# Systems for Digital Self-Sovereignty

# Systems for Digital Self-Sovereignty

## Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. dr. ir. T.H.J.J. van der Hagen,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen
op maandag 8 april 2024 om 17:30

door

**Quinten André STOKKINK**

Master of Science in Computer Science,
Technische Universiteit Delft, Nederland,
geboren te Leiderdorp, Nederland.

Dit proefschrift is goedgekeurd door de

promotor: Prof. dr. ir. D.H.J. Epema
promotor: Dr. ir. J.A. Pouwelse

Samenstelling promotiecommissie:

| | |
|---|---|
| Rector Magnificus, | voorzitter |
| Prof. dr. ir. D.H.J. Epema, | Technische Universiteit Delft, promotor |
| Dr. ir. J.A. Pouwelse, | Technische Universiteit Delft, promotor |
| *Onafhankelijke leden:* | |
| Prof. dr. M.J. van den Hoven, | Technische Universiteit Delft, The Netherlands |
| Prof. dr. A. Veneris, | University of Toronto, Canada |
| Prof. dr. D. Grossi, | University of Groningen, The Netherlands |
| Prof. dr. F. Taïani, | Université de Rennes 1, France |
| Prof. dr. ir. F.A. Kuipers, | Technische Universiteit Delft, The Netherlands |

This work was carried out in the ASCI graduate school. ASCI dissertation series number 454.

*Keywords:* anonymity, blockchain, decentralization, gossip, green, identity management, local-first, network, network latency, peer-to-peer, privacy, pseudonymity, replication, reputation, Self-Sovereign, smart contract, Sybil, Web3

*Printed by:* Gildeprint B.V., Enschede, The Netherlands

*Cover:* Quinten Stokkink and DALL-E 2. The cover is a Penrose tiling of passport photos, as interpreted by Artificial Intelligence.

*Style:* TU Delft House Style, with modifications by Moritz Beller
`https://github.com/Inventitech/phd-thesis-template`

The author set this thesis in LaTeX using the Libertinus and Inconsolata fonts.

ISBN 978-94-6366-839-2

An electronic version of this dissertation is available at
`http://repository.tudelft.nl/`.

*Τὸ ἐν ταῖς ἐξηγήσεσι μὴ δυσχεραντικόν*
*καὶ τὸ ἰδεῖν ἄνθρωπον σαφῶς ἐλάχιστον τῶν ἑαυτοῦ καλῶν ἡγούμενον τὴν ἐμπειρίαν*
*καὶ τὴν ἐντρέχειαν τὴν περὶ τὸ παραδιδόναι τὰ θεωρήματα.*

Marcus Aurelius, Meditations, describing his learnings from Apollonius[1]

[1]Transcription and translation from The Meditations of the Emperor Marcus Antoninus by Arthur Spenser Loat Farquharson (1944): *"Not to be censorious in exposition; and to see a man who plainly considered technical knowledge and ease in communicating general truths as the least of his good gifts."*

# Contents

# Summary

The digital world is evolving toward representing—and serving the interconnection of—natural persons. Instead of depending on the intrastructure of Big Tech companies and governments, users can cooperate and use their hardware to form public infrastructure. Instead of existing by virtue of a reference in some institution's database, users can interact based on a digital representation of their own choosing. It is no longer sufficient to depend on users to act out of system-imposed altruism. A new digital world is emerging that aims to provide systems that respect the rights of users to control their own digital representation. The complete control over one's own representation and all the data that belongs to it is what we know as *Self-Sovereignty*.

Solutions for digital Self-Sovereignty are wildly sought after, though their solution space remains woefully underexplored. Numerous global entities, e.g., the European Union, have stated their support for Self-Sovereign systems. However, many old problems of peer-to-peer systems that have gone ignored for decennia resurge with the need for Self-Sovereignty. For example, interconnections in peer-to-peer networks are vulnerable to attacks using fake identities and attackers can manipulate peers by depriving them of data. As most deployed peer-to-peer solutions have very little incentive for disruption by attackers, we have seen very little attacks. However, cryptocurrencies have shown that these attacks do surface when there is sizable monetary gain for attackers. In order to secure our future digital society, we must define and study these systems for Self-Sovereignty.

In this thesis we take the first steps toward defining the systems that can power a Self-Sovereign "Web3" ecosystem. In particular, we explore systems that apply Self-Sovereignty for identity, for public infrastructure, and for the execution of shared code. We describe four prototype mechanisms to form a guide for future work and to derive their general properties. Each mechanism is evaluated as realistically as possible. Thereby, this thesis mostly fulfills an exploratory role to guide the further evolution of our digital world.

In Chapter 1 we describe the application of Self-Sovereignty and its considerations for identity systems, public infrastructure maintenance, and for execution of shared code. We discuss the evolution and historical approaches that have led to the need and to the creation of these systems. We then formulate the main research question of this thesis and its four derivative research questions. Furthermore, we describe our research and engineering methodology and we outline the contributions and structure of this thesis.

In Chapter 2 we present a truly Self-Sovereign Identity system. We define the three desirable Self-Sovereign Identity system properties of Self-Sovereignty, Credibility, and Network-level Anonymity. Furthermore, we give the necessary functional requirements for such a system to operate without any third party servers while guaranteeing the privacy of its users. Our evaluation shows that the overhead of using our *TrustChainIDentity (TCID)* solution is not prohibitive for practical use and that existing work is overly focused on cryptographic research. We conclude that more research is needed for network privacy and peer interconnectivity.

In Chapter 3 we define our mechanism *SybilSys*, which achieves peer interconnectivity in the face of fake identities. By creating a surplus of identities from a single device, called Sybils, attackers can cheaply and easily interfere with connections between peers. Secondarily, there exists a circular dependency between trusting other peers and connecting to other peers. In order to filter out Sybils from the list of connections, our mechanism uses network latency. We show that our basic mechanism avoids Sybils and we further define an enhanced version of our mechanism that is resilient against attackers that attempt to subvert our strategy. We show that our enhanced mechanism can find honest peers even in networks that consist of 99% Sybils that actively interfere with latency measurements.

In Chapter 4 the *Timely Sharing with Reputation Prototype (TSRP)* mechanism is discussed, a mechanism for carrier selection to aid public infrastructure maintenance. In order for public infrastructure to be maintained, data must be carried (i.e., stored and shared) between peers. However, in order for communication to be efficient, as little data as possible must be shared. At the same time, to uphold security guarantees, data should be—at the very least—shared with some given number of peers. TSRP is such a mechanism that selects random carriers to share data with based on their reputation. Through our evaluation we verify the basic behavior of TSRP and we show that TSRP does not necessarily lead to many more copies of data when compared to approaches that depend on trust.

In Chapter 5 we describe our *Green Smart Contracts (GSC)* paradigm to execute shared code. In contrast to traditional blockchains that execute their code on all nodes, we show that a local-first approach is more efficient. A local-first approach consists of first optimistically executing code and then later merging this execution with the execution of peers. In case of a conflict between the executions of peers, a consensus mechanism has some peers roll back (i.e., undo part of their previous execution) and follow a different execution path. At the same time, we argue that a local-first approach is not necessarily useful for all applications. Specifially, we argue that the GSC paradigm is useful for liability-based applications. Nevertheless, our experiments show that a local-first approach can execute a real-world program without collapsing the execution paradigm.

In Chapter 6 we give a summary of our findings and we provide the overarching conclusions of this thesis. We end this chapter with an overview of the future directions that remain to be explored.

# Samenvatting

De digitale wereld ontwikkelt zich ter ondersteuning van de representatie en interconnectie van natuurlijke personen. In plaats van afhankelijk te zijn van de infrastructuur van “Big Tech” bedrijven en overheden, kunnen gebruikers samenwerken en hun apparaten inzetten om publieke infrastructuur te realiseren. In plaats van bestaan bij wille van een referentie in de data opslag van een of andere institutie, kunnen gebruikers interacteren op basis van een digitale representatie van hun eigen keuze. Het is niet langer voldoende om te bouwen op gebruikers die handelen uit systeemgedreven altruïsme. Er is een nieuwe digitale wereld aan het opkomen met als doel systemen voort te brengen waarin de rechten van gebruikers, om hun eigen digitale representatie volledig in handen te hebben, gerespecteerd worden. Deze complete controle over eenieder’s eigen representatie en alle data die daartoe behoort, is wat we kennen als Zelf-Soevereiniteit.

Er wordt naarstig gezocht naar oplossingen voor digitale Zelf-Soevereiniteit, ook al is de oplossingsrichting gebrekkig onderzocht. Verschillende globale spelers, zoals de Europese Unie, hebben hun toewijding aan Zelf-Soevereine systemen kenbaar gemaakt. Daarentegen doen vele oude problemen van systemen voor egalitaire interactie, die vele decennia genegeerd zijn, hun wederintrede met deze roep om Zelf-Soevereiniteit. De interconnecties in netwerken van gelijke participanten zijn bijvoorbeeld kwetsbaar voor aanvallen door nepidentiteiten en aanvallers kunnen netwerk participanten manipuleren door ze te data te onthouden. Gezien de meeste uitgerolde oplossingen voor egalitaire interactie erg weinig stimulus creëren voor disruptie door aanvallers, hebben er over de jaren vrij weinig aanvallen plaats gevonden. Echter hebben cryptografische betaalmiddelen aangetoond dat aanvallen wel degelijk plaats vinden wanneer er een substantiële monetaire beloning tegenover staat voor aanvallers. Ten behoeve van het veiligstellen van onze toekomstige digitale samenleving, moeten de systemen voor Zelf-Soevereiniteit gedefinieerd en bestudeerd worden.

In dit proefschrift nemen we de eerste stappen richting het definiëren van systemen die een Zelf-Soeverein “Web3” ecosysteem kunnen aandrijven. In het bijzonder onderzoeken we systemen die Zelf-Soevereiniteit toepassen voor identiteit, voor publieke intrastructuur en voor het uitvoeren van gedeelde programmacode. Om een leidraad te vormen voor toekomstig onderzoek en om hun algemene eigenschappen af te leiden, beschrijven we in dit proefschrift vier prototypes voor de mechanismen die onderdeel uitmaken van dit soort systemen. Ieder mechanisme is op een zo realistisch mogelijke manier geëvalueerd. Daarmee vervult dit proefschrift voornamelijk een verkennende rol om de verdere evolutie van onze digitale wereld in goede banen te leiden.

In hoofdstuk 1 beschrijven we de toepassing en overwegingen van Zelf-Soevereiniteit voor identiteitssystemen, het onderhoud van publieke infrastructuur en voor het uitvoeren van gedeelde programmacode. We bespreken de evolutie en de geschiedenis van benaderingen die voor deze systemen hebben geleid tot zowel hun behoeftestelling als hun ontwikkeling. Daarna formuleren we de hoofdonderzoeksvraag van dit proefschrift en de

vier daarvan-afgeleidde onderzoeksvragen. Verder beschrijven we onze onderzoeks- en ontwikkelmethodologie en geven we in hoofdlijnen de contributies en de verdere structurering van dit proefschrift.

In hoofdstuk 2 presenteren we een waarlijk Zelf-Soeverein Identiteitssysteem. We definiëren de drie eigenschappen van Zelf-Soevereine Identiteitssystemen als zijnde “Zelf-Soevereiniteit”, “Geloofwaardigheid” en “Anonimiteit op Netwerkniveau”. Verder geven voor dat soort systemen we de functionele vereisten om te opereren zonder servers onder het beheer van derde partijen én om privacy van de gebruikers te waarborgen. Onze evaluatie toont aan dat de extra belasting op machines door het gebruik van onze *Trust-ChainIDentity (TCID)* oplossing geen practisch bezwaar vormt en dat voorgaand werk te veel toegespitst is op cryptografisch onderzoek. We concluderen dat er meer onderzoek nodig is naar netwerkprivacy en de interconnecties van netwerk participanten.

In hoofdstuk 3 definiëren we ons *SybilSys* mechanisme, dat interconnectiviteit tussen gelijke netwerk participanten bewerkstelligt ondanks de aanwezigheid van nepidentiteiten. Door vanuit een enkel apparaat een overdaad aan identiteiten te creëren, ook wel Sybils genoemd, kunnen aanvallers goedkoop en makkelijk interfereren met connecties tussen netwerk participanten. Daarnaast is er een circulaire afhankelijkheid tussen het opbouwen van vertrouwen en het maken van verbindingen tussen netwerk participanten. Om Sybils uit de connecties te filteren gebruikt ons mechanisme netwerk latentie. We tonen aan dat onse basismechanisme Sybils vermijdt en we definiëren een versterkte versie van ons mechanisme dat weerbaar is tegen aanvallers die onder ons basismechanisme proberen uit te komen. We tonen aan dat ons versterkte mechanisme echte participanten vindt zelfs als een netwerk voor 99% bestaat uit Sybils die actief latentiemetingen proberen te ondermijnen.

In hoofdstuk 4 wordt het *Timely Sharing with Reputation Prototype (TSRP)* mechanisme besproken: een mechanisme voor het selecteren van dragers om publieke infrastructuur te onderhouden. Om publieke infrastructuur te onderhouden moet data gedragen - dat wil zeggen opgeslagen en uitgewisseld - worden tussen netwerk participanten. Voor efficiënte communicatie moet echter zo min mogelijk data uitgewisseld worden. Tegelijkertijd moet data - voor behoud van veiligheid - ten minste met een gegeven hoeveelheid gekozen participanten uit het volledige netwerk gedeeld worden. TSRP is zo’n mechanisme dat willekeurig gekozen dragers selecteert, op basis van reputatie, om data mee uit te wisselen. Door middel van onze evaluatie verifiëren we de basale functionaliteit van TSRP en tonen we aan dat het gebruik van TSRP niet per se leidt tot meer kopieën van data in relatie tot benaderingen die gebaseerd zijn op vertrouwen.

In hoofdstuk 5 beschrijven we ons *Green Smart Contracts (GSC)* paradigma om programmacode uit te voeren. In contrast tot traditionele blockchains die hun code gelijktijdig uitvoeren bij alle participanten van een netwerk, tonen we aan dat het efficiënter is om eerst lokaal “local-first” in elke participant code uit te voeren. Een local-first aanpak bestaat uit de optimistische executie van programmacode, gevolgd door het samenvoegen van deze executie met de executies van andere netwerk participanten. In het geval van een conflict tussen executies van participanten, zorgt een consensus mechanisme ervoor dat sommige participanten een rollback uitvoeren (d.w.z. een gedeelte van hun voorgaande executie terugdraaien) en dat ze een ander executiepad volgen. Tegelijkertijd beargumenteren we dat een local-first aanpak niet per se in het algemeen toepasbaar is voor alle appli-

caties. Echter tonen onze experimenten aan dat een local-first aanpak echte programma's uit kan voeren zonder dat het executie paradigma vast loopt.

In hoofdstuk 6 geven we een samenvatting van onze bevindingen en de conclusies die deze thesis overstijgen. We eindigen dit hoofdstuk met een overzicht van de onderzoeks-richtingen die nog mogelijk onderzocht kunnen worden.

# Acknowledgments

I could have started this thesis with just about any quote from the movie Rocky (1976). That would've certainly been very descriptive of the PhD process and very inspiring for new PhD students. Instead, I gave you a quote from Marcus Aurelius, one of the five people that went down in history as the so-called "Good Emperors of Rome". Beyond its superficial interpretation, the intention of this quote is to show that even one of the—objectively—greatest human beings to ever walk the earth is fundamentally formed by, and supported by, their interactions with others. In recognition of this fact, I use first-person plural pronouns throughout this thesis except for these acknowledgements. Here, I personally acknowledge all of the other people that have helped shape this plurality.

The first people that I thank are those that have been involved the closest with this thesis, my two promotors: Dick Epema and Johan Pouwelse. First, Dick: you directly inspired my epigraph; the description of Apollonius fits you very well. You have taught me how to be clear in my textual exposition and I fondly remember your stories—or complaints rather—about scientific obfuscation using mathematical constructs. You were the first one to tell me that you don't understand my text and that has been the been the single most important thing I needed to hear to improve my writing. I hope to one day match your abililties to disambiguate and to be succinct. Second, Johan: you have a sixth sense for scientific, and societal, relevance and you are a seemingly-endless source of inspiration. Thanks to your efforts, I have had the opportunity to visit many companies and organizations that I never would have expected to ever see the inner workings of. You have taught me the value of stepping away from the comfort zone of academia and connecting with industry and government. Most importantly, you have shown me that there is always fun to be had when doing science. Perhaps, in the future, I will also be able to see the value and opportunities that lie within people and their ideas with the same ease as you.

I extend my gratitude toward the people that have (almost) written papers with me. My special thanks go to Alexander Stannat, Can Umut Ileri, Dick, Georgy Ishmaev, and Johan, for seeing my submissions through to their publication. Especially Can: you have managed to stay in high spirits for years even when faced with several barely-rejected papers (one paper even with four "weak accept" reviews). However, I would also like to thank Aaron Yi Ding, Bulat Nasrulin, Jan Rellermeyer, and Martijn de Vos, for working with me and almost becoming my co-authors. In particular, Jan: even after being involved with five submissions (including chapters 3 and 4 of this thesis), you sadly left the TU Delft just before my works started to get accepted into the body of science.

