecting honest votes (Figure 6.11). Furthermore, when there is no pruning, Sybil voters perform better as more benign users vote. The reason is that the more honest votes reach the vote collector, the more tickets are to be distributed, thus the higher edge capacities on average and the more Sybil votes.

Feedback mechanism: SumUp is also designed for independent voting events, all of which take place at different points of time. In a single poll Sybils are able to influence the outcome of a vote, but SumUp utilizes a feedback mechanism in order to reduce, if not remove, the attack capacity of Sybil votes.

The best way to evaluate the efficacy of the feedback mechanism is to see how well it works when attackers establish attack edges close to the vote collector (allowing them to flood a large number of bogus votes). Thus in an experiment with multiple rounds of voting, the objective was to see if the feedback from every round can help the subsequent rounds of voting. Here, social graphs, where attack edges are directly connected to the vote collector and every node is pruned to have at most three incoming edges are used. Out of 1,000 honest users at every round a random 10% will vote, thus 100 votes per round. At the same time 10,000 bogus votes try to outvote the honest votes in every round. Figure 6.12b shows the results for this experiment. First, coinciding with SumUp's original findings, the feedback mechanism can rapidly reduce the number of Sybil votes after only a few rounds. While Sybils can initially cast 10^4 votes, no bogus votes can go through after round seven.

However, the feedback mechanism also reduce the number of honest votes. Although the bogus votes cannot outvote the honest votes after several rounds, in the end out of 100 honest votes cast every round, only seven can be taken. The cause of this problem is that as the feedback mechanism penalizes more and more links, some honest votes will not be able to reach the vote collector. Moreover, an intelligent attacker can employ some attack strategy (denoted as AS in Figure 6.12b) to counter the feedback mechanism. For example, instead of having all Sybils vote in every voting round, in every round it can cycle through the Sybil nodes and let different Sybils to vote (in this experiment every Sybil was given a probability of 0.10 to vote). By doing so, although Sybils cast less votes this way, they can outvote honest votes continuously.

Pruning and feedback mechanisms working together for Sybils: Recall that after SumUp removes penalized links, it can reintroduce pruned links to replace the penalized links in order to maintain the required number of incoming edges for every node (Section 6.4.2.2). Unfortunately, the attacker can take advantage of this feature, and Sybil nodes with a large number of attack edges can cycle through their penalized links and pruned links. If a Sybil has an attack edge that is far from the vote collector and a previously pruned edge that is close to the collector, the Sybil could even cast bogus votes to have the attack edge replaced with the pruned edge, thus moving itself closer to the vote collector.

6.6 Lessons Learned and the Impact on DOSNs

This section reflects on all the state-of-the-art, OSN-based Sybil defense approaches that were studied analytically and quantitatively, and lists the lessons learned based on their performance in both the classical scenario and the modern scenario. Table 6.2 provides a high-level summary of the analysis, measurement and comparison of these approaches.

As pointed out earlier (Section 6.4), all Sybil defense approaches assume the classical scenario. They all assume that Sybils cannot obtain many attack edges, so that a small cut exists in the social graph, whereas some further assume that the Sybils remain in a single Sybil region. Also, only few schemes acknowledge that there might be multiple benign communities, thus forming a modular OSN graph.

Sybil Detection. Based on these assumptions, Sybil Detection approaches are quite similar in their implementations. Most approaches leverage random walks or modifications and combinations of random walks. As a consequence, they suffer from a common weakness when facing the modern scenario and their ability to distinguish Sybils from benign nodes is no longer obvious. The experimental evaluation from a quantitative perspective further confirms the qualitative analysis on the drawbacks of current Sybil detection approaches.

Table 6.2: Sybil defense approaches summarized.

Solution	Year	Class	Assumptions	Technique(s)	Reason(s) for Ineffectiveness	Conditions to break
SybilGuard	2008	SD	SC / SR / SBC	RR	Sybils obtain enough RR intersections	$\geq$ 1 AE per Sybil
SybilLimit	2010	SD	SC / SR / SBC	Multiple RR	Sybils obtain enough tail intersections	$\geq$ 5 AE per Sybil
SybilShield	2013	SD	SC / SR / MBC	RR / RW	Sybils obtain enough RR intersections	$\geq$ 1 AE per Sybil
SybilInfer	2009	SD	SC / SR / SBC	modified RW	Enough traces end on Sybils	$\geq$ 2 AE per Sybil
SybilDefender	2012	SD	SC / SR / SBC	RW / RR	Sybils vs. Benign: RW frequency indistinguishable; SR detection fails (no SC)	$\geq$ 2 AE per Sybil
SybilRank	2012	SD	SC / MBC	RW	Sybils earn high enough ranks	$\geq$ 2 AE per Sybil
GateKeeper	2011	SD	SC / SBC	BFS Tickets	Sybils collect enough tickets	$\geq$ 2 AE per Sybil
Ostra	2008	ST	SC / SR / SBC	Link Credit	Sybils can block benign content	$\geq$ 1% AE in Network
SumUp	2009	ST	SC / SR / SBC	Link Credit	Sybils can outvote honest nodes	Cycle AE in Voting

Legend: SD = Sybil Detection, ST = Sybil Tolerance, RR = Random Route, RW = Random Walk, SC = Small Cut, SR = Sybil Region, SBC = Single Benign Community, MBC = Multiple Benign Communities, AE = Attack Edges

In particular, they are indeed very vulnerable to an increasing number of attack edges. For some, a single, randomly attached attack edge is sufficient for a Sybil to disguise itself as a benign node, making these schemes incapable in real-world networks where Sybils can easily obtain hundreds of these edges (see Section 6.3).

Additionally, the more sophisticated a scheme is by introducing additional steps to Sybil detection, the more likely it will introduce more serious problems. SybilShield and SybilDefender are probably the best examples. In the former, the use of agents opens up more chances for Sybils to become accepted, whereas in the latter, the Sybil community detection can have a high false positive rate.

Finally, simple modifications to these approaches are not sufficient to improve their capabilities in detecting Sybils. Out of the approaches under investigation, a modified SybilLimit required the most—but still very little—effort of the attacker. Here, a Sybil needs to obtain about five attack edges in order to hide itself successfully. The increased effort is due to a lower benefit of each attack edge to the attacker than that in other Sybil detection approaches. While having more attack edges makes it easier for the random routes from a Sybil node to intersect with random routes from a verifier, at the same time it also helps—although to a less extent—the random routes from a benign node to intersect with those from a verifier. In other approaches, adding more attack edges yields higher benefits for the Sybils, allowing them to break the defense solution with less effort. For instance, in SybilShield, an increase in attack edges results in more intersections with the agents, from which the benign nodes do not benefit in most cases.

Sybil Tolerance. In contrast to Sybil detection schemes, Sybil tolerance schemes (Ostra and SumUp) are not as broken—they still limit Sybil activity to some extent. The reason is that, unlike Sybil detection approaches, Sybil tolerance systems do not need to decide whether or not a node is a Sybil, but can rather adaptively react to the behavior of ma-

licious nodes. However, both Ostra and SumUp have serious flaws as well: in Ostra a non-negligible fraction of benign nodes may be blocked from communicating, and SumUp would allow an intelligent attacker to outvote benign users, leaving both schemes with only limited success in tolerating Sybils.

6.6.1 Prospects of Future Sybil Defense Solutions

These results further provide insights to new Sybil defense solutions. The main commonality that connects all current approaches is that they solely exploit the (same) distribution of edges in the OSN graph. Follow-up suggestions to detect Sybils, such as using the clustering coefficient [51], fall in the same category and are therefore very sensitive to changes in the graph structure as well.

One suggestion is to force Sybils into the required structure by monitoring the link request acceptance rates of different nodes [171]. As a node has to be accepted by a certain number of other nodes to be classified as benign, Sybils might be forced into creating many links among themselves (which are guaranteed to be accepted). This would eventually lead to a larger density of edges among the Sybils themselves compared to the links with honest nodes—which could ultimately allow detection using existing approaches again. However, Sybils can already achieve acceptance rates of up to 90% for their link establishment requests. Also, while this scheme depends on Sybil nodes initiating contact, Sybils actually can use simple attacks to gather a lot of requests *toward* them (Section 6.3), making it unnecessary for Sybils to reach out to benign nodes for acceptance.

In fact, structural properties only account for a very small fraction of the information incorporated within a social relation [81]. Social networks contain lots of meta-data that quantifies the strength of ties between users. Therefore, in looking forward to future Sybil defense solutions, an approach that *enriches* the structure of a social graph with more information about the *relations* between its users in order to defend against Sybils is anticipated.

For instance, one could measure the *intensity* of communication between two particular nodes—a major contributing factor to the tie strength between users [81]—to detect Sybils. Here, if a node has a low intensity of communication with nodes already identified as benign, it then might be classified as a Sybil.

However, such an approach can suffer from a high false positive rate, as honest users who rarely interact with others might be mistaken for Sybils. A similar criterion contributing to the total trust is the *intimacy* of a relation. Here, the messages between users might be analyzed for certain keywords. Again, such an approach can introduce a high false positive rate, and as message content needs to be parsed, it can interfere with the users' privacy.

Another major factor which contributes to the tie strength is *duration*. The longer a relation exists, the stronger it is. In contrary, attack edges could experience a significantly shorter lifetime than regular edges, since they might be deleted once a benign user realizes he has become connected to a Sybil. If so, one could classify nodes whose links experience suspiciously short lifetimes as Sybils.

6.6.2 Towards Other Research Directions

One key observation of the previous sections is that OSN-based Sybil defenses are overly reliant on the social relations between participants. Doing so, however, is not unique to Sybil defenses. A large number of systems of various research directions do the same and, as a consequence, may need to be rethought as well.

Like Sybil defenses, many of these systems either try to strengthen the *security* of a target system, or try to improve the *performance* of the target system. For instance, Reliable E-Mail [76] is an email spam protection service that whitelists friends of friends. Hence, if a Sybil can obtain attack edges with a short distance to the target of a spam email, the Sybil will be whitelisted as well. At the same time, in *Delay Tolerant Networks*s (DTNs), some researchers suggest that friend nodes are better carriers for messages [165]. Thus, if an attacker can obtain attack edges close to the source of a message and deny any forwarding actions, it can act as a dead end for these messages.

These systems do not face serious problems if attack edges are rare. However, serious drawbacks can be assumed if the modern scenario is considered. It is suggested above that Sybil defenses, which solely rely on structural properties of the graph are hardly viable. Practically, the same statement is valid for *many* more systems.

6.6.3 The Impact on DOSNs

The original question asked at the beginning of this section was whether or not there exist Sybil defenses that can be of help for existing or new DOSNs. The previously discussed results show that this is clearly not the case, and that in fact these defense solutions open up many possibilities for Sybils to easily sneak into the OSN by disguising themselves as honest users. This is particularly the case for *Sybil Detection* (SD) approaches, which would be more desiring for DOSNs, as they are aimed at providing a more universal defense against Sybils than Sybil tolerance approaches. These systems should *not* be applied to a novel DOSN, as they could lead honest users to believe that every participant of the network is honest. At the same time, they might even classify weakly connected benign users as Sybils.

DOSNs could still use *Sybil Tolerance* (ST) approaches to thwart certain spam or manipulation attacks. However, for both ST approaches under investigation, serious system-specific weaknesses prevent their application to DOSNs. If Ostra would be applied to prevent the distribution of spam, a significant number of honest users could be blocked from sending a certain fraction of their non-spam messages, while SumUp has shown to be inefficient in preventing the outvoting of honest users by Sybils. These findings have two major implications.

First, existing DOSNs are in fact very vulnerable to the Sybil attack. In Safebook for instance, each user stores data at her supposedly trusted friends, who also act as request forwarders towards the user. If such a friend happens to be a Sybil, both the user's privacy (e.g., the Sybil could analyze the data or track access to it) and the system performance (e.g., the Sybil could deny requests to the data) can be degraded.

The same applies for Cachet, Proofbook and MyZone, which all put a lot of trust in the social relations of a user. For instance, Cachet could possibly end up in caching data at Sybils. In MyZone passphrases to access a users data are shared among friends, and a user refuses connections from non-friends [45]. Since Sybils can in fact easily establish hundreds of social relations to honest users, they can gain access to user data and flood others after infiltrating the system, while MyZone will not react to any of these attacks. Such deficits add to the multitude of drawbacks of related works, as described in Chapter 4.

Second, and more importantly, a new DOSN needs to be resilient against an attacker orchestrating a large number of Sybils. That is, it has to *tolerate* that users might not be careful when establishing social relations. In other words, even though the adversary might be able to establish a large number of social relations to honest users, she still should not be able to have a significant impact on the quality of service of these users. For instance, recall that one property of a novel DOSN should be to exploit the opportunities of social relations in OSNs. Hence, for each collaborative element of a new approach—as, for instance, any sort of recommendation process—there must not be an opportunity for Sybils to easily exploit it.

Also, recall that systems with permanently available resources were ruled out earlier in this thesis. In a network that is based on the cooperation of users to store data for each other an attacker may have the opportunity to flood the system with data. If such flooding is possible, it must be *detected* by the DOSN as well. Addressing these challenges is mandatory for a new approach.

6.7 Chapter Summary

In this chapter a wide range of state-of-the-art OSN-based Sybil defenses have been profoundly studied. The goal was to figure out to what extent these defense solutions can be of help to protect DOSNs against suffering from a Sybil attack. After elaborating on a more recently emerging attacker behavior, in which the attacker is able to disguise her fake identities better than previously assumed, this chapter systematically analyzed and evaluated two classes of Sybil defenses (*Sybil Detection* (SD) and *Sybil Tolerance* (ST)) with regards to their performance when faced with that behavior.

The main finding was that current Sybil defenses have serious problems in the identification of Sybils, often mistake honest nodes as Sybils, or accidentally impose restrictions on benign users. Because modifying existing schemes does not result in a significantly better performance, they should not be used as reliable protection means by DOSNs.

As a consequence, an additional central challenge for a new DOSN will be to

(i) efficiently mitigate the impact Sybils can have on each honest user's experience; and

(ii) ensure the unimpaired functionality of the DOSN as a whole in the presence of an increasing number of malicious accounts.

Chapter 7

SOUP - An Online Social Network By The People, For The People

The previous chapters provided a thorough analysis of the reasons for building a competitive DOSN (Chapter 3), the shortcomings of previous efforts to do so (Chapter 4), and the challenges such a system has to overcome (Chapters 5 and 6).

In this chapter the design of a novel approach towards a robust and secure DOSN, the SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP), is presented. Ultimately, SOUP is designed to solve all the previously outlined challenges by replacing the central provider with an infrastructure substrate that is built and managed by the users themselves.

The focus of the upcoming sections is the construction SOUP's *communication* infrastructure, which inherently addresses the first set of DOSN challenges. The remaining challenges are then addressed in the process of building the *storage* infrastructure (Chapter 8).

Contents

7.1 SOUP in a Nutshell

In the SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP) all online nodes form a structured overlay that enables communication among them. The nodes maintain this overlay as a global user directory in the absence of a central provider. In particular, a user interested in communicating with another participant can search for that participant's addressable identifiers, and then initiate a direct connection over an appropriate link.

Within the overlay, every SOUP node comprises the *SOUP middleware* and the *SOUP applications*. Multiple SOUP applications can run concurrently on top of the SOUP middleware, each of which can use the same social data for a different social networking context. The applications communicate through a generic API with the middleware, which resides between the network stack and the applications. The middleware provides the means to

(i) organize all online SOUP nodes into the overlay (Section 7.2);

(ii) establish communication channels with other SOUP nodes (Section 7.3);

(iii) provide the generic API to SOUP applications (Section 7.4); and

(iv) handle mobile nodes, which may run on limited resources and with limited connectivity (Section 7.5).

The lack of a central provider also entails the lack of a global data repository. To mitigate the absence of this repository, SOUP also has to build a storage substrate. To enable this substrate, every participating user (represented by a SOUP node) maintains her own data, and selects a small set of other nodes as **mirrors** to store replicas of her data, in order to keep her data available even when the user herself is offline. The replicas are synchronized by an recursive update-based mechanism, whereas the data itself is securely encrypted. The encryption routines allow each user to define arbitrary, fine-grained access policies to each part of her data. In summary, the middleware thus further provides the means to

(v) ensure user data privacy (Section 7.6);

(vi) replicate the encrypted data to the mirrors (Chapter 8); and

(vii) maintain and synchronize the replicas (Section 7.7).

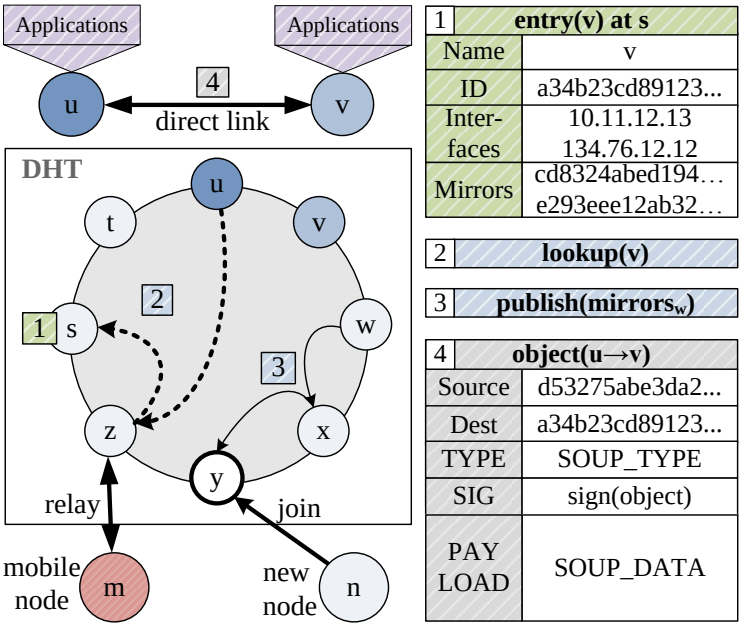


Figure 7.1: An overview of the SOUP overlay based on a DHT. A user u can store her information entry, including a list of mirrors at which others can find her data while u is absent, in the DHT for others to lookup. After a successful lookup, two users can directly communicate by exchanging signed, encrypted SOUP objects, which can carry arbitrary content from applications operating on top of SOUP. New nodes can join SOUP via a bootstrapping node and mobile nodes' DHT requests are relayed by a fixed gateway node.

7.2 The SOUP Overlay

The structured overlay is formed as shown in Figure 7.1. It acts as a globally searchable information directory and is based on the Pastry *Distributed Hash Table* (DHT) (see Chapter 2.2). The choice of a DHT as the structured overlay is justified by the following reasons:

(i) A DHT does not require any permanently available resources.

(ii) It enables efficient publish and lookup operations in a decentralized fashion, making a centralized user directory unnecessary.

(iii) The availability of the directory information is secured by an internal replication mechanism [108].

(iv) It can easily scale to millions of users [107, 108]. In fact, the more users there are in the system, the more stable the DHT operates, because a partition of the ring becomes more unlikely.

(v) DHTs have shown to be able to handle the churn rates typically present in OSNs.

(vi) DHTs offer important security features, such as, for instance, being resilient against Sybil nodes [175].

In the DHT, every SOUP user can publish her directory entry at the node that is responsible for her *Identifier* (ID) in the DHT key-space (e.g., *v*'s entry is published at *s*—Step 1 in Figure 7.1), and any other node can locate the node to retrieve the entry (e.g., *u* can look up *v*'s ID—Step 2). An entry typically contains a user's name, her SOUP ID, the interfaces (i.e., IP addresses) via which she can currently be contacted, and the SOUP IDs of all the mirrors of her data. Here, the **SOUP ID** is a 64-bit SHA-256 hash over the user's 2048-bit public key and uniquely identifies the user.

It is important to note that in contrast to some related work [44], a user only publishes pointers to mirror nodes (i.e., SOUP IDs) in the DHT (e.g., *w* publishes her mirrors at *y*—Step 3), whereas the data themselves are stored among nodes themselves. Directly storing data in the DHT would have undesirable consequences:

(i) Every user would have no control over which other nodes will be her mirrors to host her data, whereas the mirrors would have no option to reject unwanted data.

(ii) Moreover, it would increase the overhead of the system; whenever a node departs—which can be often since SOUP nodes may have a high churning rate—it has to transfer all its DHT data to another node.

Both problems are mitigated by decoupling the storage substrate (see Chapter 8) from the overlay directory.

SOUP incorporates a list of publicly known *bootstrapping nodes* to support new nodes in joining the system. A bootstrapping node is simply a regular node enhanced with a function to bootstrap others: a new node can contact a bootstrapping node as its entry point to the DHT, thus adding itself to the DHT. For example, in Figure 7.1, node *n* joins the DHT via a bootstrapping node *y*. It then can prepare its entry (including looking up its own SOUP ID in the DHT to make sure the SOUP ID is not used by another node by any chance), and publish it to the DHT, enabling other nodes to look up the entry of the newly joined node.

7.3 Communication in SOUP

To request data of interest, a user establishes a connection with another user in two steps: In the first step, the user searches the entry of her communication partner in the DHT (e.g., u looks up the entry of v in Figure 7.1; m would do so via its gateway z). In the second step, she extracts the addressable interfaces of the partner from the entry and creates a direct communication channel. This channel can be based on any networking protocol, ranging from standard *Transport Control Protocol* (TCP)/IP to Bluetooth, if available. Once a communication channel is established, the communication partners (u and v in Figure 7.1) exchange signed *SOUP objects*, which can contain arbitrary information (Step 4 in Figure 7.1).

7.4 Applications in SOUP

The communication channel is usually created upon a request of a SOUP application to send some data to another node running the same application. Multiple such applications may be running concurrently on top of the SOUP middleware. They can encapsulate their payload (such as user data or friend requests) into SOUP objects, which are then handled by the middleware. The encapsulation into the SOUP object is done over a simple API, which offers the functionality listed in Table 7.1. For instance, if an application issues a friend request, the middleware encapsulates the request into a SOUP object and then forwards the object to the request's target (`ADD_FRIEND` in Table 7.1).

Additionally, the API in particular allows applications to exchange content transparently to the middleware (`TRANSMIT_APP_DATA` in Table 7.1), and thus enables the development of any kind of OSN application on top of it. For instance, a Flickr clone would predominantly exchange images among users. In that case, the middleware can encapsulate each image into the `PAYLOAD` field of a SOUP object (see Figure 7.1), which is afterwards treated like any other object carrying application data. Only when it reaches the destination node, the payload is decapsulated and forwarded to the Flickr clone application running on that node—or stored as an update for that application if it is not running.

Such transparency establishes SOUP as a generic DOSN, in which a single set of data can be modified by a multitude of OSN operations, whereas its owner remains in control of access to the data.

Table 7.1: The API offered by SOUP to applications.

Command	Function
REGISTER_APP	Registers an application with the middleware
UNREGISTER_APP	Exits the application appropriately
FOCUS_APP	Claims user focus on this application
ADD_FRIEND	Creates a friend request
CONFIRM_FRIEND	Confirms a friend request and adds the user to friendlist
EDIT_FRIEND	Edits ABE attributes for an existing relation
REMOVE_FRIEND	Deletes a user from the friendlist
ADD_GROUP	Creates an ABE attribute
REMOVE_GROUP	Revokes an ABE attribute
MODIFY_DATA	Modifies or adds a particular data item of the local user
REQUEST_DATA	Requests a particular data item from a particular user
CHECK_UPDATES	Checks for updates on the shared social data
TRANSMIT_APP_DATA	Transmits application level data to another SOUP node

7.5 Mobile Nodes in SOUP

SOUP is designed to be friendly to mobile nodes. As these devices often experience exceptionally high churn (e.g., because of connectivity changes) and long response times (e.g., due to limited bandwidth), they can decrease the performance and stability of the DHT. SOUP addresses this challenge by exempting mobile nodes from the DHT. Instead of being on the DHT, a mobile node will relay its DHT publish and lookup operations through a *gateway node* that is on the DHT (e.g., node m will relay through node z in Figure 7.1). Doing so has multiple advantages:

(i) It frees mobile nodes from directly executing DHT operations. In particular, they do not participate in the shifting of directory entries upon the DHT join or departure of a node. The exemption thus saves resources (e.g., bandwidth and energy) on the mobile devices.

(ii) Further, the stability and performance of the DHT is improved. By removing the often instable mobile nodes from the overlay, the likelihood of a faulty DHT ring is reduced.

Also, if mobile nodes were part of the DHT, they could significantly impact the lookup latency due to a possibly very limited bandwidth (e.g., in rural areas).

A mobile node initially uses a bootstrapping node as its gateway (the same node it contacted to join SOUP). However, every time it encounters another node, it checks that node's ability to relay DHT requests (every regular node can set a limit to mobile connections) and switches to that node as a gateway if possible, thus reducing the load on bootstrapping nodes (e.g., node *m* has switched from node *y* to *z* in Figure 7.1). Note that since the data itself is not stored in the DHT, the relayed requests do not consume a lot of bandwidth at the gateway node.

7.6 Data Privacy in SOUP

To grant user data privacy, SOUP has to fulfill one paramount task: it has to endow users with means to specify fine-grained access control to their data with regards to other users according to their personal preferences [36]. The requirement for fine-grained access control stems from the observation that a user Alice may be connected to a large number of users in the OSN, but each of the links between Alice and another user Bob can be interpreted differently. In fact, Alice might actually be linked with family, friends, co-workers, sports mates, and so on, and SOUP should provide Alice with the tools to classify access to her data, so that it represents the actual off-line relationship. For instance, Alice should be able to define access policies to exclude users or groups of users from accessing more (or less) personal informations.

Solving this task also provides users with the capability to restrict a mirror from accessing their personal data. If Alice specifies access to (parts of) her data on a per-user basis, a storage provider seeking access to the data has to be included by Alice as an eligible user.

However, to provide such functionality in absence of a centrally organized, trusted provider requires some form of cryptographic support for efficient decentralized group keying [36].

7.6.1 Traditional Cryptography in DOSNs

Traditional symmetric and asymmetric cryptography can help to enable efficient group encryption [176, 177]. The OSN scenario however is different from the traditional group keying scenarios in that a user sending to a group may not control the membership to the group. For instance, Bob may post a picture to Alice with the intention to encrypt it for

Alice's friends to allow them to view the picture. However, Bob does not (and should not) necessarily know the list of Alice's friends and can thus not easily encrypt the picture for every friend of Alice.

The group keying process is even more complicated if Bob wants to further restrict the audience of a message (e.g., to target Alice's friends who at the same time work with Bob). Finally, when applying traditional group keying, the number of groups that exist in the OSN can be extremely large, as users can encrypt for arbitrary (in the worst case any) combination of friends, friends of friends or even strangers.

In detail, in order to create a group from a list of users with which Alice is connected (e.g., friends), Alice can encrypt a symmetric or asymmetric group key with the public key of each future member of the newly created group. A symmetric key leaves only the members of the group able to encrypt content destined for the group, whereas an asymmetric key also allows non-members to encrypt messages directed at the group (at a higher cost for the cryptographic routines).

Alice then forwards the group key to all members of the new group, whereafter all members can use the key to send encrypted messages to the group. This way, Alice (as the creator of a group) can include any set of friends into the group, and messages can also be encrypted for unions of groups. To do so, Alice's friend Bob, who is member in two groups G_1 and G_2 can encrypt messages for Alice's friends who are in either one of two groups (G_1 `OR` G_2) by encrypting the message with each group key separately.

On the downside, it is not possible for Bob to encrypt the message for an intersection (members who are in G_1 `AND` G_2). For that, Bob would encrypt with both group keys consecutively. However, two friends of Alice, say Malory and Peter, can collude to gain access to the message, even if both of them are in one of the groups only (e.g., Malory only in G_1 and Peter only in G_2) [36]. In the example, Peter can first decrypt with the group key of G_2, forward the result to Malory, who then can decrypt the plain text of the message by applying the group key of G_1, or vice versa, depending on the order of encryption.

Also, a friend Mary of Alice who is not a member of a group G_1 cannot easily encrypt data for members of G_1, as she does not know the group key of G_1. Alice could publicly list the groups and the corresponding public keys, and Mary could find the correct key for encryption to G_1. However, this requires additional infrastructure and, when using symmetric cryptography, only group members can know the shared symmetric key.

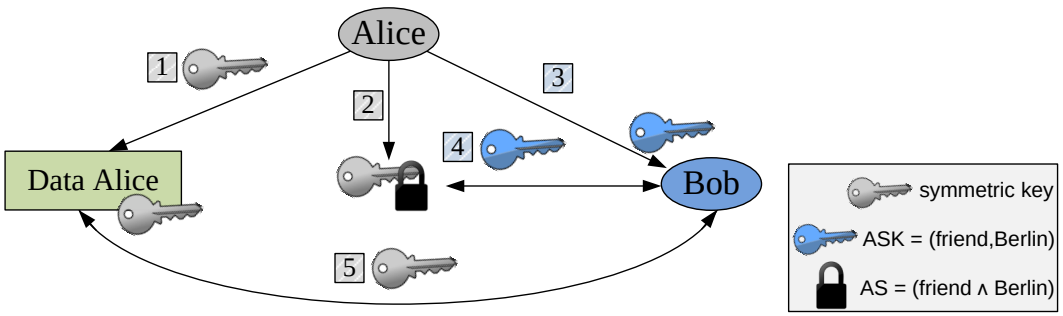


Figure 7.2: An example of encryption in SOUP for two users *Alice* and *Bob*. Here, *Alice* encrypts a data item with a symmetric key, and protects the symmetric key with an AS. She also creates an ASK for a user *Bob* she wants to grant access to the data. With that key, *Bob* satisfies the AS and can consequently access the symmetric key, with which *Alice*'s data item can be decrypted.

7.6.2 Encryption in SOUP

In order to avoid the problems listed above, every SOUP user encrypts all her data by using a combination of traditional cryptography with *Attribute Based Encryption* (ABE) [132]. Here, to ensure the confidentiality of all privacy-relevant user information, SOUP follows the ideas of Baden et al. in [36]. The symmetric key required to access the encrypted content is protected by an ABE *Access Structure* (AS), which itself is defined by a combination of attributes, so that only requesters holding the correct attribute key can decrypt it (see Chapter 2.3.3). The use of the symmetric key for the actual data encryption reduces the amount of expensive (asymmetric) ABE operations. Critical operations (i.e., granting access to the key required to access some data) are handled by ABE, while common operations (e.g., the actual access to the data itself) are based on symmetric keys [36].

In SOUP, Alice will act as a quasi-CA for her own data as depicted in Figure 7.2. She first encrypts (parts of) her data with a symmetric key (step 1 in Figure 7.2). Afterwards, she defines the AS to restrict access to the symmetric key that is required to decrypt a particular data item (step 2). The AS itself is defined as a logical expression over a set of attributes, e.g., ('friend' `AND` 'Berlin').

To grant access to her data, Alice will then create an *Attribute Secret Key* (ASK) for *each* of the users she wants to access some parts of her private data (in step 3 in Figure 7.2, Alice creates the ASK for a user Bob with the attributes *'friend'* and *'Berlin'*). In the ASK of Bob a number of attributes, which Bob fulfills from Alice's perspective, are embedded [132, 133]. In other words, the ASK specifies the groups Bob belongs to for Alice. Recall that the

ASKs are constructed such that Bob, if he is a member of both groups $friend$ and $Berlin$ (i.e., Bob holds *both* attributes), can access the key (step 4) required to decrypt the data item (step 5) (see Chapter 2.3.3 for details).

At the same time, two colluding users Malory and Peter cannot access the symmetric key if both of them are a member of one group only (i.e., each of them holds *one* attribute only). Further, any non-member of a group can encrypt to the group's access structure—or to any other access structure, thereby creating any new group—if she knows Alice's *ABE Public Key* (APK) and the names and definitions of the attributes Alice defined.

7.6.3 Attribute Management Routines

By using appropriate attributes, a user can then grant fine-grained access to her confidential data, as the data cannot be accessed by other entities except those holding the corresponding attribute keys. For instance, the user can limit access to one item to users holding two specific attributes, while three different attributes are required for another item. The attributes themselves can be arbitrary (e.g., such as *colleague* or *lives in my city*). In particular, the mirrors themselves cannot access the data stored at their premises without holding the correct attributes.

Thus, a user needs to have available particular routines to assign attributes to other SOUP users. This thesis broadly follows the implementation of management routines for attributes in OSNs as provided by Baden et al. , which only introduce a limited overhead—even for mobile devices [36]. In detail, granting access to data for individuals or sets of users is achieved as follows:

Adding a user to a group. If Alice wants to add a friend Bob to her set of friends, she will create $K = Alice.ASK_{friend'}$, the ASK that grants access to any content that is published for the group 'friend'. Afterwards, Alice will compute $C = TEncrypt(Bob.TCPK, K)$, i.e., asymmetrically encrypts K with Bob's *Traditional Cryptography Public Key* (TCPK) and makes C available to Bob who can then decrypt C using his traditional cryptography private key. Hereafter, by using K, Bob can decrypt the symmetric key to all content, which was encrypted for Alice's friends. This way, Alice can also dynamically add users to existing groups. She can then encrypt her data for arbitrary conjunctions of these groups, which allows her to define access to her data on a fine-grained basis.

Assigning Rights to a Group. Alice may additionally want to encrypt data for a set of users from different groups. For instance, suppose Alice wants to encrypt a message for both Bob (who is in the group *friend*) and Charlie (who is in the group *colleague*). Here, a combination of attributes (*friend* `OR` *colleague*) is unsuitable, as it would allow all users

in both groups to access the message. Hence, Alice will define a new group. To setup the group, she creates a new traditional public key cryptography key pair and encrypts this key pair with an AS defining the group. Then, the key pair is the group identity and can afterwards be treated as a single user as described above.

Transitive groups. Besides Alice herself, others may want to encrypt data for groups defined by Alice. For instance, Bob might want to address the set of Alice's friends. To do so, he needs to define a group based on the group definition of Alice. Thus, Bob creates $B = Bob.ASK_{'alice-friend'}$ and encrypts it with the *Access Structure* (AS) $'friend'$ using Alice's APK ($Alice.APK$), such that users holding the attribute $'friend'$ issued by Alice can access the key. Afterwards, Bob can encrypt content using B and make it available at Alice's mirrors.

Modifying groups. *Removing* a user from any group, currently, requires re-keying of the whole group. However, there exist ABE-based schemes to reduce re-keying, but unfortunately, these schemes are not implemented and thus not ready-to-use for SOUP yet [178].

The opposite case, i.e., *adding* a user to the group does not require such re-keying. However, the newly added user might not be supposed to read older contents of the group, which were discussed prior to her membership. In this case, a condition, e.g., $keyYear \leq 2012$, can be included as an attribute into the AS to prevent newly joined members to read older data.

7.7 Data Synchronization in SOUP

Besides managing her own data, a user may receive updates from other users. These updates can be varying in their type. For instance, an update could contain a message being sent to the user, or a friend request that needs approval. Depending on the content an update might require the user to alter her data. For instance, when accepting a friend request, the user would have to update her friendship list.

If the user is online herself, she can directly receive these updates, order them based on the timestamps included in the received SOUP objects, and alter her data accordingly. For instance, when receiving multiple messages shortly after another, the timestamps can help to order them correctly.

Afterwards, she will (re-)encrypt the updated or new data and can then distribute the update to her mirrors. Note that the mirrors themselves are not (necessarily) eligible to modify the (encrypted) user data themselves, but are merely used as storage facilities. Because SOUP encrypts and transmits data on a per-item basis, a user does not have to transmit her whole profile to all mirrors after each change in her data. In case of an added friendship,

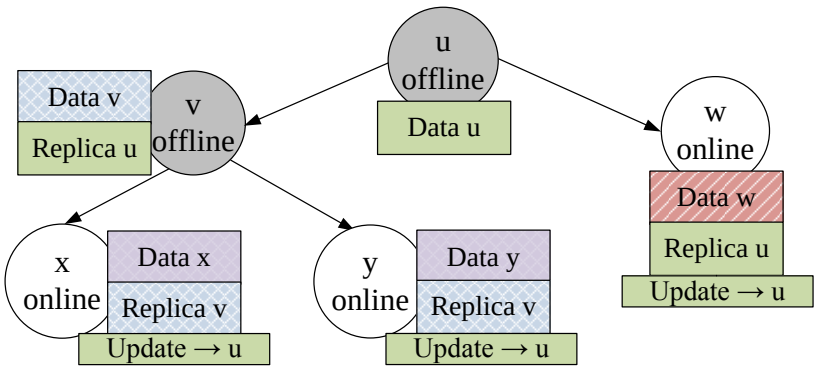


Figure 7.3: SOUP's Replica Management. An offline node *u* has its data available at its mirrors *v* and *w*. If an update (e.g., a friend request) arrives for *u*, it is stored at *u*'s mirrors. In this scenario, mirror *v* is offline itself, such that the update for *u* will be forwarded to *v*'s mirrors. *v* can then collect the update upon returning online. This way, all mirrors which are online always have the most recent updates available for *u* to collect at its return.

the user would instead just have to re-distribute her friendlist to her mirrors.

If the user is offline (e.g., *u* in Figure 7.3), she then needs assistance from her mirrors (e.g., *v* and *w* in Figure 7.3). The mirrors act as a surrogate by receiving and storing the updates to a user's data, which can then be collected and ordered by the user upon returning online later. Thus, as *u* is offline, updates for *u* have to be stored at *u*'s mirrors, *v* and *w*. *u* can then collect these updates, decrypt the update objects, and alter her data appropriately upon her return. SOUP is designed such that at least one mirror of each user is online at any time and can retrieve these updates (see Chapter 8).

In both cases, a mirror might be offline herself as well. In that case, updates destined to the user may have to be stored at the mirror's mirrors analogously. Upon returning online, the previously offline mirror will contact her mirrors to retrieve the updates destined for the user. Hereby, all mirrors always present the most recent user data if they are online, which also enables the data owner to synchronize different personal devices. In Figure 7.3, mirror *v* herself is also offline, so that updates for (not the whole replica of *u*) have to be further passed on to *v*'s mirrors *x* and *y*. Hence, *v* can retrieve any updates to *u*'s data upon returning online from its own mirrors.

Note that requests to modify any data must be encapsulated in an appropriately (i.e., with the owner's private key) signed SOUP object, and will otherwise be discarded. Else, each user with access to the symmetric key that protects the data could also modify it.

7.8 Summary of Addressed Challenges

The system architecture of SOUP inherently addresses a number of the challenges for new DOSN approaches.

(i) By relying only on other OSN nodes to form the storage substrate, SOUP remains free of charge for users. Moreover, distributing the task of data storage to a multitude of storage locations (mirrors) for each user reduces the risk of a large-scale data leak at a single instance.

(ii) By allowing applications to transparently exchange content, SOUP is generic, and since multiple applications can run concurrently on top of the middleware, SOUP also facilitates inter-operability between multiple OSN services.

(iii) By exempting mobile users from overlay maintenance, SOUP goes easy on the resources of their devices, while at the same time increasing the stability of the overlay.

(iv) By providing comprehensive ABE operations to each user, SOUP provides user-controlled data access policies, and no entity can obtain access to data by simply providing storage facilities.

However, due to removing the central data repository and relying on user machines as mirrors, the system has yet to solve multiple challenges, including two of its two key objectives.

(i) SOUP needs to be robust, i.e., it has to efficiently achieve high data availability of all users' data.

(ii) SOUP needs to be secure, i.e., it has to be protected against DDoS or Sybil attacks.

SOUP addresses these objectives and the remaining challenges in the way in which each node selects its set of mirrors.

Chapter 8

Mirror Selection in SOUP

In the previous chapter SOUP has been shown to address multiple of the challenges for DOSNs. This chapter describes how SOUP keeps its remaining promises.

To constitute a *robust* OSN, SOUP has to ensure that at any given time for every OSN node, either the node's data is available at the node itself (the node is online), or a copy of the data—called a *replica*—is available at another node—called a *mirror*. The core task for SOUP is thus mirror selection: every OSN node needs to select the most eligible nodes as mirrors before it places its data replicas there. The selection process should also consider altruistically provided resources, while it must reduce the generated overhead to a minimum.

At the same time, to also set up a *secure* OSN, the mirror selection must be resilient against a multitude of malicious attacks.

Contents

8.1 Mirror Selection in a Nutshell

Every node employs two modes to select its mirrors, a *bootstrapping* mode and a *regular* mode. When a node joins the OSN and does not know anything about the network, it runs in the bootstrapping mode, which allows it to gain a foothold in the OSN; it obtains recommendations from each node it encounters and ranks mirror candidates based on this information (Section 8.2). Once a node befriends others, it begins to learn from them about their experience in accessing its data at its mirrors, and transitions to the regular mode; it will now rely on friend experience to rank mirror candidates (Section 8.3).