During my implementation work, I have closely worked with some very talented people from industry and government. From the Rijksdienst voor Identiteitsgegeven (RvIG), André de Kok, Fons Knopjes, Frans Rijkers, Jasper Mutsaers, and Pepijn Terra: I learned a lot about the inner workings of our Dutch government from you. André, thank you for taking the nerds from Delft by the hand through the wild world of government. From CMS Legal Services, Simon Sanders: thank you for humoring me for hours on end to determine the legal status of our pilot solution and I will never forget your discussion with Pepijn on who has the most verbose connotations in their law book. From IDEMIA, Emma Smal, Emmanuel Bernard, Jan Patrick Sunglao, Joost van Prooijen, Jouri de Vos, Laurent Mercier, Michiel de Wijkerslooth, and Stef Haartman: thank you for inviting me over time and time again, both in Haarlem and Paris, in order to get the identity pilot operational. Jan Patrick, I had a lot of fun both times that you came over to the TU Delft and your mango pieces will not be forgotten! From the Dutch Chamber of Commerce (KVK), Joost Fleuren and Said Akdim: thank you for your efforts (also with our student Tim Speelman) in integrating our solution into your environment. Finally, I would like to thank everyone I worked with during my project for the European Union Agency for Cybersecurity (ENISA).

For their engaging questions, I also thank the many students that personally sought me out for guidance during their course or thesis work: Alexander Stannat, Jasmin Huber, Rowdy Chotkan, Tim Speelman, and Valentin Gérard. Likewise, I thank the students I guided during the Blockchain Engineering MSc. course, the Advanced Blockchain Engineering PhD course, and the Bachelor Seminar. For the latter, to Dirk van Bokkem, Rico Hageman, Gijs Koning, Luat Nguyen, and Naqib Zarin: I'm glad I told you to upload your paper to arXiv even though you didn't want to publish your work.

A big part of PhD life is the coffee break and there is no better way to spend it than engaging in some systemic hyperboly and I would like to thank everyone who stuck around to hear it. Here's to my (previous and current) roommates that have suffered through the bulk of my coffee talk: Alexander Kozlovsky, Andrei Andreev, Bart Gerritsen, Bulat, Can, Egbert Bouman, Gaomei Shi, Georgy, Leonard Franken, Martijn, Sandip Pandey, and Vadim Bulavintsev. In particular, Egbert: your presence always managed to synergize with my hyperboly and elevate it to a higher plane. I also enjoyed learning everyone's mother tongue. For their teachings, I would also like to shoutout Alexander, Georgy, and Vadim for Russian, Amirmasoud Ghiassi for Farsi, Can for Turkish, Mei for Mandarin Chinese, Sandip for Nepalese. Mei, I will never forget my Google Image Search to show you that Sesame Street has a Chinese version (regrettably, the random character I chose to search for happened to be Big Bird, a.k.a. ”Da Niao”).

For allowing me to flex my Artificial Intelligence skills, my thanks go out to Jole “DJ S3RL” Hughes. I had a lot of fun bringing my hobby of abusing generative adversarial networks for video generation into practice, before DALL-E even made image generation cool. It was borderline-criminal by YouTube to age-restrict the resulting video, 291 000 views is way too little for this masterpiece.

My thanks go out to the friends I spent my free time with, feeding me beers and keeping me grounded during my PhD. Especially, I would like to thank my close friends Frank van der Hulst and Matthijs Zijlstra. Frank, I hope you enjoy fatherhood and the married life. Matthijs, I hope you can also find someone to settle down with. Furthermore, my thanks go out to the “Hoogvliet 430” people I have visited Japanese all-you-can-eats with.

I would like to end by thanking my family, André, Brenda, Eric, Marco, and Nadine: who made sure to remind me to get a “real” job and a girlfriend but nonetheless supported me and my decision to stay on the PhD track.

*Quinten*

# 1 Introduction

Digital Self-Sovereignty is the idea of being in complete control of all aspects of one's representation in the digital world. The idea of Self-Sovereignty aligns with a counterculture that rejects the data collection practices of Big Tech companies. Instead of individuals getting access to their own personally identifiable information from companies, Self-Sovereignty would allow individuals to grant companies permission to access parts of their information. The aim is to completely reverse the hierarchy in the current interaction model of individuals and the companies they interact with. However, beyond the aims of the counterculture, the idea of Self-Sovereignty can be applied to all online interactions. This thesis explores the application of Self-Sovereignty not just for interactions with companies but also those with institutions like governments and other individuals.

The first inception of digital Self-Sovereignty came for the domain of digital identities in the form of Self-Sovereign Identity, coined in the essay "The Path to Self-Sovereign Identity" by C. Allen in 2016 [4]. In his essay, Allen describes the evolution of identity management systems through three phases and he attempts to predict the next phase of identity management. The first phase of identity management is "centralized identity", where one party has full control over the identity information of any individual. The second phase is "federated identity" and sees multiple third parties in charge of identity information. Finally, in the third phase individuals can carry their own credentials between third parties, dubbed "user-centric identity". Allen predicts the next phase of identity to be that an individual's identity data does not originate from some third party. Instead, the individual should be the origin of their own identity data, which Allen calls Self-Sovereign Identity.

The modern interpretation of Self-Sovereignty emerged with the concept of Blockchain. Essentially, Self-Sovereignty is a materialization of the interaction model between users of Blockchain systems. All users are equal, i.e., they are peers. Users work together in a peer-to-peer system to establish a public infrastructure (which may consist of a chain of data that forms a blockchain). As all users are equal, no peer holds authority over any other peer and no trust relationships exist between peers when they become part of a (Blockchain) system. All in all, every peer in a network is Self-Sovereign and works together to support decentralized public infrastructure (albeit on their own terms). For this reason, the model of Self-Sovereignty is also sometimes called "trustless" and "open".

The Self-Sovereign model allows for governance through shared logic, consisting of program code that is shared by all peers. Using shared logic, all peers can obtain (i.e., calculate) the same common truth from the data that is available in their public infrastructure. Establishing a common truth between peers is essential to govern public infrastructure. For instance, in a cryptocurrency system, if users can have an account balance that is not a common truth, users can create counterfeit currency without any other user being able to establish whether this currency is counterfeit or not (i.e., whether it is *true* currency). The key concept of shared code is that a system can impose rules upon its users (e.g., not allowing users to spend counterfeit money). However, such a system is not governed by a single authority, but rather the shared interest of its users.

Within this thesis we discuss the materializations of systems that are capable of providing Self-Sovereignty for the domains of identity, public infrastructure, and shared program code. We design and analyze novel mechanisms that address problems within these three domains. In particular, we analyze a mechanism to avoid connecting to a plethora of fake identities and a mechanism to identify users in a Self-Sovereign fashion, a mechanism to maintain shared public infrastructure, and a mechanism to execute shared program code. We now discuss the history of these three domains and their facets (Section 1.1, Section 1.2, and Section 1.3), our research questions (Section 1.4), our research context (Section 1.5), our research methodology (Section 1.6), and the structure of this thesis and its scientific contributions (Section 1.7).

## 1.1 Self-Sovereign Digital Identity

Digital identities are necessarily the starting point for all digital peer-to-peer interactions. This necessity is grounded in the need for identifiers in order to route messages between peers, for basic communication. For example, to route messages through the Internet, an IP address may be required. Issues arise when deciding what entity, or entities, should be provided a digital identity and who governs the assignment of these identities. Firstly, one can derive identity from almost any (physical or digital) entity and what identity is—or should be—dominant depends on the use case. For example, identity may be coupled to a machine as its associated entity (in the form of an IP address) but also to the natural person that holds a device. Secondly, peers can define their own identity or have it assigned to them. A peer may create its own cryptographic key or simply exist as a database entry in the server of some company. For instance, when buying products in a web store, identities may only need to be assigned by the store and subsequently fetched from a database, when users associate themselves with their identity using a username and password.

A Self-Sovereign approach tasks users with creating their own digital identity, fundamentally changing the established model of digital interactions and the business logic that belongs to them. No longer do users of a third party's service request access to exist in a database. Instead, both a user and a third party operate as peers that request each other to prove that they own certain identity information. For example, a bar may ask a potential customer to digitally prove that they are over 18 years old instead of associating this customer with a registered identity that has a known age. Not only does this completely change the business logic for relationships between businesses and consumers, it also affects the digital communication protocol. When communicating, users must now not only associate themselves with their identity data, they must also present it.

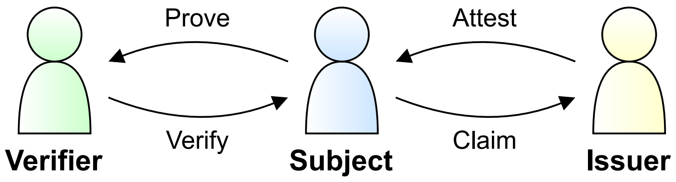


Figure 1.1: The three actors in Self-Sovereign Identity systems and their four interactions that result from creating a claim ("Claim" and "Attest") and subsequently verifying this claim ("Verify" and "Prove").

### 1.1.1 History

Since the dawn of civilization, governments have sought to identify those with special privilege. In ancient times, identity data was used to opt-in to certain services, or make use of certain rights. As early as 4000 B.C., the Babylonians conducted censuses to determine the necessary food that had to be procured to feed the population [225]. However, by the end of the 1800's most countries would do away with the opt-in nature of censuses and adopt a population registry. It would take no more than one century, in the 1900's, for this information to be misused for racial persecution based on birthplace and ancestral ethnicity [225]. To this day, we still use such a population registry.

Recent years have seen many rules imposed for governance of data derived from natural persons (e.g., the European Union's GDPR directive). The downsides of storing identity data of human beings in large databases have shown themselves and this practice is becoming disincentivized under pressure of regulation. The commonality that we observe, between all of these regulations, is that users of technology should be in full control of their own identity data, regardless of where it is stored. Though it is certainly possible for third parties (governments and industry alike) to facilitate access to the identity data they store, a user having full control over their own identity data aligns more naturally with a peer-to-peer Self-Sovereign model. Therefore, we see a Self-Sovereign model as the next necessary step to retire the flawed idea of centrally-governed registries and databases.

### 1.1.2 Actors in Self-Sovereign Identity systems

Self-Sovereign Identity models often distinguish three actor roles, shown in Figure 1.1: a subject, an issuer, and a verifier. Because Self-Sovereign Identity systems (should) operate as peer-to-peer solutions, these roles are driven by business logic. The subject is an actor that wishes to enage in the business logic of a verifier. As an example, we discuss the fictional case of the subject attempting to buy beer from a verifier. In our example, the verifier would be required by law to assert that the subject's age is over 18. However, subjects can claim just about anything and the verifier would still not be sure if this claim is true. For example, the subject could be 16 and digitally claim to be 50 years old. In other words, a verifier cannot legally trust subjects without further proof of their claims.

In order to prove that the claims of a subject are true, issuers are used for transitive trust through attestations. Whereas a random subject may be untrusted to a verifier, a claim attested to by an issuer that is an authority for a certain type of data may be sufficient. For example, a government is authorative for a subject's age data and this subject may legally buy alcohol if they present their digital claim that they are 50 years old and present an attestation to this fact created by their government. In short, in this Self-Sovereign model, an issuer is a party that attests to claims of subjects and subjects use the combination of claims and attestations to prove to verifiers that their claims are truths. This transitive truth, or transitive trust, is essentially the same idea as the web of trust as it appeared in PGP [249], where users would sign each other's keys to show trustworthiness.

### 1.1.3 Identity wallets for digital representation

The management of a subject's claims and attestations is performed by what is known as an *identity wallet*. The goal of an identity wallet is to provide a subject with control over all aspects of the disclosure of their identity information. Here "control" entails the storage and collection of new claims and attestations, updates or removal of claims, and presentation of claims and attestations to verifiers. Coming back to the essay of Allen [4], the goal of this wallet construction is to provide its users (subjects) with the ten properties of *existence* (the ability to create an identity without contacting a third-party), *control*, *access*, *transparency*, *persistence*, *portability*, *interoperability*, *consent*, *minimalization* (the ability to share only the bare minimum data needed for identification), and *protection*. The requirements of *existence*, *control*, *access*, *transparency*, *persistence*, and *consent*, are almost trivially provided by an identity wallet construction using claims and attestations [212]. However, the remaining four properties of *portability*, *interoperability*, *minimalization*, and *protection*, are more intertwined with cryptographic primitives and we summarily give examples of their intricacies.

In this thesis, we aim to implement an identity wallet sytem, and we now highlight four intricacies of implementing Allen's properties. Firstly, to provide protection, apart from following well-established security standards for session management, persistent identities must also use some form of cryptographic key rotation (i.e., the periodic change of keys). In turn, the key rotation may not be centrally governed, which would violate the independent existence property. Secondly, "selective disclosure" schemes are used to provide minimalization. These schemes allow users to construct claims that show either only part of, or even just a property of, their identity information. However, thirdly, many complex standards exist for claim disclosure which are not interoperable on the protocol level like Zero-Knowledge Proof constructions and signature schemes. This implies that identity wallets need to support a wide array of protocols and have protocol negotiation capabilities. Lastly, the copying of identity information to provide portability leads to indistinguishable duplicates of identities. However, many (legacy) systems depend on the uniqueness of identities to associate them with their stored information. We derive a comprehensive mapping of Allen's properties to technical requirements for identity wallet systems in Chapter 2.

At the time of writing Chapter 2 of this thesis, the year 2020, no solutions existed that satisfied all of Allen's properties. In retrospect, the aim of Chapter 2 may no longer seem relevant in light of new technology that does implement these properties. However, we believe that–with our groundbreaking implementation—our work was instrumental in advancing this technology and that it still forms a solid foundation for future work.

### 1.1.4 Attacks on peer-to-peer identity technology

Self-Sovereign models that use peer-to-peer communication are open to the same attacks as other peer-to-peer technology. However, to a certain extent, Self-Sovereignty is also a solution for some of those attacks. For instance, Self-Sovereign Identity can be used as a solution for an application to avoid communicating with "Sybil" identities. An attack using Sybil identities consists of the creation of a surplus of fake identities and it is based on the ability of attackers to cheaply create these Sybils. By requiring Sybils to show attestations that are difficult to acquire (e.g., an attestation from a national government that attests to an identity being unique to a single human being), their existence is functionally worthless to attackers on the application layer. However, focusing on the network layer, Self-Sovereign Identity can obviously not be used to determine the identity of peers before communicating with them. Therefore, through their sheer existence, Sybils can be used to block communication between peers by forcing them into never-ending identity checks of these otherwise-worthless identities. In Chapter 3, we give special attention to avoiding connections to Sybils to enable Self-Sovereign Identity as a solution.

The privacy guarantees of communication substrates like the Internet do not align with those that Self-Sovereign Identity aims to provide. In fact, most substrates optimize their routing for the intent and content of user traffic [93]. For instance, it has been proposed to make 6G "semantic and goal-oriented" [218]. However, this exposure of intent and goals is orthogonal to Allen's property of *protection*, which includes privacy. For example, showing intent to share highly sensitive information with a bank makes subjects an obvious target for attackers. Therefore, Self-Sovereign Identity must protect users from their communication substrate, to avoid exposing their identity data. We describe our solution to protect users from their communication substrates in Chapter 2.

## 1.2 Self-Sovereign Public Infrastructure: Web3

Web3 is the name assigned to technology that enables peer-to-peer representation and interaction without the need for trusted third-parties to enable either of these actions. In contrast, in Web2 ecosystems users request access to the resources of centrally governed institutions (e.g., the cloud infrastructure of "Big Tech", web stores and auction sites, and the web portals of governments). Some examples of Web3 technology are Bitcoin [164], Ethereum [240], and the InterPlanetary File System [25]. Most Web3 functionality revolves around currency, as payment is one of two known ways to enable peer-to-peer incentives, the alternative being reciprocity [81]. For example, in cryptocurrencies users pay miners to transact with others and in systems that offer code execution users pay to interact with the code. At this point in time, Web3 technology is critically bound to cryptocurrencies for its peer-to-peer incentives.

What sets Web3 technology apart from existing peer-to-peer technology, like BitTorrent, is its trustless nature. For example, in BitTorrent peers use a Distributed Hash Table (DHT) to determine which peers must store certain data. There are two things that disqualify BitTorrent as Web3 technology: a data structure that depends on users' honesty for its maintenance and a lack of incentive for users to store its corresponding data. If peers are dishonest, they can hijack the DHT's routing using an Index Poisoning attack [142] that routes all users to their (potentially malicious) data. Secondly, peers are not rewarded by the DHT to store data and can easily read out a DHT without contributing any storage

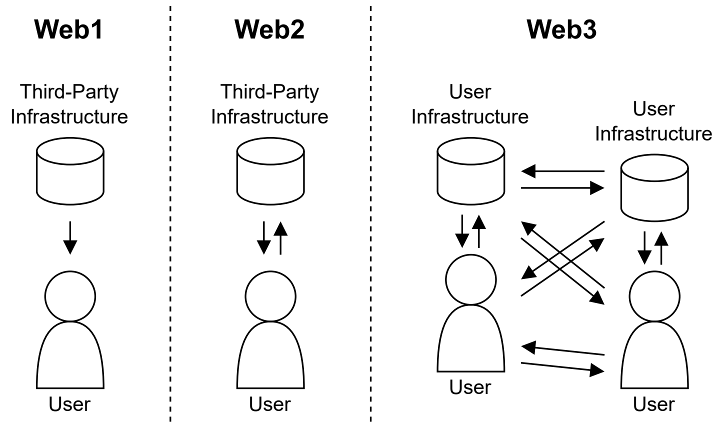


Figure 1.2: The three versions of Web interaction models: from hypertext data being served to users (Web1), to users uploading and downloading data (Web2), and users actively becoming part of public infrastructure (Web3). Note that the infrastructure may consist of multiple servers.

themselves. Web3 technology addresses these issues to some extent. However, even Web3 technology typically cannot deal with a majority of malicious nodes (e.g., Bitcoin famously claimed to resist up to 50% malicious peers, though this has been contested [77]).