The two modes differ in their way of ranking mirror candidates, but follow the same routine for selecting mirrors (Section 8.4). Here, a node will primarily consider that the higher a candidate is ranked, the more likely it will make the node's data available. Note that it is more so with the regular mode when direct user experience is used for ranking, as opposed to looking at strangers' recommendations in the bootstrapping mode. SOUP further allows every node to dynamically select as many mirrors as needed. As a result, no matter whether a node itself is online a lot or not, and no matter whether it has many friends or just a few, as long as it has enough quality mirrors via SOUP's algorithms, its data will be highly available through those mirrors.

The mirror selection is thus robust.

SOUP leverages social relationships in the mirror selection process primarily through experience exchanges, and every node functions with the help of its friends: node u's experience in accessing node w's data via w's mirror v helps w decide if v is a good mirror. But social relationships can be useful in other contexts as well. Since friends have more incentives and higher trust to store data for each other, a node assigns a higher weight to friend candidates when selecting mirrors, and protects profiles of friends when dropping data from its storage.

Dropping data may be necessary if a node is chosen as a mirror by many nodes, and its resources are exhausted. A dropping strategy is critical, especially when an adversary is flooding the OSN and many nodes receive numerous malicious requests to store data. For this task SOUP employs a *protective dropping* mechanism (Section 8.5).

Besides flooding, malicious nodes may try to manipulate the recommendation scheme by manipulating recommendations as, for instance, in a Sybil attack. By carefully deciding which and how many recommendations to take into account during mirror selection, SOUP offers protection against such adversary behavior.

The mirror selection is thus secure.

8.2 Mirror Candidate Ranking in the Bootstrapping Mode

SOUP provides a mechanism to allow new nodes to quickly achieve high data availability. At the time a node joins the OSN, it does not possess any information about well-suited mirrors. However, as it contacts other nodes, these nodes can suggest such mirrors to the new node. Here, the fact that OSN users are most active when they have just joined, and they contact many other nodes is exploited [74]. More specifically, every time a new node u contacts a node v, v suggests the set of mirrors that works well for itself to u. The initial suggestions will thus be received from the bootstrapping node, which u used to join the network. If for some reason u cannot obtain any recommendations, it will randomly select mirrors from its contacts.

However, a node should not use the bootstrapping mode for too long. A node w suggested by v might not be a good choice for u for various reasons:

(i) Node behavior in OSNs is heterogeneous (e.g., with regards to online time [156]) and w is probably not the best fitting node for u.

(ii) Moreover, w might not be willing to store data for u in the first place, or may have a limited storage capacity and has to drop some of the data it previously accepted.

(iii) Finally, a malicious node could fake recommendations and lure others to store their data at its site.

8.3 Mirror Candidate Ranking in the Regular Mode

SOUP's regular mode makes use of knowledge that a node does not have during bootstrapping, but can obtain after it has established social relations to other users. It will then leverage observations of these users to rank mirror candidates.

As illustrated in Figure 8.1, a node u in regular mode maintains two data structures: a ***Knowledge Base*** **(KB)** and ***Experience Set*** **(ES)**. In the knowledge base, every entry is about a node that u knows. With regard to an entry for node v, if v is a mirror of u, u will record an *experience value* (exp_v) based on u's friends' experience regarding v in the KB. A node w is *friends* with u if there is an edge (u,w) in the OSN's social graph G, which represents a social connection between both nodes. The experience value is the basis for ranking mirrors.

KB_u	Node v	sr(u,v)	exp_v
	w	1	-

$ES_u(w)$	Friend f	Mirror(f)	# of req	succ

(a) Initial State

Node v	sr(u,v)	exp_v
w	1	-
x	0	-
y	0	-

Friend f	Mirror(f)	# of req	succ
w	v_1	12	8
w	**v_2**	**10**	**1**
w	v_3	10	9

(b) First Observation Recordings

Node v	sr(u,v)	exp_v
w	1	0.67
x	0	0.22
y	0	0.90

Friend f	Mirror(f)	# of req	succ
w	v_1	14	10
w	**v_4**	**7**	**6**
w	v_3	8	8

(c) After Exchange of Experience Sets

Figure 8.1: Maintenance of knowledge base KB (top table) and experience sets ES (bottom table) at node u. Initially, u only knows one node (node w in (a)), which is also friends with u (i.e., $sr(u,w) = 1$). As u learns about new nodes, it adds them to KB_u (e.g., x, y in (b)). For each friend, node u further observes the performance of the friend's mirrors and records its experiences in $ES_u(friend)$ (e.g., w in (b)). u also receives $ES_j(u)$ from each friend j, allowing u to calculate the experience ranking for each node in KB_u (c). As u continues to record its own experiences for friend nodes (c), node w has replaced node v_2—for which u observed a bad performance—with node v_4.

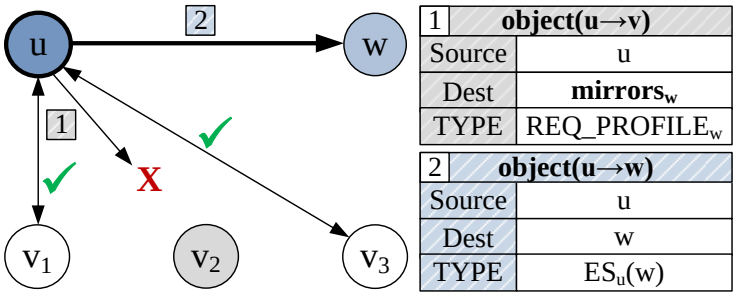


Figure 8.2: An example for a recording of Experience Sets. Here, node w has selected the nodes v_1,v_2 and v_3 as mirrors for her data. During the time in which w is offline, node u tries to request w's data from different mirrors. While u is successful in retrieving the desired data from v_1 and v_3, a request towards v_2 fails. u records these observations in the experience sets and periodically transmits the collected sets to w. Based on all collected experience sets submitted by her friends, w can then rank its existing mirrors and react to their performance. In case of a bad performance, w will increase mirrors or select different nodes as mirrors, w can reduce the mirrors in case of a good performance.

In addition, the entry for v will record whether or not v is friends with u and a *Time-To-Live* (TTL) value that decreases every time u does not choose v as a mirror (TTL not shown in Figure 8.1).

Also, for every node w that is a friend of u, u records an experience set $ES_u(w)$ as shown in Figure 8.2. This set records u's observations of w's mirrors; that is, when requesting w's data (Step 1 in Figure 8.2), u records whether or not the data is available at w's mirrors (see Figure 8.1b). It will then periodically transmit its experiences to w (Step 2).

Besides confining overhead, the experience set exchange is limited to friends for two further reasons:

(i) Users request the their friends' profiles more often than those of strangers. This way, they can record experience sets on the fly when requesting the data anyway.

(ii) The limitation raises the bar for malicious nodes trying to perform slander, as they have to establish social connections to their victims first.

Thus, for every node j that u is friends with, u will receive an experience set $ES_j(u)$ from j, which includes j's observations about u's mirrors. For any mirror v, node u can then calculate v's experience value—which also serves as v's ranking—as:

$$exp_v = (1-\alpha) \cdot exp_v^{old} + \alpha \cdot \frac{1}{n} \sum_{j=1}^{n} \cdot \frac{o_{(j,v)} \cdot av_{(j,v)}}{o_{max}} \tag{8.3.1}$$

where n is the number of experience sets that u's friends have reported, $o_{(j,v)}$ is the number of observations regarding v that a friend j is reporting since the last experience set exchange, o_{max} is the maximum number of observations that j can report, and $av_{(j,v)} \in [0,1]$ is the availability of v during j's requests of u's data. Equation (8.3.1) was designed with respect to a trade-off between accuracy and security:

- **Accuracy.** u usually does not care *who* tried to access its data (especially since the exchange of observations is limited to friend nodes). Instead, it cares for the number of attempts and successes of each reporting node in receiving its data. For instance, in retrieving u's data from v, if one node reports 100% success (9 successes out of 9 attempts) while another reports 0% (0 out of 1), v should not receive a mediocre ranking since in total 9 out of 10 attempts succeeded.
- **Security.** However, a single malicious node or multiple collaborating Sybil accounts could report huge numbers of manipulated observations, outweighing a lot of regularly observing nodes. Recall that existing defenses against vote manipulations by Sybils have been shown to be ineffective (Chapter 6). Hence, the maximum number of observations, o_{max}, that a node can report is limited to confine the influence of a few (potentially malicious) nodes in this calculation.

With exp_v computed as above, a significant portion of the recommending nodes, i.e., the friends of u, need to be malicious to have an impact on the selection scheme, while the experience from nodes with more observations still carries more weight.

Finally, α is the aging factor of observations, and a more recent observation carries more weight than an older one (exp_v^{old}). Otherwise, a malicious node could perform a traitor attack, where it obtains an excellent reputation just to exploit it afterwards. In particular, such a node could offer exceptional storage capacities and online time, and thereby likely getting selected as a mirror by many users, just to disappear later. Or, the quality of a mirror could suddenly deteriorate because of accidental reasons like connectivity problems. Applying the aging factor supports quick adaption to such situations. However, α should also not be over-valued, since a performance degradation can be temporary as well. When evaluating α, it appears that observing only the most recent observations might in fact lead to unstable mirror sets. Choosing $\alpha = 0.75$ provided the best experimental trade-off between adaptation and stability.

Algorithm 1 The mirror selection at node u. Given a ranked list of candidates for u, C_u, u will select the highest ranked nodes from C_u until the estimated data request failure from the already selected mirrors is below the target error rate ε. u will then apply the social filter β to the not selected follow-up nodes in C_u who are friends with u. The filter checks whether or not these nodes perform close to a selected mirror node, with whom u is not friends. If so, the mirror node is replaced. Finally, u adds a random node to the mirror set to prevent the overlooking of better nodes.

M_u: set of u's mirrors, initially empty
C_u: a ranked list of mirror candidates
r_v: a candidate v's ranking value
 # Select nodes from C_u
$p_{err} \leftarrow 1$
while $p_{err} > \varepsilon$ **do**
  add next top ranked element v from C_u to M_u
  $p_{err} = p_{err} \cdot (1 - r_v)$
end while
 # Apply social filter to nodes in M_u
 # ($sr(u,v)$=1 if u is friends with v; 0 otherwise.)
for all $v \in M_u$ that $sr(u,v) = 0$ **do**
  if $\exists v' \in (KB_u - M_u)$ such that
  $sr(u,v') = 1$ and $r_{v'} \cdot \beta > r_v$ **then**
    replace v with v' in M_u
  end if
end for
 # Prevent overlooking better nodes
add to M_u a random node v''
return M_u

8.4 Choosing Mirrors from the Ranking

Once a node obtains the ranking of mirror candidates from either mode, it selects its mirrors from the candidates, as depicted in Algorithm 1. It has a *target error rate*, ε, such that the probability of her data being unavailable is less than ε. First, it adds the top-ranked candidate nodes to its mirror set one by one, until the estimated likelihood of the data not being available is less than a target error rate ε:

$$p_{err} = \prod_{i=1}^{n} (1 - r_i) < \varepsilon \tag{8.4.1}$$

where r_i is the experience value of the i-th node in the candidate list.

Second, SOUP further exploits the inherent OSN incentives where nodes would rather prefer to store data of a friend than a stranger. The node applies a *social filter* to raise the

ranking values of its friends:

$$r_v = max\left(\beta \cdot r_v, 1\right), \quad where \;\; \beta > 1. \tag{8.4.2}$$

Friend nodes will thus move up in the ranking and can even replace some unrelated nodes as mirrors. The usage of this social incentive, however, must not be over-stretched. Evaluating β shows that a friend has to provide at least 80% performance of unrelated mirrors ($\beta \approx 1.25$) in order to offer the best availability and overhead, i.e., it cannot be significantly inferior to the unrelated nodes. For lower rankings, the performance gap exceeds the lesser dropping probability due to the social relation. Note that, in contrast to related works, nodes with few or low-ranked friends are not discriminated by the social filter and can still achieve high availability. The filter is rather an option for those nodes with highly ranked befriended mirror candidates. Finally, u adds to its mirror set a random node for which it has not yet determined a ranking (e.g., a new entry in its knowledge base). This way, u prevents a possible overlooking of even better suited nodes.

8.5 Protective Dropping of Data at Mirrors

A mirror node v may not always have enough space to store the data for another node, say u. While v can simply neglect u's storage request, alternatively, v can also drop another node's data to make more space for u. Dropping will not only provide more flexibility, it will also enable v to choose what data to store.

If a miscreant orchestrates a Sybil attack and floods the OSN with a large amount of storage requests, v may quickly fill up its storage space. Existing countermeasures against such attacks have been shown to be ineffective in Chapter 6. Therefore, in SOUP, each mirror node v implements a dropping policy that favors friends. The reasoning is that, although a user might accidentally befriend some Sybils, the majority of her friends will remain honest.

Note that this does not contradict the findings of Chapter 6, as it showed that a Sybil only needs to create one or two attack edges to *any* honest nodes in the OSN to circumvent existing Sybil defense solutions—and not that a majority of a user's relations are usually compromised.

v can thus drop the profiles of (possibly malicious) strangers, leaving space for the data of friends.

On the downside, this practice alone would discriminate honest nodes without or with few social relations, since these nodes need to rely on non-friend nodes. Moreover, malicious nodes can obtain social relations with some victim nodes in any case (see Chapter 6), and

Table 8.1: Protective dropping notations.

Symbol	Meaning
v	Node at which storage is exhausted
w	Node that has stored a replica at v
d_w	Dropping score for replica of node w
ß	Social filter
Θ	Blacklisting threshold
c	Mismatch increase (constant)

are thus able to flood the victims. Therefore, for every node w that stores its data at node v, node v calculates a dropping score, d_w, regarding w's data as follows (with notations given in Table 8.1, and pseudo-code listed in Algorithm 2):

- As v exchanges experience sets with each friend, say u, it also learns which nodes store their data at u.
- If a node w also stores its data at u, d_w is increased by 1. To protect the data of friends, their score is decreased by $1/\beta$ (recall that $\beta \approx 1.25$). If w is a flooder and tries to store its data on as many nodes as possible, d_w will then be high, and w's data will incur a high dropping probability. (Also, consider two benign nodes w, w' where w has a larger mirror set. Since v generally contributes less to the overall availability of w than to that of w', dropping the data of w has less impact than dropping the data of w'.)
- If v observes a copy of w's data in itself, but v is not listed in w's published mirror set, it increases d_w by a large constant c, as such a mismatch between the announced (i.e., published in the DHT) and the real mirror set may indicate a flooding attempt.
- If d_w reaches a threshold θ, node v then blacklists w from storing its data on v.

The threshold can vary depending on the willingness to avoid false positives, which can occur due to network errors, e.g., an error when publishing a new set of mirrors. The experiments provided the best results with a three-strike principle, in which $\theta = 300$ and $c = 100$. Thus, a node w will be blacklisted at v after v observed three mismatched mirror sets.

8.6 Chapter Summary

SOUP's mirror selection is supposed to offer a robust and secure DOSN, and offers the following features to justify that claim:

(i) it does not discriminate against any node as all nodes can select as many mirrors as needed to obtain high data availability.

(ii) in limiting the quantity of observations a single (possibly malicious) friend node can report, it offers a protection against the Sybil attack.

(iii) its protective dropping mechanism offers both protection of friends' data without discriminating against badly connected users, and protection against DDoS attacks by flooding.

However, it remains unclear how well these design principles help in achieving high and robust data availability, and how efficient malicious behavior is mitigated by them. The following chapter will thus evaluate every part of the mirror selection in detail.

Algorithm 2 Protective Dropping at a Node v (invoked when storage space is full)

c_v: storage capacity of v
R_v, R_u: set of replicas stored at v and u
G_v: set of nodes with social links to v
d_w: dropping score for replica of node w
M_w: set of mirrors of w
θ: ban threshold
β: social filter
ϕ: increase of d_w in case of a mismatch
B_v: ban list at v
$d_w \leftarrow 0$:
 # Compare replica sets and compute $p_d(w)$
for all $w \in R_v$ **do**
  for all $u \in G_v$ **do**
    retrieve R_u
    if $w \in R_u$ **then**
      $d_w \leftarrow d_w + 1 \cdot \beta$
      if $u \notin M_w$ **then**
        $d_w \leftarrow d_w + \phi$
      end if
    end if
    # If threshold is reached, add node to ban list
    if $d_w > \theta$ **then**
      $w \rightarrow B_v$
    end if
  end for
end for
drop $data_w$ with highest d_w
return updated R_v, B_v

Chapter 9

SOUP - Simulation and Analysis

SOUP's mirror selection scheme allows each OSN user to dynamically select as many mirrors as required to reach an *estimated* availability probability of above 99%. It considers parameters such as friendships among users, the users' collaboration, and the timeliness of their recommendations. At the same time, protective dropping provides mirrors with options to control which data they store, and malicious users are inhibited from attacking the system by various precautions.

In this chapter the selection procedure and its parameters are extensively evaluated by a large-scale simulation. The analysis of the results pays particular attention to the challenges SOUP and its mirror selection scheme face. The experiments with three real-world datasets show that SOUP provides high data availability with low overhead, and does so for *all* nodes in the OSN. SOUP performs even better when altruistic nodes exist, and successfully copes with node churn and malicious attacks. Additionally, SOUP is measured against comparable related works to also show its superior quantitative performance.

Contents

Table 9.1: The large-scale datasets used for SOUP's evaluation.

OSN	Nodes	Edges	Avg. Degree
Facebook	90,269	3,646,662	40.40
Epinions	75,879	508,837	6.71
Slashdot	82,169	948,464	11.54

9.1 Metrics, Datasets, and Methodology

To evaluate SOUP's mirror selection scheme, expressive metrics for its performance have to be specified first. In essence, these metrics should be able to express that SOUP efficiently solves its challenges. Hence, in the succeeding sections, the following two basic performance metrics are used:

(i) **Data availability** at time t: The ratio of the number of users whose data is available at time t to the total number of users in the OSN.

(ii) **Replica overhead** at time t: The average number of replicas each OSN node has at time t.

With regards to these metrics, SOUP is considered to offer high performance if the data availability is high while the replica overhead is kept low. As SOUP should achieve high performance at all times for all users, regardless of any external influence, the two performance metrics can then be used to measure SOUP's robustness, adaptivity, and resiliency:

(iii) **Robustness.** SOUP's ability to provide high performance to all nodes in an OSN, regardless of a node's online time, social relations and device capabilities.

(iv) **Adaptivity.** SOUP's ability to increase performance when altruistically provided resources are available.

(v) **Resiliency.** SOUP's ability to maintain high performance when facing adverse scenarios, including attacks by malicious users.

Three different large-scale datasets as listed in Table 9.1 are used to evaluate SOUP's performance with regards to these metrics. These datasets cover a variety of real-world social graph features and can help evaluate SOUP's performance in different contexts. For

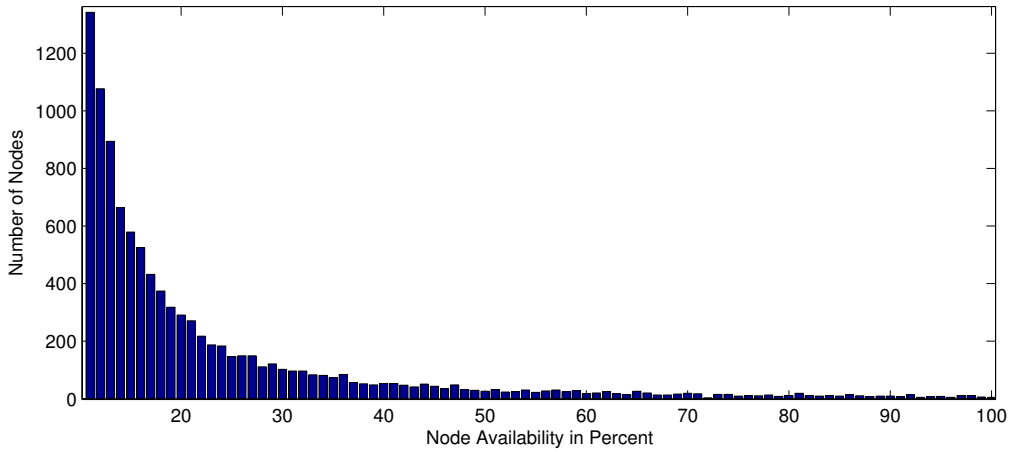


Figure 9.1: A node online time probability distribution which follows a power-law distribution. Most nodes are rarely online (towards the left end of the figure), while only few nodes are highly available (towards the right end of the figure).

instance, users should not depend on their number of friends. Therefore, the Epinions data set with an average node degree of only 17% of that in the Facebook set was chosen to validate that SOUP can also perform well for users without many friends. Moreover, all datasets provide a sufficiently large user population to evaluate whether SOUP is scalable. SOUP is simulated under the metrics defined above by using these datasets.

Further, a number of parameters associated with each user are handled. One goal of the succeeding experiments is to give a estimation for the lower boundary of SOUP's performance. Thus, the following parameter setups favor a conservative parameter setting over the best case setting:

Target error rate ε. In the experiments, every user defines her target error rate as 0.01; i.e., every user aims at a 99% likelihood of her data being available (Section 8.4).

Node online probability. Each node's availability must be known at any given time to determine if a user's data is available at this node. Node online time in OSNs is based on bursty interaction patterns of users and typically follows a power-law distribution [74, 156, 179]. Therefore, in the following experiments, around 60% of the nodes are available less than 20% of the time, and there are only very few highly available nodes, as depicted in Figure 9.1 for a 10000-node example network. Note that this power-law model incorporates the high churn rates typical for an OSN, and thus all experimental results are obtained respecting this property. Different online time distributions are evaluated in Section 9.2.7.

Further, diurnal patterns are applied to populate the online time matrix of each node. According to [180], on Facebook there exist three major time zones (US, Europe and Africa, and Asia and Oceania), where a node's probability of belonging to these zones is 0.4, 0.3, and 0.3, respectively.

Additionally, complying with [82], the online time of a node is *not* correlated with its degree, i.e., the user's number of friends. Note that even if such a correlation would exist, it would rather improve the performance of SOUP, as highly available nodes with many social relations will contribute many experience sets. As a consequence, recommendation set values would be more precise during mirror selection.

Finally, to bootstrap, nodes join the experiments asynchronously according to their online probability. Hence, the first selection of mirrors will not occur instantly for most nodes due to the power-law distribution of online times.

User activity pattern. Another parameter that affects the calculation of recommendation sets is how often a user accesses the data of other users. Different evolutions of OSN user activity have been observed [74, 156]. As the most conservative model, user activity was chosen to be exponentially decreasing [74]. After an initial phase of high interaction once joining an OSN, a user's activity decreases exponentially to become less than one interaction per day. As nodes in SOUP must gain knowledge about other participants (i.e., get in contact with them) in order to find the best-suited mirrors, this model represents the worst observed case in literature, and thus contributes to the conservative parameter setting.

Available storage space per node. Finally, each node must have a specific storage space value in order to evaluate the storage overhead and dropping strategy of SOUP. This storage space available at each node is modeled as a Gaussian distribution, with a median of space for mirroring data of 50 users, which requires no more than half a gigabyte of disk space as shown later in Chapter 11.

The assignment is repeated five times per dataset to create 15 simulation scenarios in total. Afterwards, each scenario is executed multiple times, without assuming any correlation between the online time, activity and storage parameters.

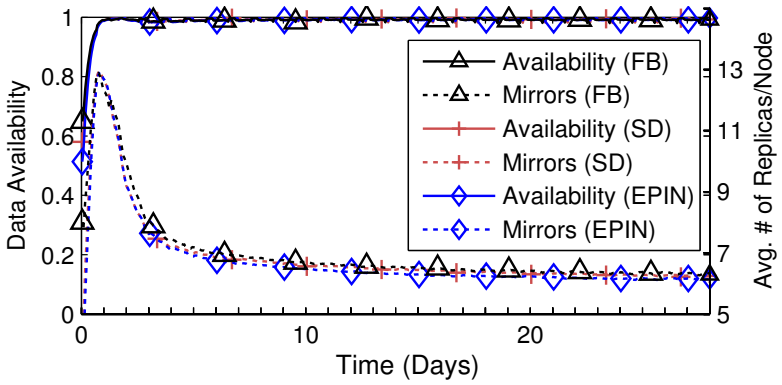


Figure 9.2: SOUP achieves high availability with low overhead.

9.2 Results and Analysis

9.2.1 Data Availability and Replica Overhead

The first step SOUP must take towards providing a robust and competitive DOSN is to offer high performance. Figure 9.2 shows SOUP's data availability and replica overhead for each dataset. In all the three datasets, SOUP achieves the targeted availability of above 99% after only one day, even though no node has any knowledge upon joining. As SOUP reaches equilibrium, the high level of availability is maintained for the entire remaining period.

Two distinct phases exist regarding the replica overhead. After joining, due to the lack of knowledge about good mirrors, nodes do not select well-suited mirrors yet, and the number of replicas increases. However, as soon as nodes obtain more precise rankings, the quality of mirrors improves and the replica overhead is reduced by about 50%. On average each node needs to store less than seven replicas.

9.2.2 Stability and Communication Overhead

SOUP needs to reach a stable state quickly in order to keep communication overhead low. In particular, if a user frequently changes her set of mirrors, all her data has to be transmitted to new mirrors often. SOUP's profile distribution is shown in Figure 9.3. For the ease of exposition, the results in the remainder of this chapter are obtained from the Facebook dataset if not explicitly stated otherwise. SOUP showed the same behavior for both other datasets.

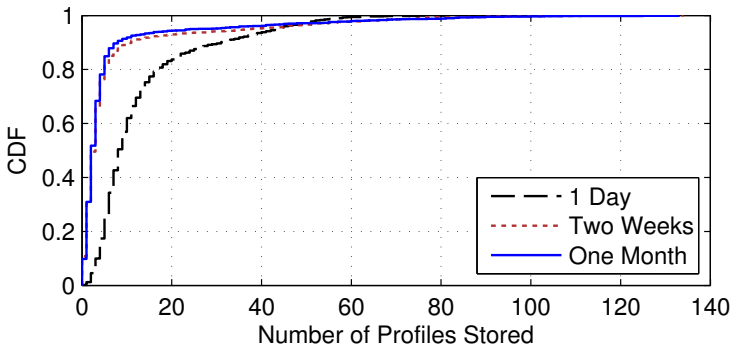


Figure 9.3: SOUP proves to be stable, and 90% of the users store less than seven replicas.

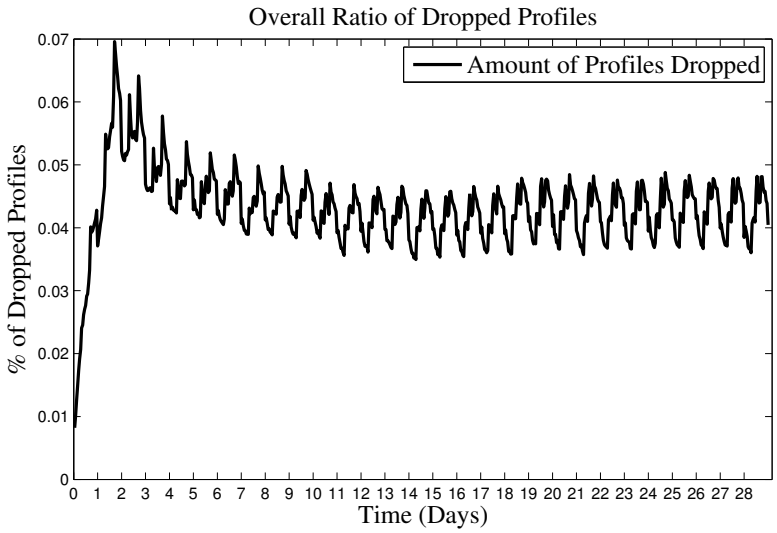


Figure 9.4: SOUP drops only a low amount of data.

Here, three snapshots are taken throughout the simulation: after the first day, after two weeks, and after one month. After day one, around half the nodes need to store 10 or more replicas so that the system achieves high availability. However, as user experiences are more accurately measured, 90% of the nodes need to store less than seven replicas (after two weeks). The same distribution is present at the end of the simulation, which indicates that SOUP has reached a stable state.

Further, while mirror rankings become more accurate, the drop rate of data converges from 0.07% to a very low 0.045%, as shown in Figure 9.4.

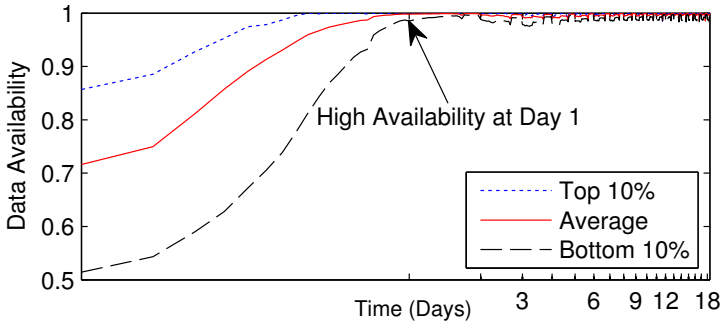


Figure 9.5: SOUP is robust and does not discriminate any node.

In other words, of 1000 profiles that are stored at mirrors, only about five are dropped. Finally, the upper half of our nodes with regards to online time provides more than 90% of all replicas. This indicates that weak nodes, in particular mobile nodes, are rarely chosen as mirrors, saving storage, bandwidth and battery on these devices.

9.2.3 Robustness

To offer a robust OSN, regardless of their own online probability or quantity of friends, every user should achieve a very high level of data availability in SOUP. The top and bottom 10% of users (first with regards to their own online probability and second with regards to their number of friends) are compared with regards to their performance in Figure 9.5. After just one day, even the bottom 10% of users obtain data availability of above 99%. Hence, in contrast to some related works (e.g., [42, 43, 45, 46]), users are discriminated neither based on their own online time nor based on the quantity and quality of their social relations.

A side observation is that, while SOUP achieves high availability even for nodes that are almost never online, inactive, or maintain no social relations, most of the few total losses also occur for such *singletons*, whose data is of little interest for the active users anyway (see Chapter 2.1.1).

9.2.4 Adaptivity

One of SOUP's challenges is to exploit altruistically provided resources efficiently. In another experiment, altruistic nodes are injected into the running system. An *altruistic node* is a regular machine, which is steadily online, but does *not* provide high storage capacity as a

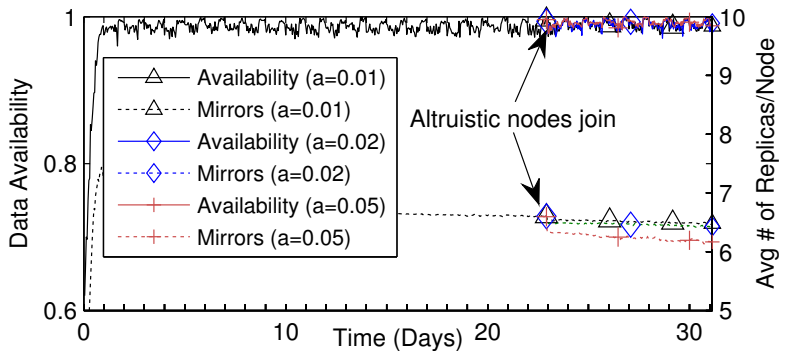


Figure 9.6: SOUP can exploit altruistic resources.

server would do. Figure 9.6 shows the impact of the presence of small percentages of such nodes. It appears that 5% (a=0.05) altruistic nodes can cause a slight increase and stabilization of availability, but the improvement in terms of replica overhead is more prominent. As altruistic nodes become known to the OSN, nodes can select fewer mirrors than before to achieve the same level of availability (recall that each node still aims for 99% availability). Hence, while SOUP does not rely on any kind of altruistic nodes, it can exploit such nodes' resources if available.

9.2.5 Resiliency Against Node Dynamics

To also offer a secure DOSN, SOUP has to be resilient when facing a variety of adverse scenarios. First, in addition to the high churn rates already included in the evaluation methodology, a fraction of the users might abruptly become unavailable to the rest of the OSN. The reasons for this can be manifold and range from accidental network and node failures to overloaded mirrors (e.g., if a mirror hosts a very popular profile) and DDoS attacks. To show SOUP's resiliency in such a case, the *best* nodes in terms of online time are assumed to be unavailable at the same time. The results of such an event are shown in Figure 9.7.

If the best 5% of nodes leave the OSN simultaneously (d=0.05), there is a noticeable drop in both data availability and replica overhead directly after the departure, caused by the concomitant loss of mirrors within the OSN. However, the remaining nodes adapt quickly by choosing new mirrors, and SOUP's performance improves without introducing any additional replica overhead.

Interestingly, SOUP is independent from the top 1-2% of nodes, as data availability does not significantly drop as these nodes leave the OSN. Thus, DDoS attacks against the system itself are difficult to execute, as even shutting down the top 5% of nodes will not be

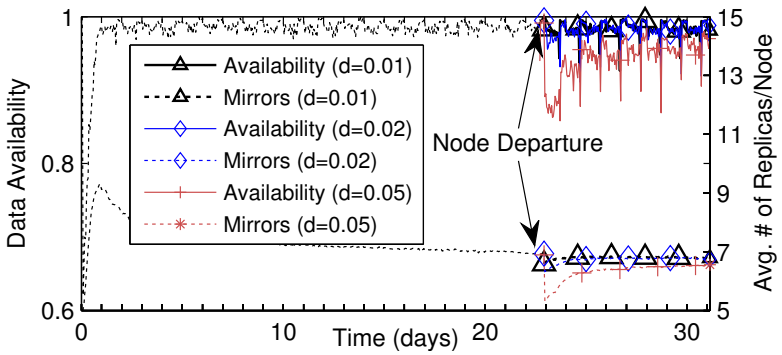


Figure 9.7: SOUP is resilient against node dynamics.

troublesome.

Even though the system itself is resilient to DDoS attacks to this extent, a specific profile might be unavailable, either when an adversary attacks its mirrors or when mirrors of popular data deny service due to overloading. In such a case, these mirrors will receive a lower ranking, and SOUP will distribute the load among additional mirrors. If a mirror is completely taken down, SOUP will choose a different one, as shown above. Compared to the static mirror choices of related work, SOUP is the only approach capable of such adaptation towards both increasing and decreasing resources.

9.2.6 Resiliency Against Malicious Nodes

Second, adversaries are not limited to DDoS attacks, but may also employ different attacks on SOUP's selection scheme as discussed in Chapter 8. To evaluate the resiliency of SOUP in this dimension, different fractions of the OSN are now assumed to be compromised. Due to increased potential to control a large number of nodes (e.g., by using cloud services), SOUP's performance is measured under attack of up to half the nodes in the OSN, whereas none of the existing DOSN solutions consider attacks on their system at all (Chapter 4). In this experiment, SOUP not only needs to tolerate the attackers after having stabilized, but also has to bootstrap in their presence.

The first attack under investigation is a *slander attack*, in which attackers use fake identities to manipulate experience sets (or recommendations to bootstrapping users), such that these sets are the inversion of what the attackers observed in reality. For instance, a Sybil will report a low performance for a well performing honest mirror, whereas it will do the opposite for a badly performing mirror.

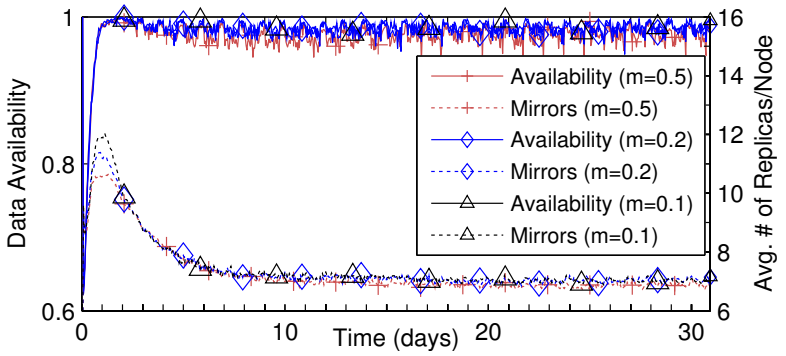


Figure 9.8: SOUP is resilient against a slander attack.

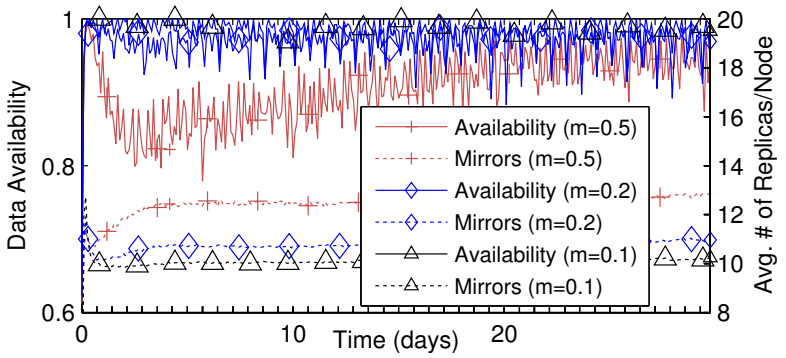


Figure 9.9: SOUP can recover from a flooding attack.

The Sybils are assumed to have infiltrated the OSN successfully and they send out recommendations at the maximum rate (see Chapter 8.3). Figure 9.8 shows that even when 50% of social relations—and thereby experience sets—are subject to slander, the data availability at most drops to around 95% ($m = 0.5$).

The second considered attack deals with a *flooding* adversary, which uses the fake identities (*Sybils*) to flood benign mirrors with data. The results for different percentages of Sybils are shown in Figure 9.9. Even with as many Sybils as regular identities in the OSN, the data availability does not drop below 90% for the benign users in the long run. The replica overhead, although increased, does not exceed 13 copies per node. In this case, protective dropping prevents data of socially connected nodes from being dropped for a Sybil's data, and avoids the full utilization of resources at benign nodes.

In addition, SOUP can even help to detect Sybils with time passing. With an increasing number of attackers being identified, SOUP is able to recover from the attack. Hence,

Table 9.2: SOUP outperforms related work (p = online probability).

Approach	Online Time Assumption	Availability	Number of Replicas
SOUP	Power-law	~99.5%	6.5
Safebook	Uniform, all nodes p=0.3	~90.0%	13-24
SOUP		~98.5%	14
PeerSoN	10% of nodes p=0.9 25% of nodes p=0.87 30% of nodes p=0.75	<90%-100%; Depends on p	6
SOUP	30% of nodes p=0.3	~100%	4

although storing some copies of malicious nodes, even a flooding attack with half of the identities in the OSN being malicious can be tolerated quite well.

9.2.7 SOUP versus Related Work

SOUP's superiority over state-of-the-art solutions mainly stems from its *qualitative* properties, which were extensively evaluated above. Compared against those, SOUP is robust, adaptive to node dynamics, and resilient against attacks.

To further compare the performance of SOUP and related work *quantitatively*, SOUP is now simulated under the node online time distributions assumed in related works, in those cases where the distributions were available. As shown in Table 9.2, SOUP outperforms both PeerSoN and Safebook, providing higher data availability and lower replica overhead. In particular, when compared with Safebook, SOUP achieved 8.5% higher availability while keeping the replica overhead near the lower bound of Safebook. In this scenario, SOUP performs slightly worse than in the original experiments. This is caused by the uniform online time distribution, due to which SOUP cannot exploit the heterogeneity of node characteristics to select well-suited mirrors.

In the PeerSoN scenario, the online times of nodes are drastically improved over SOUP's power-law assumption. Still, PeerSoN is not able to create a robust OSN and the data availability ranges between less than 90% and close to 100%, as nodes depend on their own online times. SOUP however provides close to 100% data availability for *all* nodes and further reduces the replica overhead by one third.

9.3 Summary of Addressed Challenges

In essence, the results of the large-scale simulation of SOUP show that SOUP addresses the following challenges:

(i) SOUP is robust in the sense that all data of all users is available at (almost) all times. It does not introduce user payments or excessive *storage* overhead.

(ii) SOUP is secure in the sense that its performance does not significantly deteoriate in the presence of large numbers of malicious users, which execute both Sybil attacks on the mirror selection scheme and DDoS attacks on the mirrors themselves.

(iii) It is adaptive to altruistically provided resources.

(iv) SOUP's mirror selection exploits the social links between OSN users.

However, besides storage overhead, SOUP also introduces *bandwidth and processing* overhead. For SOUP to be feasible, both overheads have to remain within reasonable boundaries. Therefore, the challenge of low overhead is only partially solved so far. To realistically measure SOUP in terms of bandwidth and processing overhead, SOUP has to be implemented and deployed.