### 1.2.1 History

The first version of the modern "web" is often accredited to T. Berners-Lee. Berners-Lee proposed a system to share text that included links and media [26], which he called "hypertext", "hyperlinks", and "hypermedia" (though these terms were not invented by Berners-Lee). There is much to say about this proposal but—for the scope of this thesis—we will focus on its interaction model. In this first version of the web, which we call Web1, users read data that is served by third party infrastructure. A user requests a web page and it is sent to the user by the infrastructure, ending the interaction. The "Web" (now capitalized) would slowly evolve into "Web 2.0" and become more interactive, with users actively interacting with third-party infrastructure [69]. Users may interact with web pages, uploading and downloading data to, and from, the third-party infrastructure. In the newest form of the Web, i.e., Web3, users no longer just interact with infrastructure but also actively take part in the creation and maintenance of this public infrastructure. The interaction model of this new version of the Web is contrasted to that of its predecessors in Figure 1.2.

The first materialization of Web3 came in the form of the cryptocurrency Bitcoin [164]. Grossly simplified, the goal of Bitcoin is to create a currency that does not rely on a single authority but, instead, on the majority of users in the system. In order to do so, Bitcoin uses a cryptographic puzzle (known as "Proof-of-Work") for its users to share information. The puzzle leverages computational complexity to make shared information difficult to construct but easy to verify. In turn, new information is only sporadically introduced into the system and all of its users have the time to receive and verify new information. However, as the number of users increased dramatically, the throughput of Bitcoin proved insufficient to serve as a digital currency. In turn, many alternative approaches with higher throughput were proposed which sacrifice connectivity, breaking Bitcoin's assumption that all users (are able to) verify all information. For example, the "Lightning Network" [181] enables multiple currency transfers locally between users before making their aggregation visible to other users of Bitcoin's blockchain. However, even though many alternatives exist, Bitcoin is still the dominant cryptocurrency to this day.

### 1.2.2 Blockchain data structures and their evolution

Traditionally, as pioneered by Bitcoin [164], a blockchain captures a list of transactions between peers. Peers in a blockchain system are known by their public key. Peers propose to transact with each other by providing a digital signature over an amount they wish to transfer to another peer. Nodes in the Bitcoin network, known as miners, collect proposed transactions and create blocks that capture a set of transactions. Each block has a hash pointer to the previous block in the blockchain's list of blocks. These blocks are special in that the miners try to find a nonce that makes the cryptographic hash of the newly proposed block start with a certain prefix (a given number of zeros). Finding a hash with the given prefix is difficult and takes time and computing power, slowing the introduction of new blocks in the system. If two blocks are found with the required prefix, then a fork of two competing chains is created and whichever block becomes part of the longest chain of successive blocks is the valid block, invalidating the other block. Finding blocks with a given prefix and adopting the longest chain of blocks is known as Proof-of-Work.

Blockchain systems have evolved to be less linear in structure. Whereas the linear data structure of the Proof-of-Work model has stood the test of time, the mining process has a large environmental impact and the throughput of transactions between peers is low. Therefore, alternative proposals have been made that modify the way in which the data structure is organized and the way in which blocks are accepted. For example, to optimize the data structure there are approaches that replace the list structure with a Directed Acyclic Graph. However, approaches also exist to optimize block mining through leader election, like Algorand [89]. The unifying aspect of most of these approaches is that data is no longer shared immediately with all peers.

### 1.2.3 Locality optimization and maintenance

One of the ways to make blockchains less linear in structure is to optimize transaction throughput and to decrease computation costs, is optimization for locality. Within the context of Web3, this "locality" is not based on physical proximity but rather logical proximity: if peers frequently interact with each other, or frequently interact with the same data, they are in the same locality. For example, a branch of a Directed Acyclic Graph may be locally updated by a small set of peers before being merged into the "main chain", which is an abstract description of "side chains" as it is used by, e.g., Bitcoin's Lightning Network. One of the downsides of the side chain approach is that the network partitions that are formed through the application of locality optimizations have lesser security guarantees. It is easier to launch a 51% attack on a group of 10 peers than it is to launch a 51% attack on a group of tens of thousands of peers.

A typical approach to deal with the security deterioration due to locality optimizations is to depend on random sampling. Traditionally, blockchain security depends on the fact that it is not easy for attackers to counteract the behavior of the majority of peers. For example, in a Proof-of-Work system it is assumed that attackers cannot overpower the computational power of the honest peers. The idea of randomness-based approaches is that it is infeasible for attackers to consistently be part of a random selection of peers. For example, when indefinitely randomly sampling, even if nine out of ten peers are malicious, eventually a random selection will contain the single honest peer. Therefore, randomness-based approaches critically depend on the selection of random peers not being malleable by attackers. Secondly, when an honest peer is eventually selected, malicious peers should be exposed by honest peers and the data structure can then be corrected. Early work that used randomly selected peers as “witnesses” to maintain a data structure is PeerReview [95]. More recently, Verifiable Delay Functions have been proposed for verifiable leader election to serve blockchains’ consensus protocols [35].

### 1.2.4 Web3’s scale and its resulting connectivity deterioration

Web3 is the newest public infrastructure for the interconnection of users. The goal of Web3’s users is—like in all interconnection networks—to share information. However, sharing information requires users to be connected through whatever communication medium they use. Historically, over the many years of development of interconnected networks, the connectivity between users is what breaks down as the number of users increases, i.e., as the infrastructure scales up. For example, having a human dispatcher create a physical connection between users of phones became untenable as the number of users increased. The same connectivity breakdown seems to be occurring in Web3, at least for Bitcoin [194], like in all interconnection networks that came before it.

The emerging lack of connectivity in Web3 should be addressed using mechanisms that can handle its scale. Just like circuit-switched connections in the telephone networks were superseded by packet switching networks, the network-wide broadcasts of information in network overlays should be replaced by mechanisms that are locality-aware, like publish/subscribe systems. However, using locality is not compatible with the assumptions of traditional Web3 technology like Bitcoin. The infrastructure maintenance of existing technology depends on information being available to all users, of which a majority must actively verify the information’s integrity. In Chapter 4, we describe a mechanism that allows for public infrastructure maintenance using locality.

## 1.3 Self-Sovereign Execution of Shared Code

A grand vision for Web3 is for it to be an ecosystem where peers that do not trust each other can act on common logic, using shared program code. This shared program code is known as a *smart contract*. Of course, having multiple nodes reach a consistent state using shared logic has been thoroughly explored in past works (e.g., forming the essence of the domain of Cloud Computing [43]). However, what sets smart contracts apart from traditional approaches is their use of replication of data to make up for the lack of trust in their ecosystem. In contrast, traditionally, replication is only used to address fault-tolerance (from faults in the network layer to the Byzantine behavior of applications).

This thesis explores a “local-first” approach to smart contract execution. The term “local-first” was introduced by M. Kleppmann and it is used to describe systems where each user generates data that is merged with data of other users later [118]. An example of such a local-first system is GitHub, where users create changes and propose them to other users. Eventually, the local proposals are merged into a global data structure. The assumption of local-first approaches is that most users, specifically humans, do not create change sets that conflict very often when collaborating. Therefore, by extension, a local-first system would not be in a perpetual inconsistent state. If this assumption holds for smart contract interactions as well, a local-first approach enables a new type of smart contract execution.

### 1.3.1 History

The use cases for smart contracts stem from the promise of trustless automated governance of both commercial and non-commercial applications. An early commercial use case for smart contracts was for them to serve as a decentralized autonomous organization. "The DAO" was the first such organization and it attempted to be a decentralized venture capital fund. The DAO is now mostly known for its failure, as a vulnerability in its smart contract code allowed a malicious party to steal roughly 70 million dollars worth of cryptocurrency. This early, and spectacular, failure of smart contracts has certainly convinced some that they are a "bad idea" [168]. Nevertheless, the promise of a decentralized token economy (using smart contracts) remains appealing to revolutionize business [132]. Furthermore, smart-contract-based distributed collaborative organizations stand to revolutionize governments and businesses alike [58].

Regardless of the promise of smart contracts, many problems exist with their execution model. We highlight the two problems of the execution expenses and the dependence on traditional blockchains here. Firstly, executing smart contracts is expensive monetarily. A fee must be paid to whatever node captures smart contract executions in a blockchain block (i.e., when a smart contract is created or whenever the state of a smart contract is updated due to a function call). The cost of single contract function calls can range into the thousands of dollars [175]. Most of the cost of executing smart contracts stems from every single node in the blockchain being required to run the code (if it was not necessary to run the code before accepting a block, no additional cost would be incurred). Secondly, smart contracts depend on network-wide consensus to ensure the order of their function calls (and thereby their implied state). Just like with any state machine, the state of a smart contract depends on its previous state and the function call that followed it. There is no guarantee that any two smart contracts will reach the same state if either their preceding state or their function calls differ. Nevertheless, a state can be retroactively corrected (i.e., rolled back), which we explore in this thesis.

### 1.3.2 Consistency and consensus

In modern Web3 systems a distinction is made between consensus and consistency, though in traditional Web3 systems like Bitcoin this distinction is not made. Bitcoin is driven by two layers of consistency: transactions must be shared with miners to be included in blocks and blocks must be shared before a longer chain of blocks emerges. Both consistency layers of Bitcoin have been attacked (e.g., through Eclipse attacks [245] and selfish mining [77]). However, if both layers do eventually successfully disseminate their respective information (transactions and blocks, respectively), peers in the Bitcoin network can agree on the fact that one chain of blocks is the longest. Of course this agreement, consensus on the longest chain, can change if the consistency layer that synchronizes the blocks updates its dominant history. In traditional consensus mechanisms, such a reversal would not be possible and this led to this mechanism being called “metastable consensus”.

Whereas Bitcoin muddles the definition of consistency and consensus, modern approaches aim for a cleaner split between the two layers. First, a consistency layer ensures that data (in currencies this data would be transactions) is known to peers. Which peers to share data with and how the data is disseminated can vary. Weaker models like eventual consistency only guarantee that peers will receive data in finite time [16], whereas traditional broadcasts make sure all network participants receive data as quick as possible. Second, a consensus mechanism evaluates whether the consistent information should be accepted. A more efficient mechanism, like that of Bitcoin, may decide locally to accept or reject information. These relaxed consensus mechanisms depend on emergent consensus through all peers sharing the same logic and (eventually) the same information. Nevertheless, some consensus alternatives opt for more traditional network-wide broadcasts to form consensus, which takes longer but provides better guarantees on the finality of decisions. What types of consistency and consensus mechanisms are acceptable depend heavily on the given application.

### 1.3.3 Smart contracts

A smart contract consists of virtual machine instructions that are shared through a blockchain. The virtual machine instructions are usually not handwritten by users but instead compiled from a higher-level language (like the Solidity language). Every smart contract exists in a given memory address in a shared virtual machine. In case the machine instructions are called in such a way that the state of the machine is updated, the call must be shared through a blockchain. Specifically, if a smart contract specifies a given writable memory location and a peer interacts with the contract in such a way that the value in the memory location updates then this interaction must be shared. By sharing these writes, all peers that receive the same interactions in the same order will obtain the same state in their local virtual machine.

The state of a smart contract heavily depends on the consensus layer that powers the interactions. In case of rollbacks in the main chain of a blockchain that underlies a smart contract, the state of the smart contract will have to be recalculated. Therefore, smart contract execution is most suited for traditional blockchains that do not see many changes in their consensus layer. At the same time, the long waiting time before blocks are mined and the corresponding smart contract instructions are finalized can also form a vulnerability. One of the attacks on smart contracts enabled by the slow throughput of blockchains is the front-running attack [56]. An attacker can use its knowledge of transactions that have not yet been accepted to propose and reorder new transactions to its own benefit. Of course, for some use cases the existence of these attacks may be an acceptable loss to enable the shared state of a smart contract but this heavily depends on the application.

### 1.3.4 Local-first

A local-first approach is an approach to writing collaborative software that favors availability over consistency and partition tolerance. The idea consists of users making changes locally and merging these changes with the changes of others at a later time. For example, in Git a user makes local changes before merging their program code with the main code branch. Only when a user makes an attempt to merge, is it revealed that the local state may have to be rolled back. The explicit assumption is that merges succeed more-often-than-not without requiring rollbacks. Of course, data structures that support more granular merging strategies are better suited for such a collaborative workflow. Therefore, the original local-first work suggests the use of "conflict-free replicated data type" (CRDT) constructions to ease merging of data [118]. The same construction can be applied to collaborative interaction with a smart contract.

The consequences of applying a local-first approach to smart contracts leads to a change in guarantees in relation to traditional blockchains. Traditionally, smart contracts depend on the fact that all peers in a network interact with code in consensus rounds. By favoring availability, a local-first approach may imply that users operate on an unmergable state without knowing it is unmergable and commit to business logic too early. For example, when using tokens for access management, an access token might be claimed locally at the same time by two peers. In this example, business logic may dictate that only one token should be claimed at the same time. Secondly, traditionally there is a monetary incentive for miners to accept all valid interactions. If merges of states are no longer the main driver of the shared data structure but a secondary effort, some applications may suffer from attackers that delay interactions. In the token example, a claim of a token of another peer could be delayed in order to claim the same token and sell it for a higher price to the original peer that attempted to claim it. In Chapter 5, we implement a local-first approach to smart contracts that addresses these issues.

## 1.4 Research Questions

This thesis revolves around providing the first insights into the solution space of Self-Sovereignty for modern peer-to-peer ecosystems, i.e., Web3. The overarching research question this thesis focuses on is:

*What technology is appropriate to enable Self-Sovereignty for a Web3 ecosystem?*

To answer this research question, we explore the three domains of Web3 ecosystems in a total of four related research questions. We define two questions for the domain of digital identities, one question for the domain of public infrastructure, and one for the domain of shared code. In most cases, the answer to a research question (perhaps indirectly) enables another research question to be solved in a Web3-compatible way. The research artifacts that are produced when answering these questions may directly implement a full Web3 solution or only support existing solutions. These relationships are shown in Figure 1.3.

**[RQ1] What components should the architecture of a Self-Sovereign Identity Management System leverage?** A Self-Sovereign Identity solution must meet the ten properties, laid out in Section 1.1.3, of *existence*, *control*, *access*, *transparency*, *persistence*, *portability*, *interoperability*, *consent*, *minimalization*, and *protection*. However, these properties do not directly lead to functional requirements for systems or the necessary system components to provide them. A Self-Sovereign Identity management solution should provide identity data control and private data minimization by the subject. Furthermore, beyond these basic properties, additional requirements must be met to have any use as

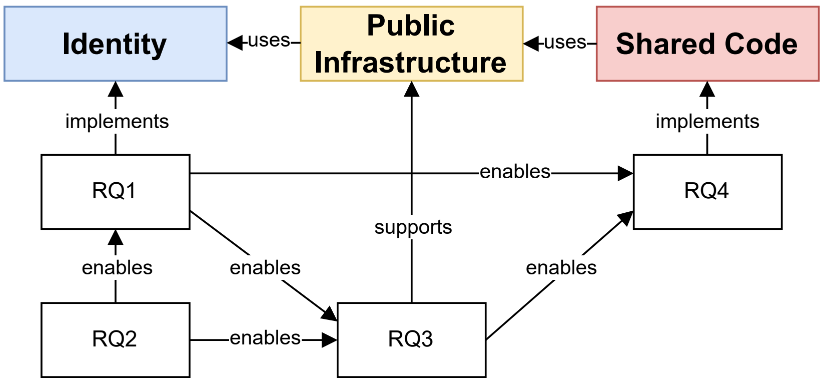


Figure 1.3: The relationships between the four research questions (RQ1-4) of this thesis and the domains of identity, public infrastructure, and shared code.

a substitute for our current physical means of identification like passports. A passport-equivalent identity management solution should provide legality and validity of data. Lastly, a Self-Sovereign Identity management system needs to provide privacy guarantees to protect its users. Our goal is to identify the functional requirements and system components necessary to enable such a system.

**[RQ2] How can connections be established between Self-Sovereign peers without depending on third parties?** In order for peers to interact, typically using a gossip protocol, they must be able to communicate with each other. However, one of the big unresolved attacks on peer-to-peer networks is the blocking of communication using fake identities. In particular, we focus on the *Sybil attack* [70], where a single party exploits the ability to cheaply and easily create many identities. There is a circular dependency between requiring Sybil-free gossip protocols and using gossiped information to eliminate Sybils. Gossip protocols do not guarantee delivery of information in a network filled with Sybils and Sybils cannot be identified without the information gossiped through these protocols. Our goal is to break this circular dependency by avoiding connecting to Sybils.

**[RQ3] How can a public infrastructure be maintained by peers in a Self-Sovereign fashion?** Self-Sovereign nodes are primarily concerned with their own data, not with that of others. Forcing all peers to be concerned with the data of all other peers is not in line with Self-Sovereign incentives. However, the shared public infrastructure degrades if peers no longer verify each other's data. Therefore, all data should be available to all nodes, allowing them to identify when the infrastructure degrades and to correct it. Our goal is to provide a Self-Sovereign incentive-compatible method to maintain shared public infrastructure without forcing all peers to perform verification of other peers' data.

**[RQ4] How can a Self-Sovereign model support shared code execution between peers?** Shared code is not necessarily compatible with Self-Sovereign incentives and data management. Generally speaking, not all program code will produce the same output given different or reordered input values. If data is local to each peer, peers may not be able to observe (all) interactions with shared code, they may not be able to observe interactions in the right order and, perhaps peers may only be able to observe interactions after a long time has passed. Our goal is to provide insight into the ramifications for applications using a Self-Sovereign model for shared code.

## 1.5 Research Context

The idea of Self-Sovereignty has spawned initiatives from many global institutions, including blockchain infrastructure and identity management solutions. The World Wide Web Consortium (commonly known as W3C) has provided the widely popular Decentralized Identifier (DID) recommendation [207], in an attempt to standardize the communication protocol for Self-Sovereign identity data exchanges. This recommendation was co-authored by C. Allen. The "Blockchain For Humanity" project of the United Nations[1] saw proposals to use Self-Sovereign Identities to prevent modern day slavery, in particular to prevent child trafficking. These first two initiatives are also closely tied to the Sovrin foundation [239]. The European Union has launched the European self-sovereign identity framework (eSSIF[2]), the European Digital Identity (EUDI[3]) identity wallet, and the blockchain initiative European Blockchain Services Infrastructure (EBSI[4]). For the latter, the European Commission has multiple parties host nodes, including the Delft University of Technology. On the national level, many organizations like the Polish Blockchain Association[5] and the Dutch Blockchain Coalition[6] attempt to use blockchains and Self-Sovereign Identity within the context of their government and industry. Even national governments themselves are involved with the Self-Sovereignty movement and below we discuss the Dutch government in particular, as it has more directly inspired Chapter 2.

The global focus on blockchain and identity research—to empower citizens and give them more freedom—spawned major conspiracy theories. In particular, the ID2020 project by Microsoft would be a target of these theories, being accused of (among other things) building a system for mass surveillance [224]. Whereas this thesis cannot prove or disprove this theory, mass surveillance is certainly not the goal of the wider field of blockchain and Self-Sovereign Identity research. In fact, the Self-Sovereign Identity research of this thesis can (and should) supersede the vaccine passports that conspiracy theorists so revile.

The research presented in this thesis has been carried out as part of the research vision of the Delft Blockchain Lab (DBL)[7], a collaboration of the Distributed Systems group, the Cybersecurity group, the Intelligent Electrical Power Grids group, and the Ethics/Philosophy of Technology group of the Delft University of Technology. This thesis focuses on the critical base layer, a public infrastructure, to enable other research output from the DBL. For instance, the PhD thesis "Decentralization and Disintermediation in Blockchain-based Marketplaces" by M.A. de Vos focuses on decentralized marketplaces [63] and the PhD thesis "Incentives and Cryptographic Protocols for Bitcoin-like Blockchains" by O. Ersoy focuses on the cryptographic primitives to enable blockchain trading [74]. Clearly, both of these theses require a (blockchain-based) public infrastructure to be in place. All

[1]`https://ideas.unite.un.org/blockchain4humanity/Page/Home`
[2]`https://essif-lab.eu/`
[3]`https://eudiwalletconsortium.org/`
[4]`https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home`
[5]`https://blockchain-polska.org/`
[6]`https://dutchblockchaincoalition.org/`
[7]`https://www.tudelft.nl/delft-blockchain-lab`

in all, the collective DBL research has succeeded in bringing together the three societal pillars of academia, industry, and government. We now discuss how these three pillars have inspired, influenced, and employed, the research of this thesis.

As a part of the societal pillar of academia, this research has been integrated into the open source software Tribler [183], the research vehicle for cooperative systems research of the Distributed Systems group of the Delft University of Technology. Tribler offers peer-to-peer filesharing and peer-to-peer search. For its networking layer, Tribler depends on other research from the TU Delft's Distributed Systems group, the peer-to-peer content synchronization framework called Dispersy [247]. After half a year of engineering effort by the author of the present thesis, Dispersy has been redesigned to let go of content synchronization and instead focus on providing identities and overlay network primitives. This new focus led, in 2017, to a new distinct version of Dispersy called IPv8[8]. IPv8 offers privacy by design, i.e., peer-to-peer exchanges that expose a minimal amount of personally-identifiable information, including a complete identity solution to do so (discussed in this thesis). Part of the reason that the academic research on IPv8 became entangled with the initiatives of industry and government, through the Dutch Blockchain Coalition, was the (at the time, in 2017) looming General Data Protection Regulation (GDPR), which would come into effect in 2018 in the European Union (discussed later in this thesis).

As a part of the societal pillar of industry, this research was conducted in close communication with the Dutch Blockchain Coalition (which would also have a physical presence on the Delft University of Technology campus for a period of time). Though the coalition had many industry members, we would mostly be in closer contact with the (quasi-)government members. Some of the members that we were in close contact with are the (Dutch) Authority Financial Markets, the ING Bank, and the Netherlands Organisation for Applied Scientific Research (TNO). The latter member would eventually take on a leading role in the development of eSSIF (the European self-sovereign identity framework). One member that was a key party in relation to this thesis is the Ministry of the Interior and Kingdom Relations (BZK). In cooperation with BZK and their passport-chip supplier IDEMIA, we created a pilot for the identity research of this thesis. Together with CMS lawyers and jurists from BZK, a Privacy Impact Analysis was created to, presented to, and approved by the Dutch Data Protection Authority (Autoriteit Persoonsgegevens).

As a part of the societal pillar of government, this research has been integrated into a pilot with BZK. The original intention was for this pilot to be deployed in the municipalities of Utrecht and Eindhoven [212]. For non-technical reasons, this did not end up happening. We believe that this thesis—though it is not officially noted—has been a part of the motivation for the Dutch government to change the Dutch passport law, in particular Amendment 35047 (R2108), 1 Oct 2018, of Article 23. This amendment allows the Dutch government to explore more novel methods for digital identity management. On a coarser scale and in particular relation to this thesis, the author of this thesis has served as a consultant of the European Union Agency for Cybersecurity to give recommendations for blockchain-based identity wallets.

[8]`https://github.com/Tribler/py-ipv8`

Table 1.1: The method of evaluation, and availability of its artefacts, per research question of this thesis.

| RQ | Evaluation | Availability | Publication Date |
|---|---|---|---|
| 1 | simulation + real-world | code | Aug 9, 2017 |
| | `https://github.com/Tribler/py-ipv8` | | |
| 2 | simulation + real-world + trace | data | Jan 6, 2020 |
| | `https://doi.org/10.4121/uuid:34850d65-1908-4249-b446-8e87c6d21ba0` | | |
| 3 | simulation + trace | code | Sep 26, 2022 |
| | `https://doi.org/10.4121/85a871c5-3782-4a21-ae7f-7d52e9f019f7` | | |
| 4 | simulation + trace | - | - |

## 1.6 Research and Engineering Methodology

The concept of Self-Sovereignty has many facets and it appears in the fields of cryptography, networking, distributed systems, and social sciences. For example, within the scope of cryptography conferences, the Self-Sovereign disclosure of data between peers is often the focus of research. Our focus, in this thesis, is on systems for Self-Sovereignty and we adopt research and engineering methodologies that are typical for networking and distributed systems research. In particular, we use the approach of qualitative analysis of system components, comprehensive experiments, and open-source system code.

In this thesis we adopt an experimental research methodology. Every research question is answered through qualitative analysis and subsequent mechanism design, implementation, and evaluation. As the goal of this thesis is to explore solution spaces, we aim for our work to be as reproducible and as usable as possible. Even though thesis predates the requirement of a Data Management Plan for its data (which is now mandatory for newly-starting TU Delft PhD students), it still follows the same best practices. When permissible by license restrictions, code is open source and data sets are publicly available through 4TU.ResearchData (a data repository). In one case, for the mechanism belonging to RQ1, the license of the open source software does not allow it to be mirrored in the 4TU repository. Additionally, we implement all of the solutions we propose in Python in such a manner that they can either be directly used or easily added to the existing maintained and deployed open-source software Tribler [183]. Within our experimental evaluation of these solutions we use real users of Tribler as much as possible. Of course, in light of responsible research, we do not disproportionately burden or attack the users of our framework.

Each of our research questions has led to research artefacts, in the forms of code and data sets. We provide detailed descriptions of the methods of evaluation and availability of artefacts when we discuss each research question in its own chapter (Section 1.7 presents the mapping of research questions to chapters). In general, we use three different methods of evaluation. The evaluation of each mechanism includes a *simulation* of peers that interact using the respective mechanism, using a real-world latency trace if possible. If possible, ethical, and feasible, we include interaction data of the proposed mechanism with *real-world* users of our proposed mechanism. Lastly, we employ *traces* of real-world interactions if they are available. For each of our mechanisms we offer open source code if its implementation is non-trivial. However, we do not make our shared program code prototype of RQ4 available to avoid misuse for financial gain. A summary of the evaluation and availability of artefacts is given per research question in Table 1.1.

## 1.7 Thesis Outline and Scientific Contributions

The following four chapters (Chapter 2-5) address our research questions (RQ1-4 of Section 1.4) in their presented order. We now discuss the content, and the contributions, of each chapter.

**[Chapter 2] A Truly Self-Sovereign Identity System.** In this chapter we address RQ1 through a discussion of what functional requirements and what properties a Self-Sovereign Identity system should meet. We present *TrustChain IDentity (TCID)*, our prototype solution that has been created in collaboration with the Dutch government to meet the use case of a digital passport analog. Our key contribution is the discription of a Self-Sovereign Identity system that truly does not use (or trust) any third party servers. Users of our prototype need only communicate with issuers of credentials and verifiers of credentials directly, without communication between the issuers and verifiers themselves. We expose the need for the three desirable system properties of *Self-Sovereignty*, *Credibility*, and *Network-level Anonymity*. We define the seven functional requirements of *direct communication*, *message authenticity*, *message integrity*, *network-level anonymization*, *decentral synchronization of messages*, *identification through credentials*, and *accountability of credential use*. Our results show that the overhead of using TCID is not prohibitive for practical use. This chapter is based on the following publication:

Quinten Stokkink, Georgy Ishmaev, Dick Epema, and Johan Pouwelse. A truly self-sovereign identity system. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 1–8. IEEE, 2021.

The system discussed in Chapter 2 is a continuation of previous research. The chapter encompasses the lessons we have learned after maturing the research artefacts first presented in the following publication (not included in this thesis):

Quinten Stokkink and Johan Pouwelse. Deployment of a blockchain-based self-sovereign identity. In *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, pages 1336–1342. IEEE, 2018.

**[Chapter 3] Web3 Sybil avoidance using network latency.** In this chapter we address RQ2 by defining a method for peer discovery that does not require peers to trust the accuracy of information from other peers. In fact, peers use only the network latencies (to other peers) that they measure themselves. Using these measurements, peers discover each other while avoiding connections to Sybils. We present the *SybilSys* peer discovery mechanism that filters connections to Sybils over time. The key contribution of SybilSys is that it allows for peer discovery while only depending on network latency measurements. In the chapter we discuss why, and how, to implement and verify light-weight Sybil-avoiding peer sampling through filtering connections on unique network latencies. Furthermore, we identify that a naive approach to filtering, only based on unique latencies, is open to the attack of delaying latency measurements. We resolve the measurement-delaying attack by using the queueing delay of messages to detect tampering with latency measurements. This chapter is based on the following publication:

Quinten Stokkink, Can Umut Ileri, Dick Epema, and Johan Pouwelse. Web3 sybil avoidance using network latency. *Computer Networks*, 227:109701, 2023.

**[Chapter 4] Reputation-Based Data Carrying for Web3 Networks.** In this chapter we address RQ3 by giving an emergent algorithm that allows peers to exchange data. The exchanged data allows peers to maintain their public infrastructure. We present our *Timely Sharing with Reputation Prototype (TSRP)*, a prototype solution that uses probabilistic guarantees to select carriers for data. These carriers are peers that store and share data. The key contribution of TSRP is its trustless exchange of data without depending on network-wide broadcasts. A witness protocol is used by peers to audit each other. We motivate and prove the emergent property of data being carried when using TSRP. We emulate multiple networks to show that TSRP properly selects more carriers when reputations are low and vice-versa. Finally, we let TSRP operate on real-world traces of Bitcoin and Twitter to show that our method does not necessarily force all peers to store all data in the system. This chapter is based on the following publication:

Quinten Stokkink, Can Umut Ileri, and Johan Pouwelse. Reputation-Based Data Carrying for Web3 Networks. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 283–286. IEEE, 2022.

Chapter 4 includes a more elaborate analysis of the basic behavior of TSRP in comparison with the above publication and we highlight three of their main differences. Firstly, the chapter contains more details on the choices for parameterization of TSRP in its design section. Secondly, a section is added that verifies the basic behavior of TSRP. Lastly, a second real-world trace of Bitcoin is evaluated next to the trace of Twitter.

**[Chapter 5] A Local-First Approach for Green Smart Contracts.** In this chapter we address RQ4 by exploring a local-first approach to program code execution. Peers execute program code optimistically and they roll back if they derive a change in decision from other peers. We argue that this approach also leads to less carbon emissions and, therefore, a greener solution. The three main drivers for greener smart contracts that we identify are probabilistic consensus mechanisms, weak consistency models, and decentralized strong identities. We present the local-first *Green Smart Contract (GSC)* paradigm for the execution of smart contract code and we motivate that economic liabilities best support it, instead of the economic assets that support traditional blockchains. Our results show that a system that uses a single publisher to handle and exchange smart contract updates leads to light-weight communication costs. Lastly, we apply our GSC-based prototype to a real-world trace to show the impact on the communication cost of the GSC paradigm. This chapter is based on the following publication:

Quinten Stokkink and Johan Pouwelse. A local-first approach for green smart contracts. *Distributed Ledger Technologies: Research and Practice*, ISSN 2769-6472. ACM, 2023.

**[Chapter 6] Conclusion.** This thesis concludes with a summary of the conclusions of the individual chapters and the future directions that have been exposed.

# 2 A Truly Self-Sovereign Identity System

*Existing digital identity management systems*[1] *fail to deliver the desirable properties of control by the users of their own identity data, credibility of disclosed identity data, and network-level anonymity. The recently proposed Self-Sovereign Identity (SSI) approach promises to give users these properties. However, we argue that without addressing privacy at the network level, SSI systems cannot deliver on this promise.*

*In this chapter we present the design and analysis of our solution TrustChain IDentity (TCID), created in collaboration with the Dutch government. TCID is a system consisting of a set of components that together satisfy seven functional requirements to guarantee the desirable system properties. We show that the latency incurred by network-level anonymization in TCID is significantly larger than that of identity data disclosure protocols but is still low enough for practical situations. We conclude that current research on SSI is too narrowly focused on these data disclosure protocols.*

[1]**A historical perspective on this chapter.** The paper that this chapter is based on was written in 2020 and published in 2021. Four years later, 2024, more solutions have become available that are not considered in this chapter because this chapter preceeds their existence. Other solutions have evolved. Some examples are that the company Ver.iD was founded in 2022, the company Sphereon implemented its backend (OID4VC) in 2022, since 2021 uPort is known as Veramo and since 2023 the IRMA solution is known as Yivi. The W3C DID standard became in Candidate Recommendation Draft in 2021 and a Recommendation in 2022.

## 2.1 Introduction

In its full generality, the problem of digital identity management is one of the most important and hardest problems associated with large-scale digital infrastructures. On the one hand, digital identities are critical elements in the pervasive digital infrastructures we use in daily life for economic activity, access to healthcare and public services, etc., but on the other hand, data breaches, data leaks and privacy violations are all too common with the current generation of digital identity management systems [67]. These issues can at least partially be attributed to the reliance of these systems on centralized trusted third parties to manage all aspects of users' identities. Over the last few years, the concept of decentralized management of digital identities labelled *Self-Sovereign Identity* (SSI) has emerged that promises users control over how to collect, store and share their own identity data [234]. This chapter presents the design, implementation and evaluation of a complete, open-source, truly Self-Sovereign Identity System.

SSI systems minimize trust in third parties and assume a decentralized infrastructure for private storage by the identity holders or *subjects* of credentials, which are confirmed pieces of identity data. Trusted third parties act only as *issuers* of credentials on request by subjects and cannot learn with whom or when subjects share their credentials. Third parties cannot learn anything about subjects except for the credentials that are explicitly shared with them. The SSI approach, in theory, has the strong privacy guarantees of *data minimization* and *private data control* [4] as subjects can use any data disclosure protocol they like for selectively sharing credentials and for implementing any cryptographic or data obfuscation algorithms on the corresponding data.

The SSI approach is unique in its ability to serve as a digital analog for identification in the physical world. What constitutes the strength of identification in the physical world is that it (a) is always presented by its owner (e.g., showing a passport), (b) is legally and practically recognized as a valid proof of identity (e.g., by verifying the passport photo) and (c) is only shared between the identity owner and the verifier without knowledge of any other party (e.g., the state that issued a passport does not know whom it is presented to). The digital analog of these desirable physical properties are the digital system properties of (a) *Self-Sovereignty*, (b) *Credibility*, and (c) *Network-level Anonymity*.

Though promising as a concept, current research on SSI is limited in its focus on data disclosure protocols for maintaining the Self-Sovereignty of data. However, these protocols are definitely not sufficient on their own to deliver strong privacy guarantees in practical applications. This is evident from the research on privacy-focused decentralized systems such as anonymous crypto-currencies [80], which shows that not only protocol-level privacy, achievable with cryptographic tools, but also network-level anonymity is necessary for practical privacy-preserving applications [32]. Network-level anonymity, that is, obfuscating the source-destination pairs of messages for credential creation and sharing, is crucial in SSI systems since Internet traffic correlation can completely undermine the effect of the data-disclosure protocols [217].

The design of our SSI system is based on 7 functional requirements that we deduce from the three system properties it needs to provide, which relate to basic trustworthy messaging in a communication substrate and to network-level anonymity for credential creation and verification in two peer-based overlays on top of this substrate. Our design is modular in that it allows for the specialization of the system for applications with differing

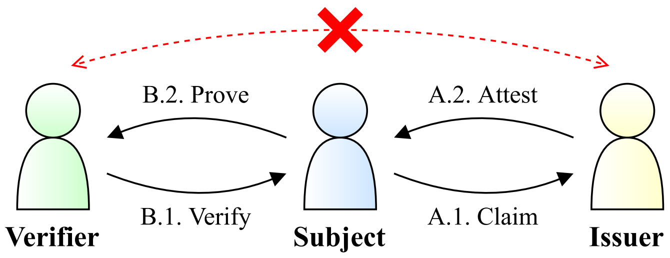


Figure 2.1: The two communication flows, A and B, when enrolling and verifying a credential: both flows involve the subject, as communication between the verifier and issuer violates the Self-Sovereignty property.

privacy and security requirements by incorporating different software modules for several functionalities. In particular, we discuss the specialization of our design for use as a digital analog of a passport.