Chapter 10

SOUP - Implementation

The large-scale simulation of SOUP's mirror selection scheme showed that the approach in fact effectively addresses many drawbacks of existing DOSNs. In a next step, this chapter describes the implementation of SOUP, including its mirror selection scheme. The implementation of a SOUP node comprises two components: the *SOUP middleware* and the *SOUP applications*. To make SOUP available to mobile users as well, both components have also been implemented on Android.

In the big picture, there exists a clear separation between the SOUP middleware and its applications. The middleware is responsible for joining the overlay of SOUP users as well as for altering encrypted social data and for securely communicating with other nodes. Further, the middleware also takes care of the selection of mirrors and provides the functionality to enable data synchronization.

The applications on the other hand exploit the functionality offered by the middleware and allow users to interact with it via graphical interfaces. They create and modify user data, which is then managed by the middleware, and can exchange their own data transparent to the middleware.

Contents

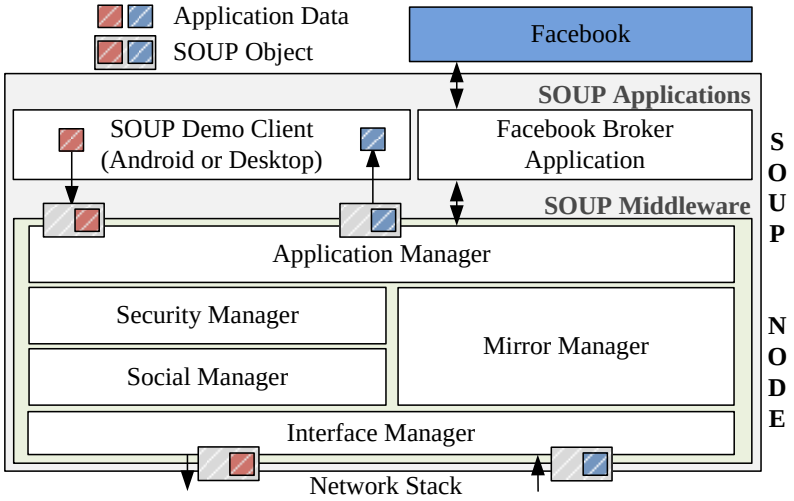


Figure 10.1: Architecture of a SOUP Node. The node consists of the modularly organized SOUP middleware and SOUP applications, which run on top of the middleware. The two components have been implemented for both desktop and mobile use.

10.1 Implementation of the SOUP Middleware

The SOUP middleware consists of several modules as shown in Figure 10.1, of which each is responsible for a pre-defined task and designed to be easily exchanged for an improved or different approach at any time. All modules communicate by passing SOUP objects to each other.

The code of the middleware is lightweight and currently contains around 3400 *Source Lines of Code* (SLOC) in JAVA. FreePastry 2.1[12], an open source implementation of the Pastry DHT [102], is used as the underlying DHT. Most of the code and executables are available online.[13]

Note that it is possible to exclusively run this middleware component—for instance, to provide an altruistic node, which only acts as a mirror or just relays DHT requests for mobile nodes.

[12] http://www.freepastry.org
[13] http://soup.informatik.uni-goettingen.de/

10.1.1 Application Manager

The application manager is the northbound interface of the SOUP middleware and provides an API to SOUP applications. It has two functionalities:

(i) it allows arbitrary social applications to run on top of the SOUP middleware; and

(ii) it enables communication between applications transparent to the middleware itself.

On one hand, it encapsulates content from a SOUP application into SOUP objects, which are then further processed by other modules. On the other hand, it decapsulates content destined for an application from SOUP objects received from other modules.

In the following sections, the life cycle of a SOUP object, starting at its creation and ending at its transmission to the network stack is followed. The accompanying example will be a friend request issued by a user Alice and targeted at a user Bob.

The ASCII protocol for communication between the middleware and the applications is kept simple but yet expressive to allow application developers to easily implement new OSN functionality. To illustrate the simplicity of the protocol, an example code snippet for the friend request is shown in Listing 10.1. Here, in only 5 SLOC, the application needs to indicate that a new request has been issued by the user, and then tell the middleware the SOUP ID of the target user. The application manager will then create a SOUP object which informs the social manager that a change to the friendlist of Alice is imminent.

Listing 10.1: Required code snippet to be written by the application developer to implement a friend request

```
1 //indicate intend to add a friendship
2 writeInt(SOUPCode.ADD_RELATION);
3 //wait for ACK by SOUP
4 isNotACKedThenThrow(bis);
5 //write target's SOUP ID
6 write(Bob.SOUPId.getBytes());
7 flush();
8 //wait for ACK by SOUP
9 isNotACKedThenThrow(bis);
```

10.1.2 Social Manager

The social manager module is responsible for processing requests when a SOUP object indicates a change to the social data. Such a change might either be an update to the actual user data, or an update to the social graph, such as a friend request.

In the example, the social manager will alter Alice's friendlist by adding an unconfirmed social relation. It will further create a new SOUP object with Bob as destination, indicating that Alice wants to establish a friendship with Bob.

10.1.3 Security Manager

However, the social graph (and all other user data) is encrypted by default. To enable SOUP operations on encrypted data, the security manager module deals with all encryption-related tasks. It uses an ABE implementation which has been optimized to lower the cryptographic overhead for this thesis.[14]

For instance, when receiving the friend request, the social manager cooperates with the security manager to decrypt Alice's friendlist, to add the request, and to re-encrypt the list. Note that the data is encrypted with a symmetric key, while the key itself is protected by the ABE *Access Structure* to prevent unauthorized access to the friendlist (see Chapter 7.6).

Also, SOUP objects exchanged between different SOUP middlewares need to be signed. The security manager thus also signs the outgoing SOUP object towards Bob with Alice's private key.

10.1.4 Mirror Manager

Changes to social data might require a re-distribution of the data to the user's mirrors, for which the mirror manager module is responsible. For instance, once Bob has confirmed the friendship with Alice, Alice needs to redistribute her updated friendlist to her mirrors. Only then can she retrieve her most recent social data in case of, e.g., switching to another device.

The mirror manager will thus try to update Alice's data at the mirrors, or, if a mirror is unavailable, store an update for that mirror (see Chapter 7.7). At the same time, it also takes care of SOUP's key element, the selection of mirrors as introduced in Chapter 8.

[14] Based on https://github.com/wakemecn/cpabe

10.1.5 Interface Manager

All SOUP objects that have to be transmitted to another node are ultimately passed to the southbound interface of SOUP, the interface manager, which can then initiate communication via a suitable network interface.

In the example, the security manager will hand the signed friend request object to the interface manager, which is then responsible to forward the request to Bob over an appropriate link. Additionally, it will also receive the SOUP objects containing the updated friendlist, which have to be sent to Alice's mirror nodes.

To find out the available interfaces of a target node, the interface manager can lookup each target node's entry in the DHT.

If the interface manager later receives an encrypted request confirmation object from Bob, it forwards it to the security manager, which unlocks the object and issues a confirmation to the application via the application manager. The security manager will further forward the confirmation of the request to the social manager, which in turn will update Alice's friendlist accordingly.

10.2 Implementation of Exemplary SOUP Applications

To make use of the functionality offered by the SOUP middleware, a demo client has been implemented as a SOUP application. It supports essential OSN functionalities: Users can search for each other in the OSN, establish friendships, view other users' profiles, share photos, and exchange messages. It implements all functionality provided by the middleware's API (see Chapter 7.4), which requires less than 350 SLOC (plus approximately 3000 SLOC for the graphical user interface).[15]

Next to the demo application, a small broker application, which allows for a soft transition between Facebook and SOUP, has been implemented. It tries to deliver content created in SOUP to Facebook and vice versa, as far as the Facebook API allows. The application is further capable of looking up possible matching contacts in SOUP when fed with a Facebook friend list. Note however that the Facebook API is constantly changing. The changes introduced by each update are at times major, and can restrict the ability of the transition application to transfer content.

[15] A video showing the demo application and its interplay with the middleware can be found at: `https://www.youtube.com/watch?v=bxfVvSXC2WY`

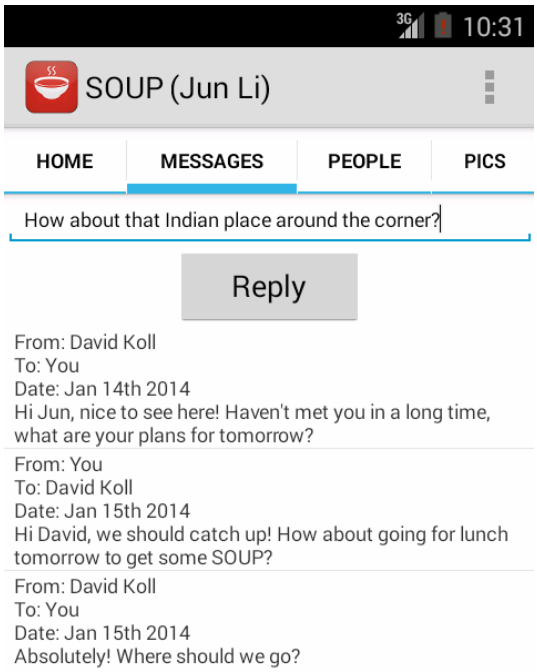


Figure 10.2: SOUP on a Nexus 4 Android Phone

10.3 Implementation of SOUP on Android

Both the SOUP middleware and the SOUP demo application have also been implemented on the Android mobile platform. During the porting, particular attention was paid to the clear separation between the middleware and its applications. Hence, the middleware was implemented as an Android service, which predominantly runs in the background. To ensure the mobile-friendliness of SOUP, it enables mobile nodes to relay their DHT requests through a fixed gateway node as introduced in Chapter 7.5.

The demo application implements the same ASCII protocol to communicate with the service in the same way a desktop application would communicate with the desktop middleware. Figure 10.2 shows a screenshot of the demo application running on top of the SOUP middleware service on a Nexus 4 Android phone. While its graphical user interface is kept simple, the application essentially offers the same functionality as the desktop demo application.

Chapter 11

Deploying SOUP in the Wild

In the final step of evaluating SOUP, the middleware and applications as implemented in the previous chapter have been deployed in order to build a real-world DOSN. The goal of this deployment is to measure the performance of SOUP in further parameters, which cannot be evaluated by simulation, but are critical to show that SOUP is feasible and does not face significant deployment obstacles.

First and foremost, these parameters include the bandwidth consumption of SOUP in several scenarios, but also its cryptographic and storage overhead along with its application level latency.

Contents

11.1 Deployment Setup and Methodology

To demonstrate the feasibility and deployability of SOUP, a prototype of the system was deployed to a real-world DOSN consisting of 31 users. Four of those were using different Android mobile phones. All phones were relaying via the same gateway node, and that node was further a bootstrapping node for all users running the regular SOUP client. Data of several days was collected, during which the SOUP users established 282 friendships, shared 204 photos, and exchanged 1189 messages.

The deployment outperformed the simulation results with regards to availability—all data was available at all times, and there was no loss at all—and replica overhead. However, note that the large-scale simulations should be more accurate as the online times observed during the deployment were much longer online than typical for OSNs.

Instead, the focus of the measurements during the deployment was on learning about SOUP's performance in those parameters, for which an evaluation by simulation is not sufficient. These parameters mainly include its bandwidth consumption, cryptographic overhead, storage overhead, and application level latency.

11.2 Bandwidth Consumption

For SOUP to be feasible, it first and foremost must not introduce excessive bandwidth consumption. Here, the traffic that was originally flowing towards a sophisticated centralized infrastructure has to be absorbed by the participants themselves (in particular by the mirrors of data), as they form and maintain the SOUP overlay themselves. In particular, there are two kinds of overhead, for which SOUP must show that it is able to keep them within reasonable boundaries.

(i) First, the overhead to maintain the overlay itself must remain low. Recall that while maintaining the overlay, some nodes act as bootstrapping nodes or relay DHT requests for mobile nodes.

(ii) Also, the overhead that is induced due to the selection of mirrors must be manageable. Recall that most nodes select around seven other nodes as their replicas. The communication overhead must be limited at both the mirrors and the selecting node itself. Other participants may request data from the former, while the latter must not change its mirrors very often, since doing so results in a retransmission of the whole data to the new mirror(s).

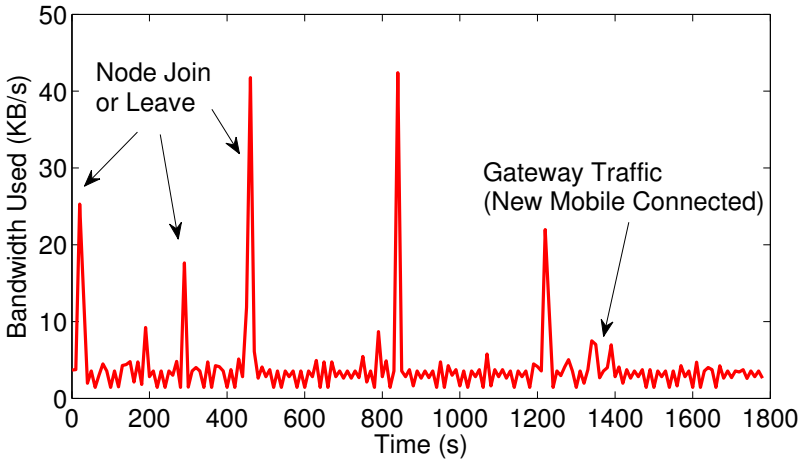


Figure 11.1: The control overhead introduced by SOUP is low.

11.2.1 Overlay Overhead

The bandwidth consumption of the DHT at the bootstrapping node is shown in Figure 11.1. The figure depicts those thirty minutes of the whole deployment in which the bootstrapping node—as the most utilized node in the deployment—experienced the most requests.

Recall that the bootstrapping node is not only responsible for booting new nodes into the DHT ring, but also acts as a gateway for all mobile nodes and additionally has to play its role as a regular node on the DHT, and is therefore responsible for queries towards a certain range in the key-space.

Overall, the overlay overhead is very low. Only upon join and leave operations (i.e., shifting some entries in the DHT) the network interface is utilized at around 20-40 KB/s. At the same time, lookups do not have a visual impact.

As a result, the cost of relaying for a mobile node (i.e., forwarding lookups and their results) is low as well. Only during the join procedure of a mobile node, which requires several DHT operations, relaying consumes a noticeable amount of bandwidth, which, however, still remains very low.

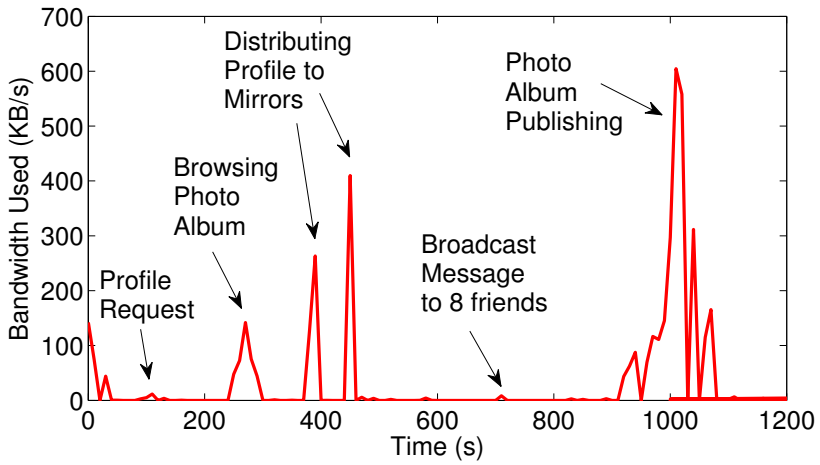


Figure 11.2: The communication overhead of SOUP remains manageable.

11.2.2 Mirroring Overhead

The traffic introduced by SOUP itself is also manageable. Figure 11.2 shows the most bandwidth intense period of 20 minutes observed for any user during the time of data collection. Messaging or simple profile requests do not consume a lot of bandwidth and are hardly distinguishable from an idle link. A more intense activity like skipping through a photo album does not consume a regular user's bandwidth as well, as she takes her time to view the pictures.

Note that this kind of data traffic, i.e., the traffic a user generates by *consuming* content, is approximately the same as in centralized OSNs, as the user needs to download the data in those systems as well. As shown above, the overhead introduced by SOUP to lookup the location of the data item of interest is low.

Only when *producing or mirroring* content, SOUP will generate additional traffic. The reason is that produced data has to be distributed to the mirrors as well. At the same time, the uplink of a user who acts as a mirror might be utilized. As a consequence, in Figure 11.2, the most traffic is generated at the creation of a photo album, and distributing that photo album to the user's mirrors. As in centralized OSNs, mobile users on a data plan should delay such uploads until they can access a WiFi link.

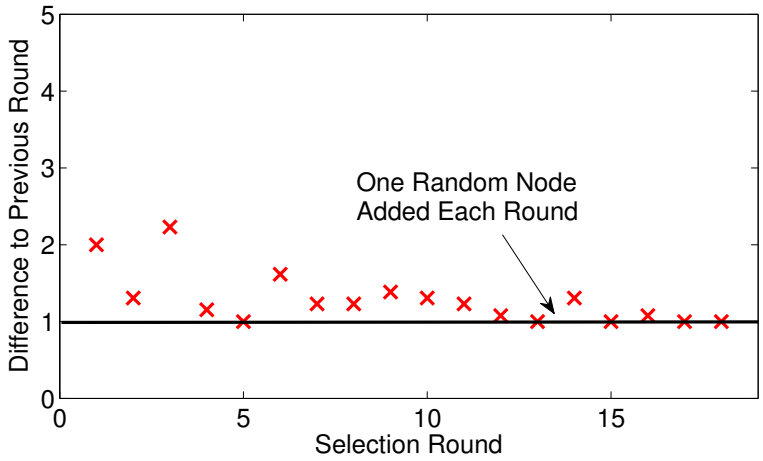


Figure 11.3: SOUP incurs little variance in Mirror Set.

11.2.3 Stability of Mirror Sets

It is further important for SOUP to offer stable mirror sets. The reason is that every time a user changes her mirrors, she needs to transmit all her data to the new mirror(s). At the same time, if there is no fluctuation in her mirror set, she can then transmit the—much smaller—updates to her data to her mirrors.

Figure 11.3 shows that, overall, the mirror sets remain stable and do not differ much between selection rounds. After the initial rounds, most mirror changes are additions of a random node as described in Chapter 8. Each round, only few nodes change additional mirrors. As a consequence, the whole data of a user does not have to be transmitted often, and the communication overhead remains modest.

Note that during the initial mirror selections many users might not have uploaded much data to the OSN yet. They have just joined the network and are interested in creating an audience for their data first. Since users find a well suited set of mirrors quickly, unnecessary retransmissions of a large set of multimedia data might thus not occur often.

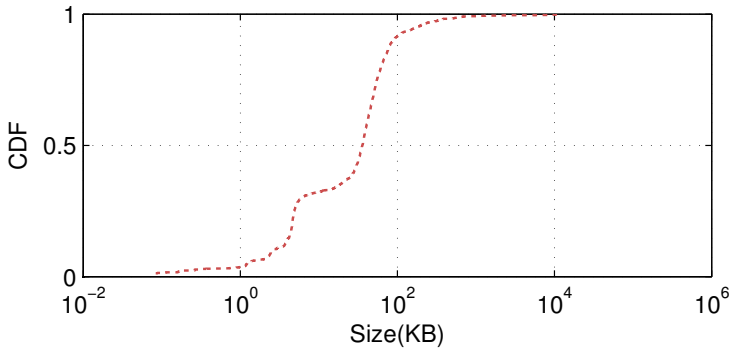


Figure 11.4: The CDF of item sizes in the collected dataset. Most items are text items and therefore relatively small in size. Only 1% of all collected items is larger than 1 MB.

11.2.4 Stress-testing SOUP

With regards to bandwidth consumption, the real-world deployment did not push SOUP to its limits. Thus, in another experiment, SOUP was specifically stress-tested. To emulate a higher load, one particular mirror was chosen to store a large amount of user data, which was then requested with much higher frequencies than during the deployment.

As the large amount of user data the complete Facebook and Google+ profiles of 20 users—including private data—were collected. These profiles offer details beyond a crawler's results, as those often do not include major parts of a user's data (e.g., photos on Facebook are not publicly available by default) [36].