We are not the first to propose using network-level anonymity for Self-Sovereign Identity systems [217], but we are the first to create a viable prototype in a truly Self-Sovereign, zero-server manner. The contributions of this chapter are:

1. We provide a complete and substantiated modular system design for the creation of Self-Sovereign Identity solutions with strong privacy called TrustChain IDentity, or TCID for short (Section 2.3).

2. We present the specialization of TCID for a passport-grade Self-Sovereign Identity solution and we discuss methods for revocation of credentials (Section 2.4).

3. We perform a performance analysis of TCID that shows that the overhead of TCID is not prohibitive for practical use (Section 2.5).

## 2.2 Problem Statement

The main research problem addressed in this chapter is "What components should the architecture of a Self-Sovereign Identity Management System leverage to provide the properties of *Self-Sovereignty*, *Credibility*, and *Network-level Anonymity*?" Secondarily, the architectural components required for guaranteeing Credibility and Network-level Anonymity may have an impact on the latency of attesting to and verifying credentials, which would seriously hamper the practical use of an SSI system. Therefore, the second research question is whether the overhead of these components is prohibitive for practical use.

The *Self-Sovereignty* property, which is defined as identity data control and private data minimization by the subject, means that subjects can build their own sets of multiple pieces of identity data (their *credentials*) by having issuers attest to them, and can selectively prove credentials to verifiers when performing transactions in a digital infrastructure. For this purpose, an SSI solution should only allow the following two types of communication flows between issuers, subjects and verifiers, with four types of messages (see Figure 2.1):

- **Flow A. Enrolling a credential:** a request for credential creation by a subject (A.1) and the attestation for this credential by an issuer (A.2).

- **Flow B. Verifying a credential:** a request for proof of a credential by a verifier (B.1) and the subsequent proof of this credential by the subject (B.2). This may also include a check to determine if the credential was revoked, which we discuss in Section 2.4.4.

As indicated in Figure 2.1, a direct data flow between the issuer and the verifier is not permitted. An analogous physical example of these digital interactions is a citizen claiming to be older than 21 (A.1) and a government issuing a passport that proves this claim (A.2). The citizen in this example could then later attempt to buy liquor in a liquor store which would ask for proof that this citizen is over 21 (B.1) for which the citizen shows his passport (B.2). The government (the issuer) does not directly communicate with the liquor store (the verifier). Nevertheless, the liquor store accepts a passport as proof because it knows it is issued by the government and the government is trusted: the claim is credible.

The second property of *Credibility* is defined as credentials being legally and practically recognized as valid proofs of identity data. The two cornerstones of credibility assurance in the digital domain are *digital signatures* and *user authentication*, which we now discuss.

Digital signatures by legitimate authorities, i.e., digital signatures by third parties that are trusted by both a subject and a verifier, serve to attest to the credibility of presented claims (A.2 in Figure 2.1). Acting as an issuer, such an authority signs a claim to provide a cryptographically verifiable proof of credibility, forming a presentable credential. For example, subjects may present a credential that shows their age and is digitally signed by a government, an issuer that most subjects and verifiers trust to be a credible source of age information. Subjects have the ability to collect signatures of multiple authorities to further enhance the credibility of a credential, similar to how users sign each other's keys to establish credible identity in the PGP web of trust [249]. However, this transitive credibility is not a one-way relationship from authorities to credentials. Instead, the credibility of a credential also influences the legitimacy—and thereby the credibility—of authorities [91]. Credentials with provably incorrect data result in a damaged reputation of the authority who signed them. Thus, issuers must be able to either retroactively change the data they signed or be able to revoke signatures of subjects to uphold their own credibility.

The second cornerstone of credibility is user authentication of the subject. While an interaction with a subject may be authentic and covert, device theft may have occurred: the holder of the device might not correspond to the enrolled identity. This means that in addition to deriving credibility from the signing authority, credibility is derived from the authentication of the user holding the device, which allows an application to bind the physical user to a device.

The third and final property of SSI management systems is *Network-Level Anonymity*. Existing work shows that an adversary who is able to observe traffic in a network can detect a pattern of a subject's associations that may lead to identity correlation [72], which we call an "identity correlation attack". In the context of SSI systems, without network-level anonymity, the credential enrollment and verification traffic of subjects can be observed, which, even though the data in the messages is encrypted, makes them vulnerable to identity correlation attacks[2] [208]. In its simplest form, even without analyzing a so-

[2]Though bad for users' privacy, a single centralized attesting authority avoids this, as no social network can be derived (i.e., all users will only ever interact with a single party that signs their data, the centralized authority, making all interaction graphs equal).

cial network, this correlation may be observed when there is only a single IP address that sends the messages for two distinct identities, which allows an adversary to determine that these identities belong to one user. Thus, an SSI management system needs to provide anonymity guarantees at the network level to preserve the protocol-level privacy guarantees.

## 2.3 System Design

In this section we present the system components that jointly deliver the *Self-Sovereignty*, *Credibility*, and *Network-level Anonymity* system properties as implemented in our open-source system TCID, which provides a framework that enables SSI-based messaging between applications that require identification. These applications may range from using a mobile phone to buy alcoholic beverages in a bar to a pre-flight registration over the Internet. Without loss of generality, for all applications we treat the subject, issuer and verifier as separate entities using our system, though in practice the issuer and verifier are often the same entity (e.g., bank account credentials being used to authenticate with the same bank). To guide the introduction of TCID, in Figure 2.2 we show its system components, distributed across a communication substrate and peer-to-peer-based overlays. These components jointly satisfy the *7 functional requirements* detailed below that we identified for enabling the SSI interactions for credential enrollment and verification over the Internet. To focus on the high-level behaviors of the components, we deliberately omit their implementation (linked in Table 1.1, RQ1) as it consists of tens of thousands of lines of code and dozens of distinct messages. The communication substrate can be implemented on top of different communication media, and in addition to our Python-based Internet implementation, we have also created a Kotlin-based Bluetooth version.

We believe TCID to be a valid reference architecture as it can even be made to support the use case of digital passports, which has arguably the most challenging set of design requirements. TCID has been created in tight collaboration with both government and industry. This was necessary to satisfy anticipated future legislation on the storage and use of identity data in Self-Sovereign Identity solutions. However, authentication levels for digital identities have long since been standardized by the National Institute of Standards and Technology [90]. TCID can be specialized for specific applications by enabling the corresponding authentication levels in a modular and transparent way, as discussed for *passport-grade* identities in Section 2.4.

Regardless of its specialization, the communication substrate of TCID (shown in Figure 2.2) serves three functional requirements for messaging. First, peers must be able to *(1) communicate* with other peers directly, i.e., without any specific trusted third party, which can be solved using the address space of a medium (like the IPv4 and IPv6 protocols). This direct communication circumvents the need for centralized portals that manage user identities. Secondly, it must be impossible for peers to impersonate each other: messages should be *(2) authentic*. Lastly, the communication substrate should ensure messages are not modified by anyone other than their originator: messages should maintain *(3) integrity*. The second and third requirements are implemented using digital signatures from decentralized PKI (public key infrastructure).

Having a communication substrate for end-to-end messaging is not enough for all possible uses of digital identity and TCID provides two network overlays to allow further

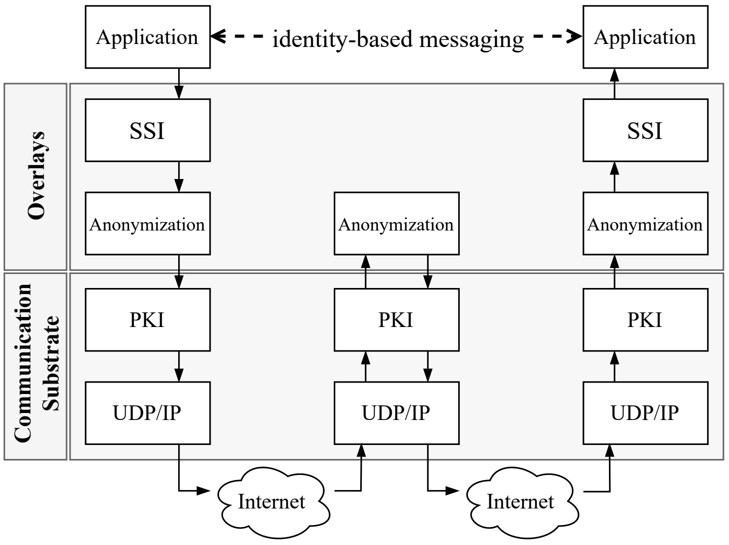


Figure 2.2: The data flow through the TCID components for applications with one intermediary for anonymization, using UDP/IP over the Internet for the communication substrate.

strengthening of its identification mechanism. If attackers are able to identify users based on their messaging patterns, the security assumptions of SSI solutions are violated [217]. For an attacker can then both disrupt communication and track users through their messages. Practically, these attacks may be as simple as sending a large number of messages or tracking IP addresses. Therefore, the first overlay on top of the communication substrate in Figure 2.2 facilitates decentral *(4) network-level anonymization.*

The anonymization overlay uses randomly-selected peers with user-grade hardware (like mobile phones) in the overlay network to create covert multi-hop communication channels (Figure 2.2 shows a two-hop channel). These peers are randomly selected and are therefore neither specific nor trusted third parties. If no peers are available as intermediaries, the anonymization is essentially reduced to an expensive form of link encryption and requires the communication medium itself to not leak device identifiers. For creating the multi-hop channels, TCID implements its own derivative of the Tor protocol [68]. This customization is necessary as anonymization—while aiding in averting attacks [217]—cannot be applied as an afterthought to existing protocols [31]. Tor's "hidden services" protocol normally requires centralized *directory servers* to share the peers that can be used for constructing channels [173]. As these directory servers are again specific and trusted third parties, which TCID wants to avoid, we have replaced them with a gossiping protocol over the anonymization overlay. Peers in TCID are able to *(5) decentrally synchronize* the information normally published by directory servers. However, this gossiping approach is classically vulnerable to both the Index Poisoning attack and the Eclipse attack through fake identities called Sybils. TCID uses a method of testing physical resources (using latency) to avoid these Sybil attacks in its anonymization overlay [235], our method is described in Chapter 3.

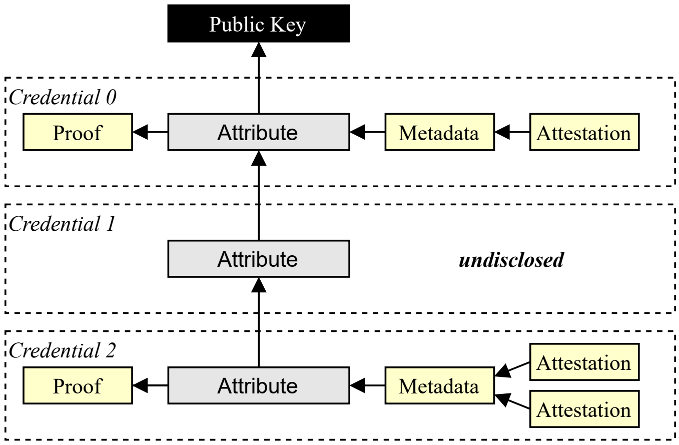


Figure 2.3: A pseudonym data structure corresponding to the public key of a subject with two disclosed credentials. Arrows denote an element storing the hash of the element pointed to. Undisclosed credentials include only an attribute, whereas their disclosure also includes a proof, metadata and attestation(s).

Peers use the SSI overlay of TCID to disclose their credentials over covert channels, i.e., to *(6) identify* themselves in a Self-Sovereign fashion. For the most stringent use cases of SSI, the overlay also offers functionality for *(7) accountability* of subjects, e.g., for a government to identify an individual in case of overstaying a visa after crossing a border. The SSI component facilitates storage for subjects and verifiers of the public and private (secure) data that allows for identification and accountability: *public key pairs*, *credentials* and *pseudonyms* (explained in Section 2.4). Furthermore, the SSI overlay handles the SSI-related data flows of Figure 2.1 between peers. The identification and accountability functionalities are exposed to the application layer, providing identity-based messaging.

## 2.4 Passport Grade Specialization of TCID

In this section we further specify how the SSI component of TCID can be configured and used to support various levels of authentication, up to the point of being able to support use cases that normally require a physical passport. This section discusses the immutable data structures that we call pseudonyms, how to check if a user is physically present when the pseudonym is used, how audit logs can be formed in a privacy-preserving manner for legal compliance and how authorities can revoke attestations previously given to users.

### 2.4.1 Identification and Pseudonyms

In TCID, identities are implemented as pseudonym data structures, which are stored on the devices of the corresponding subjects. As TCID is Self-Sovereign, a pseudonym can be created by generating a key pair with any of the supported elliptic curves, without permission of a third party. A pseudonym can also be invalidated or removed autonomously by a subject, though its use may leave a permanent record (Section 2.4.3). The pseudonym data structure itself holds a list of credentials that are hidden from verifiers until a user wishes to disclose them. We show an example of a pseudonym presented by a subject to a verifier in Figure 2.3, to guide the explanation of its elements. Figure 2.3 shows that a credential contains a *proof*, an *attribute*, *metadata* and *attestations* when disclosed, and only contains only an *attribute* when undisclosed. When included, these elements make authentication progressively stronger up to serving as a digital equivalent of a physical passport.

Disclosure of credentials is rooted in *proofs*. Each credential has one proof. Typically, a proof will consist of encrypted information (e.g., Pedersen commitments [177] and unlinkable signatures [27]). Given an agreement between the subject and the verifier on a disclosure protocol for a particular proof (normally implicit, but captured explicitly in the metadata of each credential in TCID), some property of the encrypted information can be shown to hold. For example, one may prove the input data for a proof was larger than 21. It is explicitly not allowed to create multiple proofs for the same data as this may lead to information leakage. Building on our previous example, proving one's age is larger than 21 but failing to prove one's age is larger than 22 leaks the exact age of the user.

As shown in Figure 2.3, a subject can add credentials to their pseudonym by attaching a single *attribute* using the hash of the preceding attribute or the public key of the pseudonym, if there is no preceding attribute. Each of these attributes consists of only two hashes: a hash that points to the previous attribute or public key and a hash that points to a proof. By creating an immutable linked list of attributes, we allow for attributes to depend on each other. For example, the validity of a driver's license may critically depend on being enrolled with a valid name, but the name may not always have to be disclosed. By extension, the chain of attributes implies a partial ordering of credentials, which ensures that credentials cannot be mixed-and-matched from multiple pseudonyms. To continue our driver's license example: it is important that the presented valid name is not just any valid name, but the exact valid name that was used to enroll the driver's license.

Metadata will have to support a credential, at least to identify the data disclosure protocol for its contained proof. To connect this metadata to an attribute and its proof, we once again use a hash to point to the respective attribute. Finally, the collected attestations made by issuers point to the metadata of credentials. Through the chain of hashes, this binds an attestation to the metadata of the attested credential, the proof of the credential, and all preceding credentials. Notably, a user cannot change a proof without invalidating all attestations over credentials further down in the chain. This mechanism allows verifiers to detect that the conditions for an issuer to attest to an attribute may have been violated. For example, a name change automatically invalidates a government's attestation of a driver's license that is only valid for that particular name.

Attestations are digital signatures made by issuers and form one of the cornerstones for *Credibility* (Section 2.2). However, these digital signatures are made by public keys and form pseudonyms themselves. We envision a future in which subjects require issuers to use this pseudonym to identify themselves before attesting. For example, an issuer's pseudonym must first prove to work for the government before attesting to a driver's license of a subject. However, which pseudonyms to trust when engaging in business logic is up to users. For the standardization of business logic surrounding pseudonyms with attributes, we refer to the NIST recommendations on Attribute-Based Access Control [103].

### 2.4.2 Proof of user presence

Just like attestation, user authentication is one of the two cornerstones of *Credibility* in SSI systems (Section 2.2). Establishing a communciation session between peers using just public keys is not enough for a passport use case. The user who created a pseudonym should prove they are present when using it, otherwise device theft could lead to identity theft. To this end, a communication session established by the communication substrate can be further strengthened by using credentials for authentication [127]. Such an authenticating credential functions similarly to proving ownership of a public key with a digital signature: a subject shows a verifier a publicly verifiable property of private data they hold.

The contents of authenticating credentials can vary from passwords to biometric checks, and Physically Unclonable Functions (PUFs) [167]. The more authentication credentials are added to a pseudonym, the better its security guarantees will be (potentially even beyond the standards of the NIST [90]). The first deployed prototype of TCID included only closed-source facial recognition with liveness detection.

### 2.4.3 Accountability and Audit Logs

In certain use cases, like border crossing, verifiers are legally required to keep audit logs. Such logs exist for governments to hold individuals *accountable* for their use and misuse of their identities [2, 179]. Audit logs can be formed by the verifier storing the disclosed credentials presented by users [212]. The digital signatures and hashes that constitute a pseudonym make the audit logs immutable, to avoid tampering and to hold legal status. However, logs are not shared publicly. Only certified auditors with the appropriate legal grounds may require verifiers to share an audit log. For example, immigration services may audit border authorities.

To uniquely identify users, TCID offers privacy-preserving credential construction that allows legally required checks by certified auditors. This feature is modular and can be enabled to provide the full range of passport grade use cases. Before engaging in verification of a credential, subjects present verifiers with a credential that is attested to by a certified auditor without proving the credential. This special credential contains an encrypted reference to the natural person that is captured in a government database. Through the auditor maintaining a cryptographic secret link between the attribute of a pseudonym and a central register of all citizens, a pseudonym can be traced back to the enrolled individual by the auditor. The implementation of this optional credential is transparent and limited to specific use cases grounded in appropriate legal frameworks. In the spirit of Self-Sovereignty, both the subject and the verifier may opt not to include or use the credential used for auditing, though it may not be legal to do so.

### 2.4.4 Revocation by Authorities

Several solutions have been proposed to enable revocation by authorities of the subject credentials they have signed [97]. We discuss three of them. The first and fastest solution adds a link to the revocation registry (central server) of the issuer, e.g., a "revocation authority" in IRMA [152]. However, this issuer will have to be online to allow access to the revocation register and forms a specific and trusted third party—which is not Self-Sovereign. The second solution is the complete decentralization of revocations using a shared log,

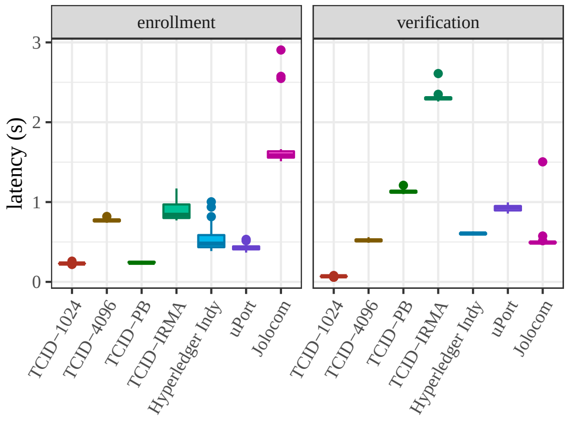


Figure 2.4: Boxplots of the credential enrollment and verification latency for different proof implementations.

e.g., a blockchain—as used by Sovrin [116], using a cryptographic construction to check for revocation in a privacy-preserving manner. However, writing to a shared log takes more time for the revocation to reach finality, can invalidate the incentives the log was built around and has the issue of unbounded growth of the log. Finally, the third solution is to include validity terms into the metadata of credentials, e.g., "epochs of lifetime" in Idemix [27]. As a credential is then only valid for a limited amount of time, the chance of a credential being revoked is related to the duration of its validity. This approach requires frequent re-enrollment of credentials. None of these three available revocation methods fits all use cases, so TCID enables all three to allow adaptation to any use case.

## 2.5 Evaluation

The key system property for the usability of SSI solutions in practical situations is low interaction latency. It would be inconvenient to wait an hour at the airport to have your identity verified and, in fact, electronic border control should take no longer than 30 seconds to be considered usable [128]. In this section we show that the latency of the interactions within the SSI overlay is well within this 30-second limit, finishing consistently within three seconds for all of the implementations of credential proofs we evaluate. In contrast to the three-second latency of these SSI-overlay interactions without anonymization, the latency of sending an SSI-overlay message across a covert channel in the anonymization overlay is shown to be five seconds and up to over 20 seconds. We also attempt to find a metric to serve as a tie-breaker for the different implementations of the SSI-overlay interactions. However, we have not been able to find it, as the latency, CPU usage and network traffic of the implementations are all similar.

### 2.5.1 Experimental Setup

We measure the performance of the enrollment and verification of credentials on a single machine. We compare the credential proof implementations of TCID (see Section 2.4.1) to those available in Hyperledger Indy[3] [197], uPort[4] and Jolocom[5] [92]. These three external solutions have been identified as usable in a study by Bartolomeu [20]. The solutions all offer functionality to create (enroll) and disclose (verify) credentials and we highlight the manner in which they implement this functionality further in Section 2.6. The performance metrics we measure for these two functionalities are latency, CPU usage and network traffic.

The four credential proof implementations of TCID that we measure use three distinct protocols. Two implementations use a Zero-Knowledge Proof protocol that allows proofs over input data with a length of 1024 bits and 4096 bits, which we refer to as the “TCID-1024” and the “TCID-4096” proofs, respectively. One implementation uses a Non-Interactive Zero-Knowledge Range Proof protocol, which we refer to as the “TCID-PB” proof [178], and which allows subjects to prove input data lies in a certain number range. Lasty, we measure TCID’s implementation of the IRMA protocol [5] (a blinded signature scheme based on Idemix [27]), denoted as “TCID-IRMA”.

We have measured the Python implementation of TCID. All experiments are performed on a virtual machine running Ubuntu 19.10, with 4 CPU cores fixed to 3.50 GHz and 16 384 MB of memory. Each of the presented boxplots is constructed from twenty data points. Credential enrollment and verification is measured as-is with default settings, no modifications have been made to the publicly available source code.

### 2.5.2 Latency

In Figure 2.4 we present our measurements of the latency of credential enrollment and verification. These latencies are defined as the time between the subject initiating Flow A of Figure 2.1 and receiving the corresponding attestation, and the time between the verifier initiating Flow B of Figure 2.1 and completing the proof, both without network transfer time. From our latency comparison, we find no clear winner from the evaluated credential implementations. What is gained in enrollment latency is lost in verification latency and vice-versa, with Hyperledger Indy being the most consistent between these two categories (but also not the fastest). The evaluated solutions all finish within 3 seconds, making them usable for, e.g., electronic border control, which should be finished in 30 seconds [128].

The results of Figure 2.4 show that the TCID-1024 proof provides the lowest latency for both enrollment and verification. However, it would be unfair to claim that the TCID-1024 proof is the best solution, as it can only handle up to 128 bytes of information, while the other credential solutions can handle arbitrary-size inputs. However, this does show that choosing the correct disclosure protocol is important to satisfy the need for either low enrollment or low verification latency.

### 2.5.3 CPU Usage

We explore CPU usage as a potential selling-point of disclosure protocols. We measure the CPU usage as the average CPU utilization of the subject and issuer during Flow A of Figure 2.1 and of the subject and the verifier during Flow B of Figure 2.1. As our experiments run on a homogeneous virtualized setup, they serve as a rough estimate for the power

[3]`https://github.com/hyperledger/indy-sdk`
[4]`https://github.com/uport-project/uport-credentials`
[5]`https://github.com/jolocom/jolocom-sdk`

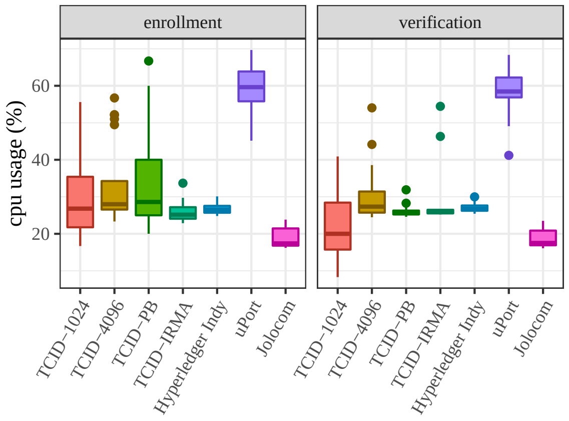


Figure 2.5: Boxplots of the credential enrollment and verification cpu usage for different proof implementations.

consumption of the different Self-Sovereign Identity solutions. As in the previous latency experiment, we measure each credential creation and verification implementation separately. The results are visualized in Figure 2.5. Only uPort offers a surprise, in that it has a higher CPU usage for both enrollment and verification of credentials, seemingly due to its web server. From our experiments, we infer that CPU is not a significant distinguishing factor for current Self-Sovereign Identities.

### 2.5.4 Network Traffic

We now evaluate the network traffic as a result of the various credential implementations. No external tools were used for these measurements: the traffic was measured and aggregated from the data flowing through the sockets used by the measured applications. To contextualize these results, we note that, since the deployment of 3G mobile communication networks, mobile phones are able to transfer megabytes per second [78]. From the results of our experiment, visualized in Figure 2.6, we derive that the transfer rate of several megabytes already amounts to overprovisioning for these implementations.

A notable result is that interactive Zero-Knowledge Proofs generate much more traffic than Non-Interactive Zero-Knowledge Proofs and signature schemes to verify. The worst case, TCID-4096 proof, requires up to 70 kilobytes of data to be transferred (though this is still completely acceptable for devices capable of communicating at several megabytes per second). The low traffic for Non-Interactive Zero-Knowledge Proofs is expected, as it is their design goal [34]. Lastly, the remaining proofs, based on digital signature derivation, also show limited network use.

### 2.5.5 Anonymization

We have shown the feasibility of using different credential implementations with vastly different cryptographic primitives and now offset this to the anonymization component in TCID. We now measure latency as the time it takes to establish a communication channel between real users of Tribler [183], our peer-to-peer file-sharing application that uses

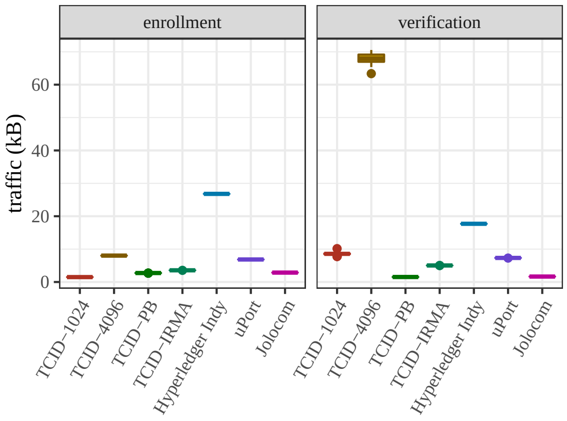


Figure 2.6: Boxplots of the credential enrollment and verification network usage for different proof implementations.

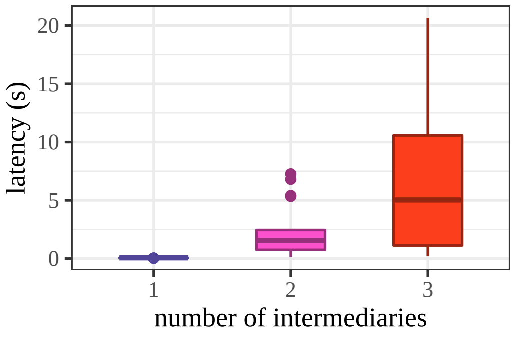


Figure 2.7: Covert channel creation time in TCID.

TCID. We present the creation time for channels of the three lengths default to Tribler: using 1, 2 and 3 intermediary peers. The more intermediaries a channel contains, the harder it will be for an adversary to decrypt or block any particular channel.

We have visualized the time it takes to create a channel for different intermediary counts in Figure 2.7, which shows that the parameterization of the anonymization component is very important for the total latency of the SSI-overlay interactions in TCID. Using 1-intermediary anonymization, the anonymization latency is hardly significant in comparison to the latency of the credential implementations (Figure 2.4). When using two intermediaries, the anonymization latency is comparable to that of the credential implementations. Finally, by using three intermediaries for anonymization, the latency is dominant compared to that of the credential implementations. This result shows that the choice of the credential proof implementation is indeed insignificant for the latency of the credential data flows when compared to the anonymization component, using the standard of three intermediaries for covert channel construction.

## 2.6 Related Work

A promising start has been made in the field to try and make sense of all the available information disclosure technology, its mapping to the W3C DID standard [188], and its interaction with blockchains [122, 162] and distributed storage like IPFS [79]. A critical view of the system components of Self-Sovereign Identity, however, is missing in academia and is only briefly discussed in industry whitepapers, the most notable analysis coming from SelfKey [83]. Many SSI solutions claim they are anonymous, but this is erroneous as they only achieve pseudonymity and do not address device fingerprinting [217]. We postulate that anonymity and identity disclosure protocols are orthogonal but can be combined to achieve pseudonymity with selective disclosure of information. We have shown how external cryptographic protocols, like that of IRMA, can be made interoperable with our solution.

**Early work using central servers.** Early work focuses on supplying users with the ability to tie their claims to a token-based identity, as exemplified by Microsoft Passport [42], OAuth [47], FIDO [143] and OpenID [187]. At the time, the use-case of identification of users was to reidentify with the same central server—but these works laid the groundwork for cryptography, the terminology (e.g., "claims" and "relying parties"), and the principles of Self-Sovereign Identity. The advantage of using central servers is that the metrics considered by this paper of latency, CPU usage, and network traffic, are all superior as opposed to other solutions. However, the disadvantage is that these solutions require third parties and thereby they are not Self-Sovereign.

**Unlinkable signature-based disclosure schemes.** Many mainstream SSI solutions are based on signature derivation (most notably Idemix's CL signatures [41] and JSON Web Tokens [76]), possibly storing revocations on blockchains [1]. These signatures bind the key of the identity holder to the disclosed data and usually allow derivation of a new signature that is also valid for the same data (which is claimed to make this data *unlinkable*, though fingerprinting makes users completely linkable as we have previously discussed). Examples of these systems are IRMA [5], Jolocom and uPort [92], ClaimChain [126], HyperLedger Indy and Sovrin [197]. The advantage of signatures schemes is that they have less latency and bandwidth usage than classical Zero-Knowledge Proof protocols. However, the disadvantage is that these schemes require disclosure of data instead of only proving a property of that data.

**ZKP-based identities.** Identification through Zero-Knowledge protocols has been proposed decades ago [66] and these protocols have been rediscovered as a key component of Self-Sovereign Identities [11]. However, Zero-Knowledge Proofs over data and their security implications for SSI systems remain scarcely documented and rarely implemented by academia [36], though this type of system is widely proposed by industry, with `https://github.com/peacekeeper/blockchain-identity` listing 134 Self-Sovereign Identity solutions based on blockchain. The advantage of Zero-Knowledge Proofs is that they allow a subject to only prove a property of the data that they wish to allow verification of. However, the disadvantage is that classical (interactive) Zero-Knowledge Proofs require more latency and bandwidth. As shown by the results of this paper, more recent "non-interactive" ZKPs mitigate these weaknesses.

## 2.7 Conclusion

Research on Self-Sovereign Identities is narrowly focused on cryptographic data disclosure protocols. However, our analysis shows that these protocols are not the only critical consideration for Self-Sovereign Identity solutions. Such an isolated approach ignores performance and security concerns at the networking layer of SSI systems. We have presented our implementation of a system that addresses these concerns. Furthermore, we have shown the feasibility of a passport grade Self-Sovereign Identity through our design and implementation of TCID. Our Self-Sovereign Identity solution of TCID disallows network traffic correlation, while still enabling passport-grade interactions with acceptable latency.

# 3 Web3 Sybil Avoidance Using Network Latency

*Web3 is emerging as the new Internet-interaction model that facilitates direct collaboration between strangers without a need for prior trust between network participants and without central authorities. However, one of its shortcomings is the lack of a defense mechanism against the ability of a single user to generate a surplus of identities, known as the Sybil attack. Web3 has a Sybil attack problem because it uses peer sampling to establish connections between users. We evaluate the promising but under-explored direction of Sybil avoidance using network latency measurements, according to which two identities with equal latencies are suspected to be operated from the same node, and thus are likely Sybils. Network latency measurements have two desirable properties: they are only malleable by attackers by adding latency, and they do not require any trust between network participants.*

*In this chapter we present our basic SybilSys mechanism that avoids Sybil attackers using only network latency measurements if attackers do not actively exploit their malleability. We present an enhanced version of SybilSys that protects against targeted attacks using a variant of the flow correlation attack, which we name TrafficJamTrigger. We show how the message flows of Round-Trip Time measurements can be used to expose attack patterns and we propose and evaluate six classifiers to recognize these patterns. Our experiments show, through both emulation and real-world deployment, that enhanced SybilSys can serve a fundamental role for Web3, effectively establishing connections to real users even in the face of networks consisting of 99% Sybils.*

## 3.1 Introduction

The Web3 ecosystem, the decentralized web, is the most recent step in 50 years of continuous evolution of the Internet. This evolution is tightly connected to the history of the birth, adoption, and governance of *distributed protocols* [176]. Web3 is the emerging decentralized alternative to the currently dominant Web2, which is centrally governed [231]. In contrast to the governance of Web2, the governance model of Web3 could be described as a "collaboration of strangers," characterized by a lack of prior trust between users. Therefore, Web3 could be best defined as a *collectively maintained public infrastructure* [231]. To create this infrastructure, users interact with other users, which they find through peer sampling. However, attackers can interfere with this sampling by using Sybils, a surplus of fake identities [70]. We present a peer sampling mechanism that avoids connections to Sybils using only network latency.

A primary driver of Web3 is the use of transparency to become "trustless," the absence of a need for trust in a system [48]. To this end, Web3 typically consists of peer-to-peer technology (e.g., BitTorrent) with a distributed ledger (e.g., Bitcoin) and the social practice of sharing (e.g., GitHub). Web3 is based on open protocols, open source, open collaboration, and freely re-usable components. Web3 can provide critical infrastructure for identity, money, markets, data, and AI. Even security improvements are handled openly: Web3 is aiming to alter the incentives for exploiting bugs, thereby improving security. Incentive alignment is achieved by offering payment (e.g., Bitcoins) through bug bounty programs [100]. For example, the community has tried to address frontrunning, a method to exploit pending trading actions [56], collaboratively [24, 189, 222].

Scalability to billions of users is a problematic requirement for Web3. Scaling issues arise as nodes within large-scale distributed systems are unable to keep track of all other participating nodes. The membership table of nodes may become unmanageable due to its size and due to the need for frequent updates because of node arrivals and departures. Early work used this approach, leading to considerable synchronization costs at scale [29]. Instead of depending on a single global (data) structure, full scalability is only achieved with distributed infrastructure. However, the typical Web2 approach of using predefined critical infrastructure, like the cloud or privately owned servers, violates the trustless nature of Web3. Therefore, Web3 applications employ different means to tackle their scalability issue.