The average profile size was approximately 10 MB, with the largest profile containing hundreds of photos in 27 photo albums and one video. This profile consumed 60 MB of disk space in total. At the same time, many profiles are small-sized, i.e., people do not upload much multimedia information. This coincides with observations that only few users in OSNs have a high degree and therefore are encouraged to share their data [156].

Overall, the data disclosed 2035 unique data items, for which Figure 11.4 shows the CDF of item sizes. More than 35% of all items are less than 10 KB in size, and 93%—including most images—are less than 100 KB in size, while large items rarely exist. These findings mainly coincide with those in [36]. The whole data sums up to 206 MB. Note that even if some profiles in the OSN happen to be much bigger, this would mainly affect the storage required at the mirrors, as long as the item sizes remain small.

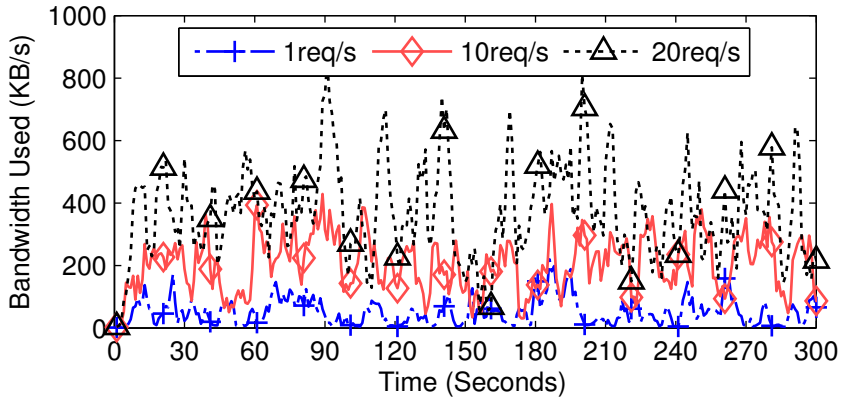


Figure 11.5: Bandwidth consumption at high request rates.

One mirror was then selected as a host for all this data. Recall that storing these 20 profiles is three times as much as 90% of SOUP nodes will have to store (see Chapter 9.2.2). Afterwards, the mirror received requests asking for text, photo and video data according to the request probabilities for each data type as described in [181].

As shown in Figure 11.5, the average bandwidth consumption is well below 600 KB/s, even if the mirror has to handle 20 requests per second. With an increasing request frequency large items are hit more often, which causes the spikes in the measurements.

As a result, a request might time out once a mirror becomes overloaded or limits its bandwidth, which may happen especially to nodes mirroring popular data. Note that unlike other approaches, in which users are stuck with a static or heteronomous set of mirrors [42–46], SOUP will adapt to this situation.

11.3 Cryptographic Overhead

Overall, the cryptographic overhead remains low, which coincides with the results obtained by Baden et al. in [36]. The processing times for SOUP's ABE operations are approximately 10% faster than existing JAVA implementations. When encrypting a key with four attributes (the processing time grows linearly with the number of attributes, independent of the data size [132, 182]), the 90th percentile encryption time was 262ms, whereas decryption was four times faster and took 61ms per data item. As improved libraries for encryption (e.g., DOSN-specific approaches [182]) become available, SOUP can implement these as a new version of the Security Manager.

11.4 Storage Overhead

Recall that the mirror chosen for the stress test of SOUP stores 20 profiles—three times of what 90% of SOUP nodes will have to handle. Still, the data sums up to 206 MB of consumed space, which is both surprisingly low and easily manageable, even for nodes with little resources. In fact, for 90% of all nodes, SOUP will only use as little as 70 MB of disk space. Even if all profiles are treated to be of the same size as the largest profile in the dataset, the consumed space is only 1.2 GB for the stress-tested mirror, and 390 MB for a node that stores SOUP's average of 6.5 profiles.

11.5 Latency and Processing Overhead

Finally, the *application level* latency that a client experiences when requesting the data from the mirror, which is under full load (12 requests per second), was measured. Regardless of the location of the client (distributed over several locations in the Amazon EC2 cloud), the request latency is $2 \cdot RTT + \varepsilon$, where 1 *Round-Trip-Time* (RTT) is due to TCP connection establishment, 1 RTT is caused by the data request over the established TCP connection, and $\varepsilon < 2ms$ is the actual processing overhead introduced by SOUP.

11.6 Summary of Addressed Challenges

The deployment shows that SOUP can indeed also fulfill the last remaining challenge, as it—next to a limited storage overhead—also operates with little bandwidth and processing overhead. Thus, in combinations with the results obtained in previous chapters, SOUP can enable high data availability for all users *with little overhead*.

Chapter 12

Discussion and Future Work

This thesis underlined that a promising DOSN approach needs to fulfill a number of challenges to be able to compete with centralized DOSNs. After presenting the design, evaluation, implementation and deployment of SOUP in the previous chapters, this last but one chapter starts with a summary of whether or not SOUP fulfills *all* these challenges.

The remainder of the chapter then mainly discusses possible improvements to SOUP, which are left for future work. Users could for instance exchange more meaningful recommendations by valuing certain recommendations higher than others, or by including more measurements than just the mere success or failure to retrieve data at a mirror's site.

Contents

12.1 Recap: Does SOUP Meet the Challenges?

To begin with, SOUP is robust. It achieves high data availability with limited overhead for all users, i.e., it does not discriminate against users based on their social relations or online probability. As each user is able to select as many mirrors as needed, SOUP is able to provide robust data availability that is close to centralized solutions. Coincidentally, SOUP allows for the consideration of altruistic nodes, which will obtain high rankings and will thus be chosen as mirrors with a high probability.

At the same time, SOUP is also secure. The use of aging factors in the regular mode makes SOUP resilient to churn and node dynamics (as, e.g., provoked by DDoS attacks), as it is able to immediately react to such phenomena. If a node acts maliciously and tries to flood the system, protective dropping provides a defense. Further, Sybil users are successfully hindered to manipulate SOUP's cooperation schemes. Moreover, user data privacy is granted by appropriately encrypting the data with *Attribute Based Encryption*, such that only eligible users can access it.

Additionally, SOUP pays particular attention to mobile nodes on several layers. First, during mirror selection, SOUP considers differing node characteristics, as weak nodes will obtain low experience rankings and will be chosen as mirrors only rarely. This is of particular help to mobile users, as their devices with limited resources will not be used as mirrors frequently. Second, mobile nodes are treated with further special care, and do not have to bear the burden of DHT maintenance, which also helps the system to stabilize. As SOUP is also implemented on Android, it is ready-to-use on mobile devices as well. Users on a data plan can additionally disable mirroring if required in order to avoid high data volume transmissions on their mobiles.

Moreover, SOUP considers social relations between OSN users, as it ranks socially related nodes higher than unrelated ones. SOUP further respects social relations with protective dropping, which prefers to keep data of friends over that of strangers.

Furthermore, both the SOUP node architecture and its implementation allow for a realization of SOUP as a generic DOSN. The SOUP middleware allows multiple social applications to run on top of it concurrently. All applications exploit the same social data, which leaves the OSN user with the responsibility to maintain a single set of data, instead of having to administer one set of data for each application.

Finally, SOUP and its applications have been shown to be feasible in a deployment of a prototype. There exist no costs for the user, as only user-provided resources are exploited, and the bandwidth, storage and cryptography overheads are within reasonable bounds, which do not contra-indicate SOUP's feasibility.

12.2 The Role of User Online Time

One particular challenge in which SOUP proved to be successful is the availability of user data. The results obtained in Chapter 9 are of course linked to the assumptions made with regards to the online time of users (and thus the online time of mirrors). As a consequence, the availability of user data might turn out differently when users turn out to adopt divergent online time patterns.

However, there are three key observations to consider:

(i) Among all online time distributions chosen by SOUP and related works, the power-law distribution is the worst observed case.

(ii) SOUP—next to offering many qualitative advantages—is further already performing much better than related work, regardless of the assumed online time distribution (Chapter 9.2.7). If users turn out to be online *more* often, SOUP can exploit such changes towards lesser replication and thus communication overhead.

(iii) Even if users turn out to be *less* often, SOUP can adapt to the new situation by increasing the number of replicas in order to keep availability high.

SOUP thus provides an improvement over relevant related works, which cannot react to such changes because they lack adaptivity.

12.3 Protecting User Privacy Beyond Encryption

Although SOUP addresses its challenges well, and in particular significantly increases the security of DOSNs, some malicious exploitation might still be possible.

For instance, SOUP is currently protecting user data based on its encryption. However, accessing the data content itself is not necessarily always needed to breach user privacy. Another possibility for a miscreant is to serve as a mirror and perform tracking of accesses to the mirrored data. Leaking the resulting access patterns could compromise the accessing user's privacy.

However, the gain for the attacker and the damage to the user remain questionable. First, the attacker has to operate as a benign mirror constantly to obtain and maintain a high ranking, which is costly in terms of providing resources. Second, unless the attacker is able to control *all* replicas of a user, his view of the overall data access is still limited, as he

can only put his eyes on a portion of the data accesses. The attacker can thus not be sure whether or not this is representative for a user's overall access behavior.

Furthermore, achieving control of all replicas of a user is not easy. Initially, the attacker would need to provide a wide set of exceptionally well performing mirrors. Afterwards, it would have to get the target user to store its replicas on exactly these mirrors. This is difficult if the user already has established a stable set of mirrors, as the attacker has to rely on being randomly added to that set multiple times. This puts SOUP ahead of those related works which rely on a single replica for which access can be easily tracked by the replicating entity [36–38, 47], and also of current centralized OSNs, in which the *provider itself* can track access for *all* members of the network.

Still, leakage of access information might still be embarrassing for a user, if the leaked information only needs to entail a binary notion of *whether or not* the user accessed the data. This can be treated as an instance of *Private Information Retrieval* (PIR) [183], which deals with the problem of access pattern tracking at an untrusted server. There is a variety of established approaches to countering the adversary [184–186], of which some (e.g., *The Onion Router* (TOR) [184]) can be already set up by the user herself, if desired.

12.4 Use of Social Relations

In SOUP, social relations not only provide incentives to store data, but also play a role when selecting mirrors, deciding which data should be dropped, or filtering out malicious users and limiting their impact.

However, the trustworthiness of social relations within an OSN is questionable, as shown in earlier parts of this thesis (Chapter 6). As a consequence, recently, there have been approaches to breaking this model and studying the effect of more expressive social relations [81, 187]. In fact, friend relations in OSNs are multi-faceted and the existence of the relation itself only contributes very little to its tie strength according to a model in [81].

Even though SOUP is performing reasonably well even if a large fraction of the social relations in the OSN is compromised, exploiting such models may provide an opportunity to further improve its performance, robustness, and security. For instance, during the process of mirror selection, SOUP could prefer closely related users represented by a strong tie. The selecting node could value their experience sets more than those of mere acquaintances, which could further lower the impact of manipulated experience sets. Or, the value of the social filter β could be dynamically adjusted to the strength of the relation with each particular friend. Also, data of closer friends could be more secured from being dropped.

However, as learned from the drawbacks of related works and the design of SOUP, the tie strength must not be overvalued. Otherwise, the performance of users without strong ties (e.g., passive, but possibly still actively consuming users) might deteoriate.

12.5 SOUP and Applications for Directed Social Graphs

SOUP currently considers the social graph to be undirected. Thus, links in the graph represent a mutual friendship between users. Some OSN applications (e.g., Twitter) are however based on a directed social graph (see Chapter 2.1.1), in which one user establishes a link to another user, but not vice versa. SOUP's mirror recommendation scheme is not bound to an undirected graph. To enable a Twitter-like functionality, recommendations can be sent from followers to the followee.

However, doing so would remove the need for recommendations to originate from befriended nodes, and thus lower the bar for a Sybil attack on the scheme. The amounts of fake accounts on popular OSNs is estimated to range between 1% and 5%, and accounts usually have around 10% fake (and thus possibly Sybil) followers [49, 188]. At the same time, SOUP has shown to be resilient even if up to 50% of the social links are compromised, which would enable a use of SOUP's recommendation scheme even for Twitter-like networks.

However, SOUP should maintain its genericness and the option of exploiting social incentives, and should not be left at risk in the case users accumulate more fake followers. This could particularly happen if SOUP would use a directed graph, because in that case the attacker would have additional motivation (manipulating the recommendation scheme) to follow a large amount of honest users. As a result, the amount of fake followers might grow to a point, at which SOUP can no longer function properly (i.e., more than 50% per user).

Fortunately, SOUP has also been shown to only need a limited amount of input recommendations to function properly (see Chapter 9.2.3). Thus, applications for a directed graph should be implemented on the application layer, whereas the data availability can be taken care of as described in previous chapters. For instance, a user may run a Twitter-like application and the follower-relations can be managed by that application. At the same time, rather than relying on her followers in the Twitter clone, the SOUP data accessed by that application would still be made available with the help of the user's friends in SOUP. This way, the clear separation between the middleware and its applications is also maintained.

12.6 Use of an Extended Recommendation Scheme

The recommendations in SOUP currently measure whether or not a user's friends were able to retrieve the user's data from her mirrors. The mere availability of data is, however, only one part of the quality of service perceived by a user requesting that data from a mirror. Most obviously, the delay of the data transfer contributes to that quality of service as well.

SOUP can be extended in a way that a user's friend also reports the bandwidth available at the mirrors, which is then considered during mirror selection. Mirrors could opt-in (based on their privacy preferences) to also providing their geographical location within the DHT, such that a data request can be routed to the geographically closest high bandwidth mirror. Ultimately, this could lead to a better quality of service for users requesting data from mirrors.

12.7 The Special Case of Large Profiles

Although the storage and communication overhead is currently unproblematic when deploying SOUP to the real world, there might be difficulties if users share much larger data items or generate extremely large user profiles in the future. One option to overcome such difficulties could be the use of network coding to distribute a large profile among mirror nodes.

Network coding originally was proposed to improve the throughput utilization of a given network topology [189], but can also be used in the context of decentralized data storage [190]. Here, a file f can be split into k equally sized (f/k) *pieces*, which are in turn encoded into n fragments using an (n,k) maximum distance separable code (for details see [189]). After distributing the fragments to n nodes, it is possible to obtain the complete information from k encoded fragments.

Thus, instead of storing full replicas among mirrors, SOUP could distribute encoded parts of the profile (pieces), and then allow for reconstruction from those parts' fragments. Doing so can (i) prevent one node from being overloaded with a large profile and (ii) increase the data availability of SOUP as only k fragments need to be available to reconstruct the data of interest.

Chapter 13

Conclusion

At present, users of *Online Social Network* (OSN) are facing severe confidentiality breaches with regards to the privacy of their data, which have their roots in the centralized nature of these networks. As a result, OSN providers exhibit global control over all user data within their domain, and yet they are frequently trying to weaken their users' privacy even more. At the same time, there is little prospect of a significant turn-around towards strengthening the privacy of users and their data.

Drawing on the consequences of this, in this thesis, in consensus with a wide range of researchers, the need for a different approach to online social networking was explored. To increase user privacy and to comply with the P2P nature of OSNs, *Decentralized Online Social Network* (DOSN) constitute a promising alternative. In a DOSN, access to user data can be controlled by the users themselves such that the data is only accessible by eligible entities, and the users can be extricated from the need to maintain a wide range of OSN identities at a multitude of providers that are inspecting their data.

In a comprehensive review of the profileration of existing decentralized solutions, this thesis showed the difficulty of building a competitive DOSN. The review revealed a series of imperfections, which make a successful adaptation of any state-of-the-art DOSN unlikely. By learning from these shortcomings, a series of challenges became apparent which must be addressed by a novel, promising DOSN.

Among these, attackers orchestrating Sybil attacks against the DOSN emerged as one central challenge. To decide whether or not existing defense schemes may help with dealing with the Sybil threat, a thorough analysis of major OSN-based Sybil defenses was carried out. By focusing on the performance of each Sybil defense solution under a new scenario, which more truly reflects the evolving behavior of Sybil attackers, the analysis revealed that, unfortunately, current OSN-based Sybil defenses contain severe weaknesses. In fact, when Sybils are not grouped together in a distinct Sybil community with very few links to honest users, all of the evaluated schemes suffer in their effectiveness—some more than others.

Sybil detection approaches, even with modified designs, have a hard time reliably distinguishing Sybil nodes from honest nodes. In particular, whereas it has been shown that Sybils can obtain hundreds of attack edges in real-world OSNs, this thesis showed that all existent approaches can be circumvented by the presence of only a handful of attack edges.

Sybil tolerance schemes are application specific and, rather than being fundamentally flawed, their weaknesses are mostly in the details. Nonetheless, they too are vulnerable if Sybils deviate from their assumed behavior. As a consequence, current DOSNs cannot be protected against Sybil attacks, and a new DOSN itself must be resilient against Sybil nodes.

Motivated by the absence of a full-fledged DOSN, this thesis then presented the design, implementation, evaluation and deployment of a novel, robust and secure DOSN approach. The SELF-ORGANIZED UNIVERSE OF PEOPLE (SOUP) efficiently compensates for the lack of a centralized infrastructure by combining an efficient structured overlay — maintained by the users themselves — with a new recommendation-based approach to storing user data in a large-scale DOSN. In doing so, SOUP solves multiple problems of state-of-the-art DOSN approaches.

SOUP is *robust* in the sense that it successfully obtains the best mirror nodes for each user to achieve high data availability with little overhead, and without node discrimination. The main principle is for each user to function "with a little help from her friends" by obtaining their observations and recommendations about mirrors particularly suited for the user herself. A recursive update-based mechanism keeps all selected mirrors synchronized, and SOUP can converge to a stable state quickly.

SOUP is also *secure* in several aspects. First, user data is protected from unauthorized access and analysis as it is securely encrypted with *Attribute Based Encryption* (ABE). The encryption routines introduced by ABE allow each user to effectively exert fine-grained access control to her data. Second, SOUP tackles the severe problem of susceptibility to malicious users, in particular those executing Sybil attacks. In fact, SOUP can tolerate up to half of the social links in the OSN to be compromised by Sybils and still function properly. Besides the Sybil attack, it further copes with additional adverse situations effectively, including the high churn rates typically observed in OSNs and DDoS attacks.

In addition, SOUP can opportunistically leverage social relations and altruistically provided resources. It provides solid support towards mobile users, and enables any kind of social application to be built on top of its easy-to-use application interface. As a consequence, a user can execute multiple concurrent applications sharing the same social data, which frees her from the need to maintain and manage a copy of her data for each social application.

A comprehensive implementation and real-world deployment demonstrated that SOUP is also practicable and does not face hindering economic or technical deployability issues. SOUP remains free of charge and does not require the user to follow a complicated setup routine.

Enabling the aforementioned features distinguishes SOUP as a unique, full-fledged DOSN from existing DOSNs and their deficiencies. Finally, the discussion of opportunities to enhance SOUP has unfolded a series of possible ways to complete the last steps on the path towards a robust and secure decentralized online social network.

13.1 Thesis Impact

The author of this dissertation was the lead investigator and first author of several research papers. In particular, the work on designing, implementing and evaluating SOUP has been published in the following peer-reviewed international conference proceedings:

- **David Koll**, Jun Li, and Xiaoming Fu. SOUP: An Online Social Network By The People, For The People. In: *Proceedings of the 15th ACM/IFIP/USENIX Middleware Conference (Middleware 2014)*.
- **David Koll**, Jun Li, and Xiaoming Fu. SOUP: An Online Social Network By The People, For The People. In: *Proceedings of the 2014 ACM SIGCOMM Conference (SIGCOMM 2014), Demo Session*.

The work on analyzing OSN-based Sybil defenses has been published in the following peer-reviewed international conference proceedings:

- **David Koll**, Jun Li, Joshua Stein, and Xiaoming Fu. On The State of OSN-based Sybil Defenses. In: *Proceedings of the 13th IFIP International Conference on Networking (Networking 2014)*.
- **David Koll**, Jun Li, Joshua Stein, and Xiaoming Fu. On The Effectiveness of Sybil Defenses Based on Online Social Networks. In: *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP 2013), Poster Session*.

In advance of, but connected to this thesis, preliminary results and ideas in decentralizing online social networking have been published in the following peer-reviewed international conference proceedings:

- Florian Tegeler, **David Koll**, and Xiaoming Fu. GEMSTONE: Empowering Decentralized Social Networking with High Data Availability. In: *Proceedings of the 54th IEEE Global Communications Conference (GLOBECOM 2011)*.
- **David Koll**, and Florian Tegeler, and Xiaoming Fu. GEMSTONE: A Generic Middleware for Social Networks. In: *8th Annual International Conference on Mobile Systems, Applications and Services (MobiSys 2010), Poster Session*.