A fundamental mechanism at the heart of Web3 applications, which addresses the scalability issue of node discovery, is the *peer-sampling service* [108] as it appears for example in Bitcoin and BitTorrent. To this end, Web3 requires the creation of an overlay network from network participants' computers without relying on centrally governed servers ("zero-server"), and resilience against fraud in the form of fake identities. A peer sampling service is required by all three known styles of gossip protocols used by Web3 applications, i.e., rumor-mongering protocols, anti-entropy protocols, and aggregation protocols [28]. Rumor-mongering protocols use flooding to spread information at a fixed rate to all nodes. Anti-entropy protocols connect to nodes randomly, comparing information and reconciling differences. Aggregation protocols conduct pair-wise information exchanges and combine values to arrive at a system-wide value. Therefore, all gossip-based Web3 designs are susceptible to attacks on the peer sampling mechanism, e.g., a Sybil attack.

Any node can use a peer sampling service to discover other nodes that are randomly sampled from all nodes in the system, these nodes are the node's initial *neighbors*. Further connections can be made to even more peers through connections to the neighbors of a node's neighbors. Thereby, the concept of neighbors exploits locality as an alternative to a centrally maintained membership table. However, if a bad-faith actor creates many fake identities (Sybils) to serve as neighbors, real identities may perpetually be introduced to more Sybils. This systemic oppression of nodes hinders their ability to maintain public infrastructure and, thereby, allows for their abuse. For example, an attacker can deprive nodes of the latest information in a cryptocurrency, which leads them to think that they will receive currency though this won't happen (also known as "double-spending" [113]).

Using network latency to avoid Sybils is a unique opportunity for Web3. In Web2 systems, centralized platforms are used to bring users in contact with each other, and the indirection of traffic that goes with it makes it difficult, if not impossible, to measure network latency between users. In contrast, in the Web3 ecosystem, nodes send messages to each other over the Internet without intermediaries, enabling a new method of defense through latency. Furthermore, these latency measurements are a particularly attractive option as they do not require a gossip protocol, avoiding a chicken and egg situation.

This chapter proposes a *Sybil-avoiding peer sampling* service to enable the collective infrastructure of Web3 that provides nodes with non-Sybil nodes without a-priori trust. To achieve this, we counter the Sybil attack solely using measured network distances, which does not require a centrally governed entity. The core contribution of our work is **a mechanism for dependable, zero-server, trustless network topology creation in spite of Sybils** called *SybilSys*. On a more granular level our contributions are as follows:

1. We argue that Sybil avoidance must be addressed in the peer sampling mechanism (Section 3.2) and we define an adversary model (Section 3.3).

2. We rationalize, implement and verify light-weight Sybil-avoiding peer sampling using only network latency measurements (Section 3.4).

3. We show how message flows of latency measurements can be used to detect Sybils over the Internet (Section 3.5).

4. We create a version of our peer sampling mechanism that specifically counters attacks against our initial mechanism (Section 3.6), based on message flows, and we verify it with real users (Section 3.7). We derive that this mechanism is trustless, predominantly samples honest nodes and cannot be efficiently attacked.

## 3.2 Problem Description

The core problem of our work is breaking the dependency between requiring Sybil-free gossip protocols and using gossiped information to eliminate Sybils. Gossip protocols do not guarantee delivery of information in a network filled with Sybils and Sybils cannot be identified without the information gossiped through these protocols. In avoiding gossip protocols, Sybil avoidance becomes a key problem for the peer sampling service. A straightforward gossip-free approach, like relying on unique IP-addresses or `traceroute` to filter Sybils is still open to attacks, as recently shown through the Erebus attack [226].

**Adversarial behavior is often considered to be out of scope** for peer sampling research (e.g., with assumptions that "each participant is limited to one identity" [138]). For decades, analytical studies proposed peer sampling services and tried to determine if they actually lead to uniform sampling or become unstable—while ignoring security [107, 108, 115, 138].

Bitcoin is an example of a system that seemingly resists Sybils, but can still suffer attacks on the network layer [245]. Only on its application layer does Bitcoin provide a solution to Sybil attacks by crafting a high computational-cost barrier for tampering through a mathematical puzzle known as proof-of-work. Regrettably, alternatives to proof-of-work—or in general proof-of-something—systems with both lower computational costs and preservation of decentralization remain elusive [96]. Nevertheless, attacks on Bitcoin using Sybils are effective. Using Sybils to cause (even temporary) network partitioning is monetarily profitable for attackers [201] and leads to long-term advantages [193]. An interesting ongoing experiment is Blockstack, a chain which externalized protection against Sybils by relying on Bitcoin [3]. Blockstack rewards prior mining in Bitcoin and uses a verifiable random function for leader election, presumably leading to undesirable rich-get-richer dynamics.

**Peer sampling is inherently fragile and sensitive** to fraud and faults. Protocols with mass adoption typically encounter problems such as malicious behavior, accidental deviations from protocol specifications, malfunctioning nodes, correlated failures, network partitioning, data corruption, and dissemination of incorrect information. The state-of-the-art remains fragile. Especially Sybils that follow specialized attack strategies (beyond simply existing) go undetected in social networks [6, 86, 109] and are able to influence applications based on (federated) Artificial Intelligence [84, 85, 120].

An example of using a "score function" on social interactions, to protect against Sybils, is found in GossipSub [232]. With various parameters, each node attempts to keep track of the honesty of its neighbors. Their "application-specific score" imports information from the application level for Sybil avoidance and their "IP Address Collocation Factor" parameter is very effective against simple Sybil attacks. The global impact of this approach is still limited because it is not possible to trust your neighbors to accurately report their neighbors' Internet addresses (i.e., this violates our trustless requirement). Every address obtained by an attacker can be re-used to create unique Sybils in GossipSub. Services that offer unique IP addresses, *rotating proxy infrastructure* rental, can dramatically decrease the monetary cost of an attack on this system.

**Using network latency is a promising approach** to the Sybil problem, due to the grounding in the laws of physics. Network latency can only be tweaked by attackers by artificially increasing it, *never* by decreasing it [21]. No software-based attack can alter the lower bound on latency between two nodes, as the underlying shortest physical link between nodes remains the same.

Network latency for Sybil avoidance is used in related fields outside of epidemic protocols and peer sampling. There have been prior attempts to make use of network latency for the detection and avoidance of Sybils [21, 204]. Early work uses triangulation to pinpoint physical coordinates within wireless sensor networks. Unfortunately, triangulation has proven to fail with realistic network conditions [130, 131, 153] due to lack of latency symmetry for bidirectional wireless links and invalidity of the *triangle inequality* from Euclidean geometry. Moreover, triangulation requires trust in the localization data provided by strangers, violating the trustless requirement.

## 3.3 Adversary model

Web3 nodes aim to cooperatively maintain public infrastructure in the presence of attackers that employ an overpowering number of identities on a limited set of physical resources. Violation of public infrastructure by attackers can be detected by sharing immutable Web3 data. For example, the immutable history of Bitcoin exposes any attempted violations by attackers through its chain of blocks (connected using cryptographic hashes). Therefore, a connection to only a single other honest node is required for any honest node to detect violations. Conversely, the goal of the attackers is to disallow this sharing of data, to not have their violations detected. To distinguish attackers, two types of entities are considered:

- *honest nodes* control only one logical node and one identity per physical machine in the network.
- *attackers* create and control multiple identities per physical machine in the network. Any identity controlled by an attacker entity is a *Sybil identity*.

The focus of this work is to counter the cheapest attack on peer sampling services: the generation of Sybil identities on a single node. Within this scope, the goal of attackers is centered around operating a surplus of Sybil identities on a single node in an attempt to isolate a cluster of honest nodes from the rest of the network (also known as an Eclipse attack [205]). Therefore, we consider a Sybil-avoiding peer sampling mechanism to be successful if it associates only a single identity with a single node. Our goal does not include the more complex case of countering Sybil identities operated by a single attacker from multiple nodes, e.g., by using a botnet, as existing work has already been shown to work when attackers are limited to one identity per IP address [135, 138].

The peer sampling mechanism that we present in this work depends on latency measurements between nodes and we now discuss how it can be attacked by adversaries. In particular, in the remainder of this section we discuss how the possible attacks are addressed in our work. Our analysis considers attacks on the hardware, software, and sessions, required to perform a latency measurement between nodes over the Internet.

Attacks on the ability of distributed communications networks, like the Internet, to pass messages between nodes have been studied for over 60 years [19]. However, to this day, messages that are sent over the Internet are still not guaranteed to arrive at their destination. We mitigate all cases of latency measurement messages not arriving (e.g., due to attacks, offline nodes, and network disruption or outage) by periodically contacting neighbors and disconnecting from them if they do not respond. Of course, a network that lacks even the basic means of communication between nodes trivially forgoes the need of a Sybil defense, like the one we propose. Therefore, we make the following assumptions of basic meaningful connectivity:

- **Assumption 1:** Nodes are addressable over the Internet (e.g., through IP) and connectable based on their address.

- **Assumption 2:** Honest nodes are online long enough to engage in the peer sampling process.
- **Assumption 3:** Attackers do not control all hardware on the routing path over the Internet between them and an honest node. In Section 3.5.1 we discuss attackers that control part of the hardware along the routing path.

It may be possible for adversaries to replay messages that are sent between other nodes. Our approach to counter replay attacks is to make use of unique pairs of a request and response message (explained in Section 3.4). These pairs are made unique through the use of a random nonce that is added to the request and response message. However, if the random nonce can be predicted, an attacker can send a latency response before any request was made and potentially lower its measured latency. Therefore, we assume that the random nonce generation used provides the following guarantee:

- **Assumption 4:** The request and response messages that are sent between nodes contain a unique nonce that cannot be predicted by an attacker.

The manner in which our basic mechanism establishes node identities is through their latency. However, honest nodes that share the same physical location are all seen as belonging to the same identity and, thereby, as attackers. In this case, a single honest node would still be sampled but all others in the same location would be ignored, which is inefficient. In Section 3.5, we further enhance our identity establishment through estimation of Internet routing paths between nodes and how they overlap. However, in the case that there is no overlap at all, e.g., using wormhole routing, our enhanced mechanism may erroneously sample an attacker. Both of these cases can be addressed by assuming that honest nodes have typical consumer connections:

- **Assumption 5:** Honest nodes are geographically dispersed.
- **Assumption 6:** Honest nodes use a single network adapter and a single last-mile connection to their physical node.

Adversaries may attempt to abuse the connections that are opened between nodes by sending a large amount of data. In this work, we do not provide or implement any protection against these Denial of Service attacks. However, we note that our peer sampling mechanism has highly regular communication patterns, which allows it to be coupled to most of the many available Denial of Service protection mechanisms [151].

## 3.4 Basic SybilSys: basal Sybil-avoiding peer sampling

We first describe a basic mechanism called Basic SybilSys, which leverages round-trip time (RTT) measurements to avoid connecting to multiple identities from the same physical location, i.e., Sybils. This mechanism does not address the problem that comes with using network latency for Sybil avoidance: we assume attackers do not delay messages in order to try to defeat Basic SybilSys (altering latency measurements, see Section 3.2). Enhanced SybilSys drops this assumption (Section 3.5).

In every node, Basic SybilSys operates three lists of nodes: *discovered*, *connected*, and *accepted*. Node identities are moved between these lists in three steps. During each of these steps a node identity may be ignored or a connection to it may be dropped. When an identity ends up in the accepted list, it is made available by SybilSys to a given application.

In the first step, Basic SybilSys is executed by a node to request new nodes from its initial accepted neighbors. These initial neighbors are untrusted and typically either defined by a user or sampled from a specialized server, known as a rendezvous server [82]. A rendezvous server does not act as a peer in the peer-to-peer network, it is not a critical component that causes centralization, and it does not need to be trusted. Basic SybilSys can be implemented using a single request message and a single response message. The request message indicates a node wants to learn about new nodes and the response contains the Internet addresses of one or more nodes. New addresses in the responses are added to the *discovered node* list.

In the second step, the latency toward newly discovered nodes is measured. Repeatedly (using a fixed-interval schedule), a single identity is randomly sampled from the discovered node list and an outgoing connection attempt is made. During the connection establishment we measure the response time. We use the time between the transmission of the request and the response as the RTT, which includes the remote message processing time in a measurement. A node's identity moves into the *connected* node list once a connection is established (to the physical node over the Internet) and its RTT is known.

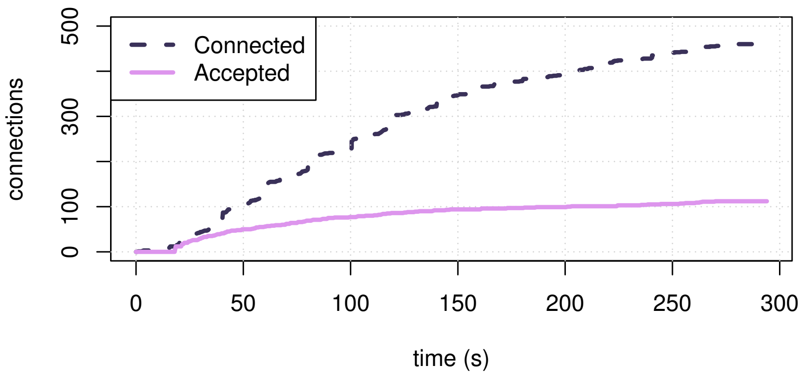
Finally, the essential step of Sybil filtering is carried out. The measured latencies are compared to those of all nodes on the accepted list. Non-Sybil identities are promoted from the connected node list to the *accepted* node list, while suspected Sybil connections are simply dropped. Basic SybilSys reasons that two identities with the same RTT reside on the same node and are therefore Sybils. We only accept a new node if we do not yet have a node on the accepted list with the same RTT. This filtering ensures the *latency diversity* property: all nodes using Basic SybilSys are surrounded by nodes with unique latencies.

The performance of Basic SybilSys relies on the filtering step, which ensures that no two (or more) nodes are accepted with a similar latency. This dramatically limits Sybil attacks to one identity per node (which is the same number as an honest node has). However, SybilSys may also filter identities that are not Sybils if they are operated from different nodes that happen to have the same latency.

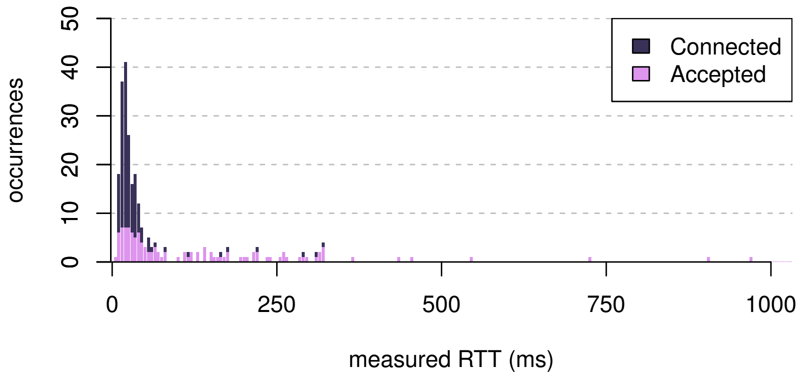
### 3.4.1 Latency and diversity

Our algorithm uses a simple, practical, and economical method to measure latency and (latency) diversity. We use a simple probe packet and response to measure Internet latency, an application-level ping approach. Application-level pings require no special Operating System permissions or specialized hardware. For example, in contrast, sending ICMP messages may require super user permissions. Due to its ease of adoption, this approach fits our aim of mass adoption by billions of users. Repeated probing of a single target increases the measurement accuracy [21, 130, 131, 153, 204].

The latency measurements of two target nodes are considered (latency) *diverse* if their difference is larger than some threshold parameter $\Delta$. Choosing a $\Delta$ depends on the network's properties: it will not cover the natural variance in latency measurements if it is



(a)



(b)

Figure 3.1: The number of connected and accepted nodes without Sybils (a) and a histogram of their latencies at the time of 300$s$ (b).

too small, which could lead to a single node being seen as distinct from itself, but if Δ is too big then only very few nodes will end up being accepted. Our implementation uses the median of five latency probes as the latency measurement to avoid transient congestion and packet loss [55]. If more probes have been performed, the last five are used to form the latency measurement value.

Other, more advanced, implementations of latency diversity are also possible. For example, the Kolmogorov–Smirnov test can be used to test latency similarity [129], giving the distance between two series of latency measurements. It would also be possible to modulate Δ in our approach by deriving the variance of measurements from already-accepted nodes.

### 3.4.2 Real-world deployment

We now deploy Basic SybilSys over the Internet for a Web3 application to demonstrate to what extent our peer sampling service avoids Sybils. The distributed system we utilize is called Tribler [183], which is our research vehicle for cooperative systems research, and which has been installed by 2 million users[1]. Tribler uses peer sampling and gossip protocols to offer a video streaming service.

Our setup consists of a single "measuring" node that uses Basic SybilSys to sample nodes from the Tribler network, which still use the default peer sampling service of Tribler. We evaluate one configuration with and one without Sybils, and we extend our setup with a second "attacker" node that operates Sybils in the former configuration. For the configuration without Sybils we do not include an "attacker" node in our setup. The experiment terminates after 300 seconds.

When used for the configuration with Sybils, the "attacker" node operates 100 Sybils in such a way that they do not hinder real users in the Tribler network. To obtain a presence of 99% Sybils, we modify the mechanism that provides an initial sample of nodes to be non-random. With a probability of 99% each identity in this sample is a Sybil, operated from our "attacker" node, instead of being randomly sampled from the Tribler network. Each Sybil only introduces one of the other Sybils from the "attacker" node for Basic SybilSys' discovery step response messages (instead of a random sample of its accepted nodes). Lastly, we modify our response messages to not forward Sybils to the regular users from the Tribler network.

We present (a) the numbers of connected and accepted nodes over time, for which we (b) verify the diversity property (at a time of $300s$ into the experiment). These results are presented for our configuration without Sybils in Figure 3.1 and for a configuration with a Sybil presence of 99% in Figure 3.2.