Building on the work listed above, the author has further supervised and identified topics for the following Bachelor and Master theses:

- David Kelterer. Prevention and Mitigation of Denial of Service Attacks in Enterprise Environments. Bachelor Thesis, 2013.

- Fabien Mathey. Gemstone Goes Mobile: Enabling Decentralized Online Social Networking on Mobile Devices. Bachelor Thesis, 2012.
- Kai-Stephan Jakobsen. Transitioning Social Graphs Between Different Online Social Networks. Bachelor Thesis, 2012.
- Swen Weiland. Enabling Zero-Knowledge Authentication in Decentralized Online Social Networks. Master Thesis, ongoing.

Bibliography

[1] Michael Zimmer. Mark Zuckerberg's theory of privacy. http://wapo.st/1gJOqEu (all links checked December 30th 2014), February 2014.

[2] Facebook Inc. World Cup 2014: Facebook Tops A Billion Interactions. http://newsroom.fb.com/news/2014/06/world-cup-2014-facebook-tops-a-billion-interactions/, June 2014.

[3] Twitter Inc. Insights into the #WorldCup conversation on Twitter. https://blog.twitter.com/2014/insights-into-the-worldcup-conversation-on-twitter, July 2014.

[4] Google Inc. YouTube Statistics. https://www.youtube.com/yt/press/statistics.html, July 2014.

[5] Sanvine Inc. Global Internet Phenomena Report. https://www.sandvine.com/trends/global-internet-phenomena/, May 2014.

[6] Twitter Inc. About Twitter. https://about.twitter.com/company, July 2014.

[7] Facebook Inc. Facebook Company Info. http://newsroom.fb.com/company-info/, July 2014.

[8] Alan Mislove, Hema Swetha Koppula, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Growth of the Flickr Social Network. In *Proceedings of the First Workshop on Online Social Networks (WOSN 2008)*, pages 25–30, New York, NY, USA, 2008. ACM.

[9] Twitter. Measuring tweets. https://blog.twitter.com/2010/measuring-tweets, February 2010.

[10] GlobalWebIndex.com. Twitter Now The Fastest Growing Social Platform In The World. http://bit.ly/1rJXKgA, January 2013.

[11] Alfred Hermida, Seth C. Lewis, and Rodrigo Zamith. Sourcing the Arab Spring: A Case Study of Andy Carvin's Sources on Twitter During the Tunisian and Egyptian Revolutions. *Journal of Computer-Mediated Communication*, 19(3):479–499, 2014.

[12] Habibul Haque Khondker. Role of the New Media in the Arab Spring. *Globalizations*, 8(5):675–679, 2011.

[13] Philip N. Howard, Aiden Duffy, Deen Freelon, Muzammil Hussain, Will Mari, and Marwa Mazaid. Opening Closed Regimes: What Was the Role of Social Media During the Arab Spring? 2011.

[14] Thomas Poell and Erik Borra. Twitter, YouTube, and Flickr as Platforms of Alternative Journalism: The Social Media Account of the 2010 Toronto G20 Protests. *Journalism*, 13(6):695–713, 2012.

[15] Clara Shih. *The Facebook Era: Tapping Online Social Networks to Build Better Products, Reach New Audiences, and Sell More Stuff*. Prentice Hall, 2009.

[16] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is Twitter, a Social Network or a News Media? In *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, pages 591–600, New York, NY, USA, 2010. ACM.

[17] Andreas M. Kaplan and Michael Haenlein. Users of the World, Unite! The Challenges and Opportunities of Social Media. *Business Horizons*, 53(1):59–68, 2010.

[18] Mani R. Subramani and Balaji Rajagopalan. Knowledge-sharing and Influence in Online Social Networks via Viral Marketing. *Communications of the ACM*, 46(12):300–307, 2003.

[19] BBC. Google buys YouTube for 1.65 billion dollars. bbc.in/1p92pSK, October 2006.

[20] Jim Hu. Yahoo buys photo-sharing site Flickr. cnet.co/1r3TA0x, March 2005.

[21] Balachander Krishnamurthy and Craig E. Wills. Characterizing Privacy in Online Social Networks. In *Proceedings of the First Workshop on Online Social Networks (WOSN 2008)*, pages 37–42, New York, NY, USA, 2008. ACM.

[22] Sonja Buchegger and Anwitaman Datta. A Case for P2P Infrastructure for Social Networks - Opportunities and Challenges. In *Proceedings of the 6th International Conference on Wireless On-Demand Network Systems and Services (WONS 2009)*, pages 161–168. IEEE, 2009.

[23] CNN. Facebook acquires instagram for $1 billion. cnnmon.ie/1mh3vK2, April 2012.

[24] Dominic Rushe. WhatsApp: Facebook acquires messaging service in $19bn deal. https://bitly.com/shorten/, February 2014.

[25] Catherine Dwyer. Privacy in the Age of Google and Facebook. *IEEE Technology and Society Magazine*, 30(3):58 –63, 2011.

[26] Emma Barnett. Facebook settles lawsuit with angry users. bit.ly/YYo1eZ, May 2012.

[27] Jose Van Dijck. *The Culture of Connectivity: A Critical History of Social Media*. Oxford University Press, 2013.

[28] Glenn Greenwald and Ewn MacAskill. NSA Prism program taps in to user data of Apple, Google and others. bit.ly/193WyKq, June 2013.

[29] Oliver Smith. Facebook terms and conditions: why you don't own your online life. http://www.telegraph.co.uk/technology/social-media/9780565/Facebook-terms-and-conditions-why-you-dont-own-your-online-life.html, January 2013.

[30] Alexis C. Madrigal. Why Facebook and Google's Concept of 'Real Names' Is Revolutionary. theatln.tc/1uEzeNo, May 2011.

[31] Bernhard Debatin, Jennette P. Lovejoy, Ann-Kathrin Horn, and Brittany N. Hughes. Facebook and Online Privacy: Attitudes, Behaviors, and Unintended Consequences. *Journal of Computer-Mediated Communication*, 15:83–108, 2009.

[32] Wired-Magazine. How Facebook Moved 20 Billion Instagram Photos Without You Noticing. http://www.wired.com/2014/06/facebook-instagram/, June 2014.

[33] LinkedIn. An Update on LinkedIn Member Passwords Compromised. bitly.com/Ni5aTg, June 2012.

[34] Lori Andrews. Facebook is using you. nyti.ms/1oVFxtC, February 2012.

[35] Facebook. Facebook Annual Report 2012. bit.ly/1rO7eC0, January 2013.

[36] Randy Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: An Online Social Network with User-defined Privacy. *ACM SIGCOMM Computer Communication Review*, 39(4):135–146, 2009.

[37] The Diaspora Project. https://joindiaspora.com/, 2010.

[38] Amre Shakimov, Harold Lim, Ramón Cáceres, Landon P. Cox, Kevin Li, Dongtao Liu, and Alexander Varshavsky. Vis-á-Vis: Privacy-preserving Online Social Networking via Virtual Individual Servers. In *Proceedings of the 3rd International Conference on Communication Systems and Networks (COMSNETS 2011)*, 2011.

[39] Patrick Stuedi, Iqbal Mohomed, Mahesh Balakrishnan, Z. Morley Mao, Venugopalan Ramasubramanian, Doug Terry, and Ted Wobber. Contrail: Enabling Decentralized Social Networks on Smartphones. In *Proceedings of the 12th ACM/IFIP/USENIX International Middleware Conference (Middleware 2011)*, pages 41–60. Springer, 2011.

[40] Dinh Nguyen Tran, Frank Chiang, and Jinyang Li. Friendstore: Cooperative Online Backup Using Trusted Nodes. In *Proceedings of the 1st ACM EuroSys Workshop on Social Network Systems (SNS 2008)*, pages 37–42. ACM, 2008.

[41] Sonja Buchegger, Doris Schiöberg, Le-Hung Vu, and Anwitaman Datta. PeerSoN: P2P Social Networking: Early Experiences and Insights. In *Proceedings of the 2nd ACM EuroSys Workshop on Social Network Systems (SNS 2009)*, pages 46–52. ACM, 2009.

[42] Krzysztof Rzadca, Anwitaman Datta, and Sonja Buchegger. Replica Placement in P2P Storage: Complexity and Game Theoretic Analyses. In *Proceedings of the 30th IEEE International Conference on Distributed Computing Systems (ICDCS 2010)*, pages 599–609. IEEE, 2010.

[43] Leucio Antonio Cutillo, Refik Molva, and Thorsten Strufe. Safebook: A Privacy-Preserving Online Social Network Leveraging on Real-life Trust. *IEEE Communications Magazine*, 47(12):94–101, 2009.

[44] Shirin Nilizadeh, Sonia Jahid, Prateek Mittal, Nikita Borisov, and Apu Kapadia. Cachet: A Decentralized Architecture for Privacy Preserving Social Networking with Caching. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 2012)*, pages 337–348. ACM, 2012.

[45] Alireza Mahdian, Richard Han, Qin Lv, and Shivakant Mishra. Results from a Practical Deployment of the MyZone Decentralized P2P Social Network. *CoRR*, 2013.

[46] Sebastian Biedermann, Nikolaos P. Karvelas, Stefan Katzenbeisser, Thorsten Strufe, and Andreas Peter. ProofBook: An Online Social Network Based on Proof-of-Work and Friend-Propagation. In *Theory and Practice of Computer Science (SOFSEM 2014)*, pages 114–125. Springer, 2014.

[47] Dongtao Liu, Amre Shakimov, Ramón Caceres, Alexander Varshavsky, and Landon P. Cox. Confidant: Protecting OSN Data without Locking it Up. In *Proceedings of the 12th ACM/IFIP/USENIX International Middleware Conference (Middleware 2011)*, pages 61–80. Springer, 2011.

[48] Rajesh Sharma and Anwitaman Datta. SuperNova: Super-peers Based Architecture for Decentralized Online Social Networks. In *Proceedings of the 4th IEEE International Conference on Communication Systems and Networks (COMSNETS 2012)*, pages 1–10. IEEE, 2012.

[49] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended Accounts in Retrospect: An Analysis of Twitter Spam. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement (IMC 2011)*, pages 243–258. ACM, 2011.

[50] Kurt Thomas, Damon McCoy, Chris Grier, Alek Kolcz, and Vern Paxson. Trafficking Fraudulent Accounts: The Role of the Underground Market in Twitter Spam and Abuse. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security 2013)*, pages 195–210. USENIX, 2013.

[51] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. Uncovering Social Network Sybils in the Wild. In *Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement (IMC 2011)*, pages 259–268. ACM, 2011.

[52] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The Socialbot Network: When Bots Socialize for Fame and Money. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC 2011)*, pages 93–102. ACM, 2011.

[53] John R. Douceur. The Sybil Attack. In *Peer-to-Peer Systems*, pages 251–260. Springer Berlin / Heidelberg, 2002.

[54] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. All Your Contacts are Belong to us: Automated Identity Theft Attacks on Social Networks. In *Proceedings of the 18th International Conference on World Wide Web (WWW 2009)*, pages 551–560. ACM, 2009.

[55] Rajat Bhattacharjee and Ashish Goel. Avoiding Ballot Stuffing in eBay-like Reputation Systems. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON 2005)*. ACM, 2005.

[56] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky, and Feng Xiao. SybilLimit: a Near-optimal Social Network Defense against Sybil Attacks. *IEEE/ACM Transactions on Networking*, 18(3):885–898, 2010.

[57] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the Detection of Fake Accounts in Large Scale Social Online Services. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI 2012)*, pages 197–210. USENIX, 2012.

[58] Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, and Sherman S.M. Chow. Optimal Sybil-resilient Node Admission Control. In *Proceedings of the 30th Annual IEEE International Conference on Computer Communications (INFOCOM 2011)*. IEEE, 2011.

[59] George Danezis and Prateek Mittal. SybilInfer: Detecting Sybil Nodes using Social Networks. In *Proceedings of the 16th Annual Network & Distributed System Security Symposium (NDSS 2009)*. Internet Society, 2009.

[60] Nguyen Tran, Bonan Min, Jinyang Li, and Lakshminarayanan Subramanian. Sybil-resilient Online Content Voting. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2009)*. USENIX, 2009.

[61] Alan Mislove, Ansley Post, Peter Druschel, and Krishna P. Gummadi. Ostra: Leveraging Trust to Thwart Unwanted Communication. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2008)*. USENIX, 2008.

[62] Wei Wei, Fengyuan Xu, C.C. Tan, and Qun Li. SybilDefender: A Defense Mechanism for Sybil Attacks in Large Social Networks. *IEEE Transactions on Parallel and Distributed Systems*, 24(12):2492–2502, Dec 2013.

[63] Lu Shi, Shucheng Yu, Wenjing Lou, and Y.T. Hou. SybilShield: An Agent-aided Social Network-based Sybil Defense Among Multiple Communities. In *Proceedings of the 32nd Annual IEEE International Conference on Computer Communications (INFOCOM 2013)*, pages 1034–1042. IEEE, 2013.

[64] Danesh Irani, Marco Balduzzi, Davide Balzarotti, Engin Kirda, and Calton Pu. Reverse Social Engineering Attacks in Online Social Networks. In *Proceedings of the 8th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA 2011)*, 2011.

[65] Vasumathi Sridharan, Shankar Vaibhav, and Minaxi Gupta. Twitter Games: How Successful Spammers Pick Targets. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC 2012)*, pages 389–398. ACM, 2012.

[66] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons, and Abraham D. Flaxman. SybilGuard: Defending Against Sybil Attacks via Social Networks. *IEEE/ACM Transactions on Networking*, 16(3):576–589, 2008.

[67] Stanley Wasserman and Joseph Galaskiewicz. *Advances in Social Network Analysis: Research in the Social and Behavioral Sciences*, volume 171. Sage Publications, 1994.

[68] John Arundel Barnes. *Class and Committees in a Norwegian Island Parish*. Plenum, 1954.

[69] Mark S Granovetter. The Strength of Weak Ties. *American Journal of Sociology*, pages 1360–1380, 1973.

[70] Mark Granovetter. Economic Action and Social Structure: The Problem of Embeddedness. *American Journal of Sociology*, pages 481–510, 1985.

[71] Harro Dietrich Kähler. Das Konzept des sozialen Netzwerkes: Eine Einführung in die Literatur. *Zeitschrift für Soziologie*, 4(3):283–290, 1975.

[72] Nicole B. Ellison. Social Network Sites: Definition, History, and Scholarship. *Journal of Computer-Mediated Communication*, 13(1):210–230, 2007.

[73] Eytan Bakshy, Itamar Rosenn, Cameron Marlow, and Lada Adamic. The Role of Social Networks in Information Diffusion. In *Proceedings of the 21st International Conference on World Wide Web (WWW 2012)*, pages 519–528, New York, NY, USA, 2012. ACM.

[74] Christo Wilson, Bryce Boe, Alessandra Sala, Krishna P. N. Puttaswamy, and Ben Y. Zhao. User Interactions in Social Networks and their Implications. In *Proceedings of the 4th ACM European Conference on Computer Systems (EuroSys 2009)*, pages 205–218. ACM, 2009.

[75] Alan Mislove, Massimiliano Marcon, Krishna P. Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and Analysis of Online Social Networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC 2007)*, pages 29–42. ACM, 2007.

[76] Scott Garriss, Michael Kaminsky, Michael J. Freedman, Brad Karp, David Mazières, and Haifeng Yu. Re: Reliable Email. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI 2006)*. USENIX, 2006.

[77] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A Centralized "Zero-queue" Datacenter Network. In *Proceedings of the 2014 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2014)*, pages 307–318, New York, NY, USA, 2014. ACM.

[78] Yu Wu, Chuan Wu, Bo Li, Linquan Zhang, Zongpeng Li, and Francis CM Lau. Scaling Social Media Applications into Geo-Distributed Clouds. In *Proceedings of the 31st Annual IEEE International Conference on Computer Communications (INFOCOM 2012)*, pages 684–692. IEEE, 2012.

[79] Lei Jiao, Jun Li, Wei Du, and Xiaoming Fu. Multi-objective Data Placement for Multi-cloud Socially Aware Services. In *Proceedings of the 33rd Annual IEEE International Conference on Computer Communications (INFOCOM 2014)*, pages 28–36. IEEE, April 2014.

[80] Abedelaziz Mohaisen, Aaram Yun, and Yongdae Kim. Measuring the Mixing Time of Social Graphs. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC 2010)*, pages 383–389. ACM, 2010.

[81] Eric Gilbert and Karrie Karahalios. Predicting Tie Strength with Social Media. In *Proceedings of the 27th SIGCHI Conference on Human Factors in Computing Systems (CHI 2009)*, pages 211–220. ACM, 2009.

[82] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing User Behavior in Online Social Networks. In *Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference (IMC 2009)*, pages 49–62. ACM, 2009.

[83] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The Anatomy of the Facebook Social Graph. *arXiv preprint arXiv:1111.4503*, 2011.

[84] Alessandra Sala, Haitao Zheng, Ben Y. Zhao, Sabrina Gaito, and Gian Paolo Rossi. Brief Announcement: Revisiting the Power-law Degree Distribution for Social Graph Analysis. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2010)*, pages 400–401, New York, NY, USA, 2010. ACM.

[85] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and Evolution of Online Social Networks. In *Link Mining: Models, Algorithms, and Applications*, pages 337–357. Springer, 2010.

[86] Michelle Girvan and Mark EJ Newman. Community Structure in Social and Biological Networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[87] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast Unfolding of Communities in Large Networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008.

[88] Mark E.J. Newman. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[89] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical Properties of Community Structure in Large Social and Information Networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW 2008)*, pages 695–704, New York, NY, USA, 2008. ACM.

[90] Mark E.J. Newman and Michelle Girvan. Finding and Evaluating Community Structure in Networks. *Physical Review E*, 69(2):026113, 2004.

[91] Karl Pearson. The Problem of the Random Walk. *Nature*, 72(1865):294, 1905.

[92] Jack F. Douglas. Aspects and Applications of the Random Walk. *Journal of Statistical Physics*, 79(1):497–500, 1995.

[93] László Lovász. Random Walks on Graphs: A Survey. *Combinatorics, Paul Erdos is eighty*, 2(1):1–46, 1993.

[94] Gregory F. Lawler and Vlada Limic. *Random Walk: A Modern Introduction*. Cambridge University Press, New York, NY, USA, 2010.

[95] W. Keith Hastings. Monte Carlo Sampling Methods Using Markov Chains and their Applications. *Biometrika*, 57(1):97–109, 1970.

[96] Allen E. Nix and Michael D Vose. Modeling Genetic Algorithms with Markov Chains. *Annals of Mathematics and Artificial Intelligence*, 5(1):79–88, 1992.

[97] Haifeng Yu. Sybil Defenses via Social Networks: A Tutorial and Survey. *SIGACT News*, 42(3):80–101, October 2011.

[98] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, et al. *Introduction to Algorithms*, volume 2. MIT Press Cambridge, 2001.

[99] Rudolf Ahlswede, Ning Cai, S-YR Li, and Raymond W Yeung. Network Information Flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[100] Avinash Lakshman and Prashant Malik. Cassandra: A Decentralized Structured Storage System. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[101] Joaquin Casares. Multi-datacenter Replication in Cassandra. http://www.datastax.com/dev/blog/multi-datacenter-replication, November 2012.

[102] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. *Lecture Notes in Computer Science*, 2218:329–350, 2001.

[103] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)*, pages 149–160. ACM, October 2001.

[104] Petar Maymounkov and David Mazières. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the 1st International Workshop on Peer-to-Peer Systems (IPTPS 2001)*, pages 53–65, London, UK, 2002. Springer-Verlag.

[105] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proceedings of the 2001 ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM 2001)*, pages 161–172, New York, NY, USA, 2001. ACM.

[106] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A Resilient Global-Scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41 – 53, January 2004.

[107] Raul Jimenez, Flutra Osmani, and Björn Knutsson. Sub-second Lookups on a Large-scale Kademlia-based Overlay. In *Proceedings of the 11th IEEE International Conference on Peer-to-Peer Computing (P2P 2011)*, pages 82–91. IEEE, 2011.

[108] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim, et al. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4):72–93, 2005.

[109] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC press, 2010.

[110] Gary L. Mullen and Carl Mummert. *Finite Fields and Applications*. Springer, 2007.

[111] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES-the Advanced Encryption Standard*. Springer, 2002.

[112] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In *Fast Software Encryption*, pages 1–18. Springer, 2001.

[113] Bruce Schneier. Description of a new Variable-length Key, 64-bit Block Cipher (Blowfish). In *Fast Software Encryption*, pages 191–204. Springer, 1994.

[114] Lidong Zhou, Fred B. Schneider, and Robbert Van Renesse. COCA: A Secure Distributed Online Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.

[115] W^3Techs. Usage of SSL certificate authorities for websites. http://bit.ly/1zePzg8, October 2014.

[116] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995.

[117] IETF. RFC 2828. http://tools.ietf.org/html/rfc2828.

[118] Ronald L. Rivest, Adi Shamir, and Len Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[119] Neal Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[120] Victor S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology (CRYPTO 1985)*, pages 417–426. Springer, 1986.

[121] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. In *Advances in Cryptology—CRYPTO'96*, pages 1–15. Springer, 1996.

[122] The German Federal Office for Information Security (BSI). Algorithms for Qualified Electronic Signatures. http://bit.ly/ZHxyHb, October 2013.

[123] Mozilla Foundation. Mozilla CA Certificate Maintenance Policy, Version 2.2. mzl.la/1uTyL83.

[124] Kristin Lauter. The Advantages of Elliptic Curve Cryptography for Wireless Security. *IEEE Wireless Communications*, pages 63–67, 2004.

[125] NIST. Recommended Elliptic Curves for Federal Government Use. http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf, July 1999.

[126] Michael Brown, Darrel Hankerson, Julio López, and Alfred Menezes. *Software Implementation of the NIST Elliptic Curves over Prime Fields*. Springer, 2001.

[127] Daniel J. Bernstein. The Poly1305-AES Message-authentication Code. In *Fast Software Encryption*, pages 32–49. Springer, 2005.

[128] Lily Chen. Recommendation for Key Derivation Using Pseudorandom Functions. *NIST Special Publication*, 800:108, 2008.

[129] Dan Boneh and Matt Franklin. Identity-based Encryption from the Weil Pairing. In *Advances in Cryptology (CRYPTO 2001)*, pages 213–229. Springer, 2001.

[130] Clifford Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Cryptography and Coding*, pages 360–363. Springer, 2001.

[131] Amit Sahai and Brent Waters. Fuzzy Identity-based Encryption. In *Advances in Cryptology (EUROCRYPT 2005)*, pages 457–473. Springer, 2005.

[132] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-Policy Attribute-Based Encryption. In *Proceedings of the 28th IEEE Symposium on Security and Privacy (S&P 2007)*. IEEE, 2007.

[133] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS 2006)*, pages 89–98. ACM, 2006.

[134] Jean-Paul Berrut and Lloyd N. Trefethen. Barycentric lagrange interpolation. *Siam Review*, 46(3):501–517, 2004.

[135] Michiel Hazewinkel. *Handbook of Algebra*, volume 2. Elsevier, 2000.

[136] BBC. Do we reveal too much about ourselves online? http://www.bbc.com/news/technology-22854745, June 2013.

[137] Wall Street Journal. How Private Are Your Private Facebook Messages? http://on.wsj.com/ZASqje, October 2012.

[138] Charles Riley. Facebook faces suit over private messages. http://money.cnn.com/2014/01/03/technology/facebook-privacy-lawsuit/, January 2014.

[139] Sai Teja Peddinti, Keith W. Ross, and Justin Cappos. "On the Internet, Nobody Knows You'Re a Dog": A Twitter Case Study of Anonymity in Social Networks. In *Proceedings of the 2nd ACM Conference on Online Social Networks (COSN 2014)*, pages 83–94. ACM, 2014.

[140] Emre Sarigol, David Garcia, and Frank Schweitzer. Online Privacy as a Collective Phenomenon. In *Proceedings of the 2nd ACM Conference on Online Social Networks (COSN 2014)*, pages 95–106. ACM, 2014.

[141] Kate Nibbs. What's a Facebook shadow profile, and why should you care? http://bit.ly/1p7e5ag, July 2013.

[142] Matt Dickmann. Inside//Out: Facebook Beacon. http://technomarketer.typepad.com/technomarketer/2007/11/insideout-faceb.html, November 2007.

[143] Thinking-aloud.eu. 1,6 million sets of data drawn from German social network SchülerVZ. http://bit.ly/1seTqDI, May 2010.

[144] Markus Goebel. Hacker arrested for blackmailing StudiVZ and other social networks. http://tcrn.ch/1pbpXa7, October 2009.

[145] Michael S. Bernstein, Eytan Bakshy, Moira Burke, and Brian Karrer. Quantifying the Invisible Audience in Social Networks. In *Proceedings of the 31st ACM SIGCHI Conference on Human Factors in Computing Systems (CHI 2013)*, pages 21–30, New York, NY, USA, 2013. ACM.

[146] TheGuardian.com. Mark Zuckerberg's sister learns life lesson after Facebook photo flap. http://bit.ly/1BswHbc, December 2012.

[147] Facebook Inc. Statement of Rights and Responsibilities. https://www.facebook.com/legal/terms, November 2013.

[148] Google Inc. Locations of Datacenters. http://bit.ly/YhZqAF, September 2014.

[149] Josep M. Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The Little Engine(s) That Could: Scaling Online Social Networks. *SIGCOMM Computer Communication Review*, 40(4):375–386, August 2010.

[150] Wyatt Lloyd, Michael J Freedman, Michael Kaminsky, and David G Andersen. Stronger Semantics for Low-Latency Geo-Replicated Storage. In *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2013)*, pages 313–328. USENIX, 2013.

[151] Ching-Man Au Yeung, Ilaria Liccardi, Kanghao Lu, Oshani Seneviratne, and Tim Berners-Lee. Decentralization: The Future of Online Social Networking. In *W3C Workshop on the Future of Social Networking Position Papers*, volume 2, pages 2–7, 2009.

[152] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.

[153] Adele Lu Jia, Xiaowei Chen, Xiaowen Chu, Johan Pouwelse, and Dick Epema. User Behaviors in Orivate BitTorrent Communities. *Computer Networks*, 60:34–45, 2014.

[154] John R. Douceur. Is Remote Host Availability Governed by a Universal Law? *ACM SIGMETRICS Performance Evaluation Review*, 31(3):25–29, 2003.

[155] William J. Bolosky, John R. Douceur, David Ely, and Marvin Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *ACM SIGMETRICS Performance Evaluation Review*, 28(1):34–43, 2000.

[156] László Gyarmati and Tuan Anh Trinh. Measuring User Behavior in Online Social Networks. *IEEE Network*, 24(5):26–31, 2010.

[157] Johan Pouwelse, Paweł Garbacki, Dick Epema, and Henk Sips. The BitTorrent P2P File-sharing System: Measurements and Analysis. In *Peer-to-Peer Systems IV*, pages 205–216. Springer, 2005.

[158] David Hales. From Selfish Nodes to Cooperative Networks - Emergent Link-based Incentives in Peer-to-Peer Networks. In *Proceedings of the 4th IEEE International Conference on Peer-to-Peer Computing (P2P 2004)*, pages 151–158. IEEE, Aug 2004.

[159] Diaspora. Installation Guide. https://wiki.diasporafoundation.org/Installation/Ubuntu/Saucy, January 2014.

[160] http://aws.amazon.com/ec2/.

[161] http://azure.microsoft.com/.

[162] Zack Whittaker. Snowden: 'Wannabe PRISM partner' Dropbox is 'hostile to privacy'. http://zd.net/1r84eDz, July 2014.

[163] Marc Kührer, Thomas Hupperich, Christian Rossow, and Thorsten Holz. Exit from Hell? Reducing the Impact of Amplification DDoS Attacks. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security 2014)*. USENIX, 2014.

[164] John Douceur. The Sybil Attack. In *Peer-to-Peer Systems*, pages 251–260. Springer Berlin / Heidelberg, 2002.

[165] Pan Hui, Jon Crowcroft, and Eiko Yoneki. Bubble Rap: Social-based Forwarding in Delay-tolerant Networks. *IEEE Transactions on Mobile Computing*, 10(11):1576–1589, 2011.

[166] Bimal Viswanath, Ansley Post, Krishna P. Gummadi, and Alan Mislove. An Analysis of Social Network-based Sybil Defenses. *ACM SIGCOMM Computer Communication Review*, 41(4):363–374, 2011.

[167] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *Proceedings of the 5th ACM Conference on Electronic Commerce (EC 2005)*, pages 102–111. ACM, 2004.

[168] Rajat Bhattacharjee and Ashish Goel. Avoiding Ballot Stuffing in eBay-like Reputation Systems. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Economics of Peer-to-Peer Systems (P2PECON 2005)*, 2005.

[169] Bimal Viswanath, Mainack Mondal, Allen Clement, Peter Druschel, Krishna P. Gummadi, Alan Mislove, and Ansley Post. Exploring the Design Space of Social Network-based Sybil Defense. In *Proceedings of the 4th IEEE International Conference on Communication Systems and Networks (COMSNETS 2012)*, 2012.

[170] Yazan Boshmaf, Konstantin Beznosov, and Matei Ripeanu. Graph-based Sybil Detection in Social and Information Systems. In *Proceedings of the 5th IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2013)*, pages 466–473. IEEE, 2013.

[171] Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, and Alessandro Panconesi. SoK: The Evolution of Sybil Defense via Social Networks. In *Proceedings of the 34th IEEE Symposium on Security and Privacy (S&P 2013)*, pages 382–396. IEEE, 2013.

[172] Prateek Mittal, Charalampos Papamanthou, and Dawn Song. Preserving Link Privacy in Social Network Based Systems. In *Proceedings of the 20th Annual Network and Distributed System Security Symposium (NDSS 2013)*. Internet Society, 2013.

[173] Amy N. Langville and Carl D. Meyer. Deeper Inside Pagerank. *Internet Mathematics*, 1(3):335–380, 2004.

[174] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the Evolution of User Interaction in Facebook. In *Proceedings of the 2nd ACM Workshop on Online Social Networks (WOSN 2008)*, pages 37–42. ACM, 2009.

[175] Chris Lesniewski-Laas and M. Frans Kaashoek. Whanau: a Sybil-proof Distributed Hash Table. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2010)*. USENIX, 2010.

[176] Chung Kei Wong, Mohamed Gouda, and Simon S. Lam. Secure Group Communications Using Key Graphs. *IEEE/ACM Transactions on Networking*, 8(1):16–30, 2000.

[177] Dalit Naor, Moni Naor, and Jeff Lotspiech. Revocation and Tracing Schemes for Stateless Receivers. In *Advances in Cryptology (CRYPTO 2001)*, pages 41–62. Springer, 2001.

[178] Sonia Jahid, Prateek Mittal, and Nikita Borisov. EASiER: Encryption-based Access Control in Social Networks with Efficient Revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2011)*, pages 411–415. ACM, 2011.

[179] Jing Jiang, Christo Wilson, Xiao Wang, Wenpeng Sha, Peng Huang, Yafei Dai, and Ben Y. Zhao. Understanding Latent Interactions in Online Social Networks. *ACM Transactions on the Web*, 7(4):18, 2013.

[180] Amodiovalerio Verde. Facebook Demographics. http://slidesha.re/1Ct8r9f.

[181] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. Understanding Online Social Network Usage from a Network Perspective. In *Proceedings of the 9th ACM SIGCOMM Conference on Internet Measurement (IMC 2009)*, pages 35–48. ACM, 2009.

[182] Oleksandr Bodriagov and Sonja Buchegger. Encryption for Peer-to-Peer Social Networks. In *Security and Privacy in Social Networks*, pages 47–65. Springer, 2013.

[183] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private Information Retrieval. *Journal of the ACM*, 45(6):965–981, 1998.

[184] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium (USENIX Security 2004)*. USENIX, 2004.

[185] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-Tor: Scalable Anonymous Communication Using Private Information Retrieval. In *Proceedings of the 20th USENIX Security Symposium (USENIX Security 2011)*. USENIX, 2011.

[186] Peter Williams, Radu Sion, and Bogdan Carbunar. Building Castles out of Mud: Practical Access Pattern Privacy and Correctness on Untrusted Storage. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS 2008)*. ACM, 2008.

[187] Shaojie Tang, Jing Yuan, Xufei Mao, Xiang-Yang Li, Wei Chen, and Guojun Dai. Relationship Classification in Large Scale Online Social Networks and its Impact on Information Propagation. In *Proceedings of the 30th Annual IEEE International Conference on Computer Communications (INFOCOM 2011)*. IEEE, 2011.

[188] Keith Wagstaff. 1 in 10 Twitter accounts is fake, say researchers. http://nbcnews.to/1CEk3GH, November 2013.

[189] Shuo-Yen Robert Li, Raymond W. Yeung, and Ning Cai. Linear Network Coding. *IEEE Transactions on Information Theory*, 49(2):371 –381, 2003.

[190] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin J. Wainwright, and Kannan Ramchandran. Network Coding for Distributed Storage Systems. *IEEE Transactions on Information Theory*, 56(9):4539 –4551, 2010.

Curriculum Vitae

Contact Information

David Koll
Am Steinsgraben 28
37085 Göttingen
davidkoll@gmail.com

Personal Information

Born on March 18th 1984 in Bad Hersfeld, Germany
German Citizen
Marital Status: Single

Work Experience

University of Göttingen, Göttingen, Germany

Technical Co-Coordination and Project Management, **Since 09/2014**
University of Göttingen,

- Project: *EU FP7 CleanSky ITN*

Research Assistant, **Since 03/2011**
Institute of Computer Science, Computer Networks Group, University of Göttingen

Education

University of Göttingen, Göttingen, Germany

PhD Student, **Since 03/2011**
PCS Programme in Computer Science (PCS),
Georg-August University School of Science (GAUSS),
Institute of Computer Science, Computer Networks Group

- Thesis Topic: *Towards a Robust and Secure Decentralized Online Social Network*
- Advisors: Prof. Dr. Xiaoming Fu, Prof. Dr. Dieter Hogrefe

Master of Science, **11/2010**
Institute of Computer Science, Computer Networks Group

- Thesis Topic: *Development of a Mobile Social Networking Platform Supporting Decentralized Data Storage Optimized by Social Trust*
- Advisor: Prof. Dr. Xiaoming Fu
- Thesis Grade: 1.0
- Degree Grade: 1.2, **Graduated with honors**

Bachelor of Science, **07/2007**
Institute of Computer Science in cooperation with the Faculty of Law

- Thesis Topic: *Rechtsfragen der GPL in ihrer dritten Version (Legal Issues of the GPLv3 with Regards to German Law)*
- Advisor: Prof. Dr. Gerald Spindler
- Thesis Grade: 1.1
- Degree Grade: 1.7 (Specialization: Law of Informatics)

University of Wollongong, **12/2008–08/2009**
Wollongong, Australia

Study Abroad, Faculty of Informatics

Refereed Conference Publications

Koll, D., and Li, J., and Fu, X. SOUP: An Online Social Network By The People, For The People In: *Proceedings of the 15th ACM/IFIP/USENIX Middleware Conference (Middleware'14)*, Bordeaux, France, December 8-12, 2014.

Koll, D., and Li, J., and Fu, X. SOUP: An Online Social Network By The People, For The People In: *Proceedings of the 29th ACM SIGCOMM Conference (SIGCOMM'14), Demo Session*, Chicago, USA, August 18-22, 2014.

Koll, D., and Li, J., and Stein, J., and Fu, X. On The State of OSN-based Sybil Defenses In: *Proceedings of the 13th IFIP International Conference on Networking (Networking'14)*, Trondheim, Norway, June 2–4, 2014.

Koll, D., and Li, J., and Stein, J., and Fu, X. On The Effectiveness of Sybil Defenses Based on Online Social Networks In: *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP'13), Poster Session*, Göttingen, Germany, October 7–11, 2013.

Tegeler, F., and **Koll, D.**, and Fu, X. GEMSTONE: Empowering Decentralized Social Networking with High Data Availability In: *Proceedings of the 54th IEEE Global Communications Conference (GLOBECOM'11)*, Houston, USA, December 5–9, 2011.

Koll, D., and Tegeler, F., and Fu, X. GEMSTONE: A Generic Middleware for Social Networks. In: *Proceedings of the 8th ACM Annual International Conference on Mobile Systems, Applications and Services (MobiSyS'10), Poster Session*, San Francisco, USA, June 15–18, 2010.

Other Publications

Koll, D.. Development of a Mobile Social Networking Platform Supporting Decentralized Data Storage Optimized by Social Trust. Master's Thesis, ISSN 1612-6793, University of Göttingen, Germany, November 2010, Available online.

Koll, D., and Li, J., and Fu, X. With a Little Help From my Friends: Replica Placement in Decentralized Online Social Networks. Technical Report IFI-TB-2013-01, Institute of Computer Science, University of Göttingen, Germany, January 2013.

Research Visits

Universitet Uppsala, Uppsala, Sweden **01/2014–03/2014**
- Department of Information Technology, Uppsala University, Uppsala, Sweden
- Visiting Researcher, DAAD/BMBF Scholarship
- Host: Prof. Dr. Edith Ngai

Fudan University, Shanghai, China **12/2012**
- School of Computer Science, Fudan University, Shanghai, China
- Visiting Researcher, DAAD Scholarship
- Host: Prof. Dr. Jin Zhao

University of Oregon, Eugene, Oregon, USA **03/2012–06/2012**
- Network Security Research Lab (NETSEC), University of Oregon, Eugene, USA
- Invited Visiting Researcher
- Host: Prof. Dr. Jun Li

Invited Talks

- 02/2014: "Online Social Networks By The People, For The People"
  Universitet Uppsala, Sweden
- 06/2013: "Data Availability in Decentralized Online Social Networks"
  TU Darmstadt, Germany
- 12/2012: "On the Effectiveness of Sybil Defenses based on Online Social Network"
  FU Berlin, Germany
- 12/2012: "With a Little Help From My Friends: Replica Placement in DOSNs"
  Fudan University, Shanghai, China
- 05/2012: "Decentralized Data Storage in Online Social Networks"
  University of Oregon, Eugene, USA

Honors and Awards

- 08/2014: ACM SIGCOMM 2014 Travel Grant
- 01/2014: DAAD/BMBF U4 Scholarship
- 12/2012: DAAD Travel Scholarship (DAAD PPP Exchange)
- 12/2011: IEEE GLOBECOM 2011 Student Travel Grant
- 11/2010: Graduated with honors, Master of Science

Organization Committee Memberships

- 06/2015: Web Chair at the 23rd IEEE/ACM International Symposium on Quality of Service (IWQoS 2015)
- 10/2013: Local Chair at the 21st IEEE International Conference on Network Protocols (ICNP 2013)

Referee Service

- *Journals:*
  Elsevier Computer Networks Journals
  IEEE Transactions on Vehicular Technology
  IEEE Journal on Selected Areas in Communications
  Elsevier Journal of Parallel and Distributed Computing
- *Conferences:*
  2015: IEEE INFOCOM
  2014: IEEE ICNP, IEEE IWQoS, ACM COSN
  2013: NDSS, IEEE INFOCOM, IEEE IWQoS, IEEE GLOBECOM
  2012: IEEE ICCC, IEEE ICNP

Teaching Experience

Computer Networks Group, University of Göttingen, Göttingen, Germany

Teaching Assistant **Since 03/2011**

- Advanced Computer Networks (Master level)
  - Summer 2013–present
  - Responsible for several 2 hour weekly lectures.
- Computer Networks (Bachelor level)
  - Summer 2011–present
  - Responsible for several 2 hour weekly lectures.
  - Responsible 1 hour weekly exercise where junior and senior undergraduate students learn the basics of computer networks.
  - Responsible for 2 hour exam.
- Seminar on Internet Technologies (Bachelor/Master level)
  - Winter 2011/12–present
  - Responsible for the supervision of student projects.

Thesis Supervision

Bachelor and Master Theses **Since 03/2011**

- Fabien Mathey. Gemstone Goes Mobile: Enabling Decentralized Online Social Networking on Mobile Devices. Bachelor Thesis, 2012.
- Kai-Stephan Jakobsen. Transitioning Social Graphs Between Different Online Social Networks. Bachelor Thesis, 2012.
- David Kelterer. Prevention and Mitigation of Denial of Service Attacks in Enterprise Environments. Bachelor Thesis, 2013.