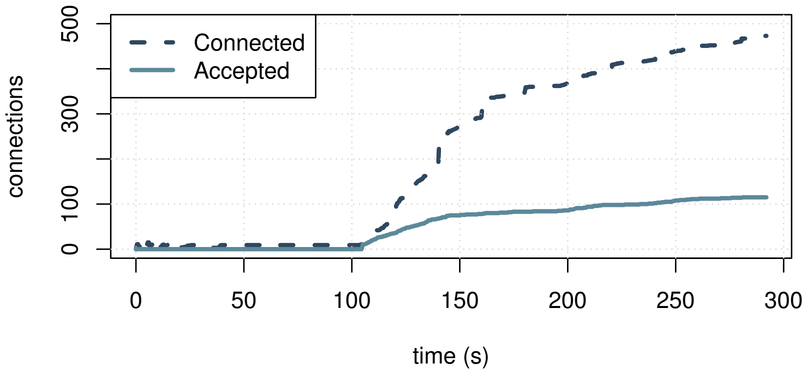
The horizontal axes of Figure 3.1a and Figure 3.2a represent the elapsed time since the start of the experiment, and their vertical axes give the number of identities that are connected or accepted (all accepted identities are also counted as connected). From a network crawl we estimate that the Tribler network contains roughly 50 000 nodes, meaning that approximately one hundredth of our network is reached in about five minutes.

The histograms of Figure 3.1b and Figure 3.2b show the number of occurrences of measured latencies, in intervals of $5ms$. The chosen time range of the histograms (0 through 1000 milliseconds) omits one occurrence in Figure 3.1b (at $1295ms$) and two occurrences in Figure 3.2b (at $1655ms$ and $1730ms$).

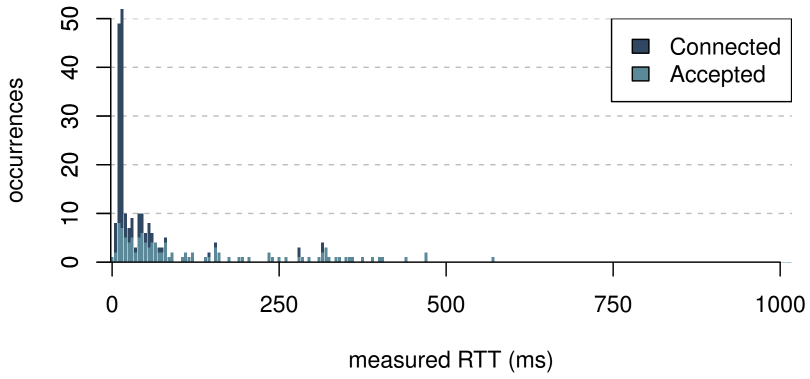
Our results show that Basic SybilSys correctly provides latency diversity and eventually finds honest nodes. The implementation repeatedly traverses its most-recently added neighbors starting from 10 initially accepted nodes, which is visible in Figure 3.1b and Figure 3.2b: no single latency interval contains more than 10 accepted nodes. Furthermore, it can be observed that the accepted nodes are spread out over the RTT intervals, which is the latency diversity we seek. The second property of Basic SybilSys is that it eventually finds and accepts nodes in the network. However, the more Sybils are present in the network, the longer it will take to find latency-diverse nodes.

Our results highlight two implementation details of Basic SybilSys. First, in Figure 3.1a and Figure 3.2a sudden increases in the number of connections are visible (e.g., at 140

[1]`https://www.tribler.org/`



(a)



(b)

Figure 3.2: The number of connected and accepted nodes with 99% Sybils (a) and a histogram of their latencies at the time of 300*s* (b). At the end of the experiment 12 out of 112 accepted nodes turn out to be Sybils.

and 160*s* in Figure 3.2a), which occur due to the spam prevention for requesting random samples from the deployed rendezvous servers. The second detail is that the Basic SybilSys implementation uses latency-diverse random walks from latency-diverse starting points. As long as SybilSys cannot find 10 latency-diverse nodes to start from, it continues to repeatedly drop connections and request new random samples. This leads to a longer period of low numbers of connections as the number of Sybils grows.

## 3.5 Enhanced SybilSys: hardening to attacks

The Basic SybilSys algorithm presented in the previous sections is vulnerable to the *latency measurement interference attack*, in which an attacker may wait before answering a network latency measurement message. Furthermore, Basic SybilSys does not account for changes in latency (e.g., due to mobile nodes that physically travel) nor a node's initial accepted neighbors completely consisting of Sybils (causing a node to be perpetually introduced to only more Sybils). The issue of latency updates (and any other updates in routing path) is addressed in Section 3.5.1. Countering attackers that manage to occupy the initial set of accepted neighbors is discussed in Section 3.6.4.

### 3.5.1 Detecting measurement interference

We first consider the case when two messages flows are sent over the Internet to a single "measuring" node. If the two flows are of "sufficient size" and they are each sent by a different node, they will join in a network interface queue at some point along their ways. To create flows of sufficient size, they should be adjusted to account for the latency of the nodes that send them. For example, if two nodes both send only one message at the same time and their latencies to a shared queue are 50*ms* and 100*ms*, the queue has likely already forwarded the first message within the roughly 50*ms* before the second message arrives. By sending more messages, over a longer time period, it becomes more likely that messages of the two message flows occupy the same queue at the same time. Thereby, additional delay for messages occurs when they have to wait for messages of another flow.

We now consider the second case, when messages follow the same physical path during their flow, i.e., they originate from the same physical machine and are Sybils. In this case, there is no deferred joining of these flows' physical paths and the additional delay will not occur. Therefore, the lack of additional delay can be used as an indicator for identities being Sybils. Secondarily, if message flows do follow the same path (i.e., to Sybils on the same node), we expect higher delays to be measured over the entire series of measurement responses as all messages already occupy the same queues.

In both of the cases we presented, we assume that flows follow a fixed path over the Internet. However, routing over the Internet does not necessarily follow fixed paths: any two messages sent to the same IP address may take radically different paths over the Internet. However, even though the paths are not explicitly defined by the application layer, paths over the Internet have enough temporal stability to detect joining message flows: Internet paths can remain stable for days or months, and wireless paths are expected to last for multiple seconds, if not minutes [185].

Enhanced SybilSys leverages the temporal stability of Internet routing paths and the fact that RTT measurements capture queuing delays. By sending a message flow of latency probes (i.e., RTT measurement messages) to each of two identities, a joining of both flows is exposed through the RTT measurement responses. The pattern that emerges from joining flows is a sudden (short) increase in measured RTT. Our measurement strategy, called *TrafficJamTrigger*, creates these latency probe flows and adjusts their size such that the probes' response messages are likely to join.

TrafficJamTrigger is applied to a message flow between a node sending latency probes and one of two identities that are simultaneously being measured. An identity is classified as an attacker if its flow does not show signs of the two flows joining. Enhanced SybilSys uses this classification to further filter nodes beyond using the latency diversity of Basic SybilSys. This intrinsically avoids identities whose message flows are routed through a single proxy (e.g., the Erebus attack that reroutes message flows through a single autonomous system [226]).

Our TrafficJamTrigger mechanism is grounded in existing work on the *flow correlation* attack on Tor [119]. The goal of the flow correlation attack is for an attacker to detect whether a single honest node receives messages from two message flows, for each of which the honest node uses a different identity. Thereby, the attacker wishes to defeat Tor's anonymization (that allows the honest node to communicate using the two identities without exposing the Internet address of its node). An attacker uses the correlation between a sudden increase in RTT and messages being sent over both message flows to infer that both flows lead to the same node. This attack has been shown to work for both TCP and UDP [246]. TrafficJamTrigger uses the concept of the flow correlation attack to determine whether identities are Sybils or not by creating RTT measurement message flows to those identities and looking for an RTT increase.

### 3.5.2 Verifying the flow join pattern and its impact on RTT

We now verify our claim that flow correlation can be used over the Internet through two small experiment setups. Our first setup shows the increase in the measured RTT as the number of identities on a single node increases when messages share a single path. Our second setup shows the patterns in RTT measurements of joining flows and of a single flow, focusing on the manner in which measurements increase over time instead of the increase's intensity. Both setups use TrafficJamTrigger.

For both setups, we implement TrafficJamTrigger by sending bursts of 20 messages to each identity that is being measured. The identities are sent such bursts in descending order of their "initial RTT", which is measured for each identity in isolation before using TrafficJamTrigger, in order to avoid message flow joins during these initial measurements.

Our first setup consists of two nodes connected over a LAN, with two intermediate routers. Whereas this setup uses real routing hardware, it does not use the Internet, which we evaluate in Section 3.7. We use one node to measure the RTTs and the other to respond to RTT measurement requests. The measuring node uses TrafficJamTrigger in an attempt to detect an increase in the RTT of the identities operated from the responding node. To show to what extent the number of identities influences a latency measurement interference attack, we test three configurations with fixed delays of $0ms$, $100ms$, and $500ms$ that determine how long every Sybil waits before responding to a latency probe. Assuming the Sybils to be numbered starting from zero, Sybil $i$ uses a delay of $i \cdot d$, with $d$ the basic delay of the configuration. The delays correspond to an optimal attack, where an attacker only requires a single Sybil to take over every unique latency slot in the accepted node list.

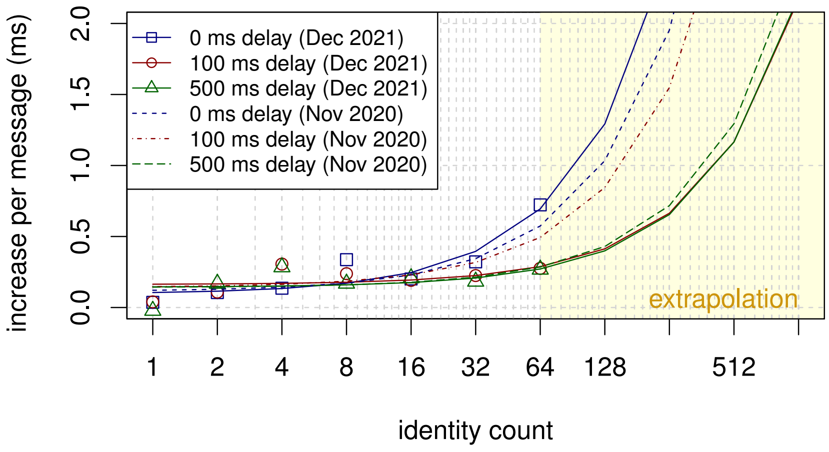
Our results go up to the point where our measuring node starts filling its (network socket) receiving buffer, which we find at 64 identities. When the receiving buffer of the measuring node starts filling, the increase in RTT jumps into the order of dozens of milliseconds due to the limits of our hardware. In turn, if the jump is used to detect Sybils, the use of limited hardware would directly result in trivial Sybil detection. Therefore, to predict for more powerful hardware, we provide an extrapolation of our results for higher identity counts instead.

In Figure 3.3 we present the increase in measured RTT versus the number of identities on the responding node. We calculate linear regression models for each set of measurements and plot them as lines in our graphs (these would be straight lines if the horizontal axis would be linear instead of on a log scale). To show programming language (in)dependence, both a Python (Figure 3.3a) and a Java (Figure 3.3b) implementation are evaluated. To show the repeatability of this experiment we include the results from both 2020 and 2021 for the same Python implementation.

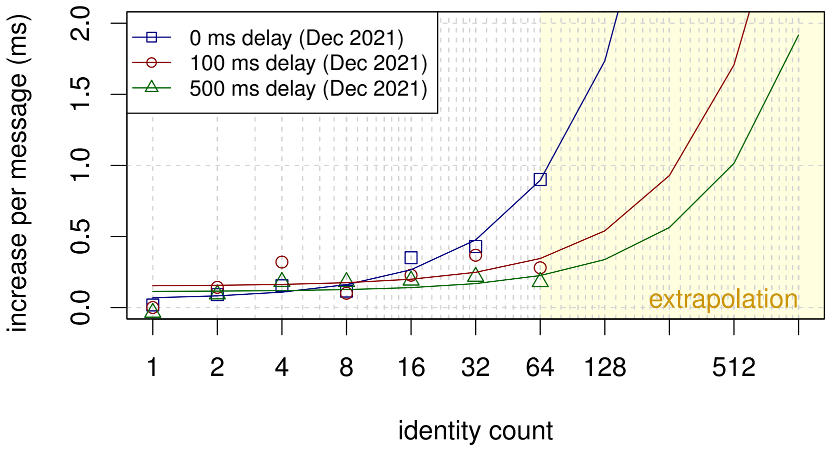
From the results of Figure 3.3 we observe an increase in the measured RTT as the number of identities deployed on the responding node increases. The increase in RTT is even observed when Sybils wait longer before responding to probes. However, the RTT increase due to identities sharing a path should disappear when Sybils increase their waiting time for probe responses. Therefore, the shown increase using $500ms$, which is almost double the highest measured increase of $256.6ms$, must be due to other factors (e.g., identity management inefficiencies in our implementation). Whereas these other factors help in the detection of Sybils, we note that the $500ms$ RTT increase could feasibly be eliminated with specialized attacker hardware. Nevertheless, when message flows do share a path ($0ms$ delay), the RTT clearly increases with the number of identities.

Our second setup, to show the pattern that emerges when flows join, consists of a measuring node that is connected to 50 honest nodes sampled from our Tribler network and two separate nodes that each run 25 Sybils. Of the two Sybil nodes, one is at a distance of $300m$ on the same (TU Delft) campus as the measuring node, and the other is $48km$ away, hosted in a datacenter. None of the 52 nodes that are connected to the measuring node are on the same LAN as the measuring node. We present the typical pattern of RTTs resulting from each of the bursts' response messages when two honest nodes are being measured (a) and when two Sybils are being measured (b). The latter situation, with two Sybils, corresponds to that of our first setup when two identities share the same node (see Figure 3.3).

To highlight the difference between honest nodes and Sybils, Figure 3.4 shows two typical bursts, which we found through qualitative analysis of several dozens of measurement pairs. The first obvious difference is the steady increase in measured RTT for the honest node versus its absence for the Sybil. However, this steady increase (roughly $1.6ms$ between messages) simply shows the application's buffer being filled by other messages. This is not necessarily a result of applying TrafficJamTrigger: the jump in measured RTT due to filling buffers has been previously described related work, like that on packet chirping [125]. Instead, the pattern is characterized by the sudden jump in measured RTT for the honest nodes (at timestamp $1.74ms$, after the sixth measurement), which does not oc-

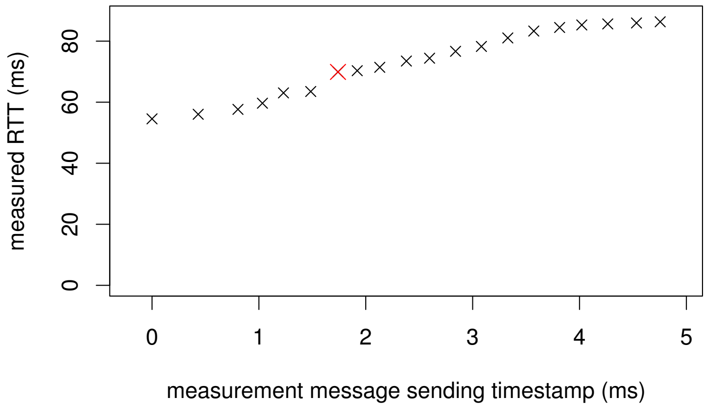


(a) Python implementation (measured in 2020 and 2021)

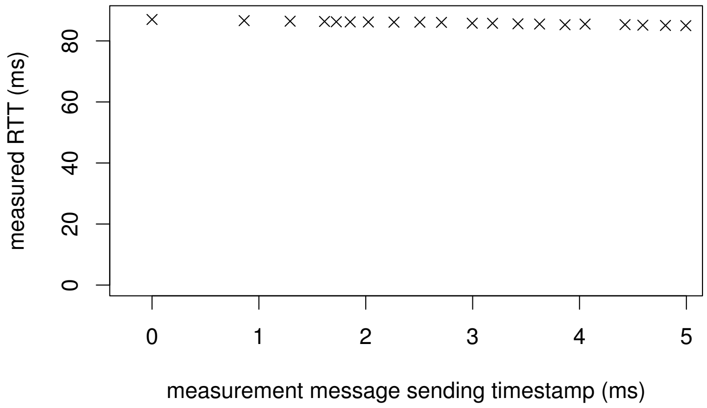


(b) Java implementation

Figure 3.3: The measured latency increase versus the number of identities on a node with TrafficJamTrigger in comparison to the measured latency without using TrafficJamTrigger, with linear regression lines to provide extrapolation. The regression lines of 100$ms$ and 500$ms$ for December 2021 in Figure 3.3a overlap.



(a)



(b)

Figure 3.4: The measured RTT per message sent in a single 20-message burst versus the message sending timestamp (starting from 0) for an honest node (a) and a Sybil (b). The sudden jump in measured RTT for the honest node at timestamp 1.74$ms$ has been made slightly larger and colored red.

cur for the Sybils.

The effectiveness of classification using the pattern that we observed (i.e., whether it is statistically significant) depends on the manner in which the pattern is detected. Therefore, in the remainder of this work we define six binary classifiers to detect the RTT jump pattern and we evaluate them using real users.

### 3.5.3 Properties of Enhanced SybilSys

Enhanced SybilSys does not aim to prevent attackers from creating Sybils. An attacker can temporarily block communication between honest nodes using Sybils, known as an Eclipse attack [205]. Nevertheless, we summarily present the five properties of Enhanced SybilSys that greatly increase the effort needed for attackers to perform a successful Sybil attack, based on our findings of Section 3.4, and Section 3.5.

**Trustless self-reinforcing Sybil avoidance** is guaranteed through a bias toward honest nodes when introducing neighbors. Nodes keep connections open to those neighbors that they have accepted as non-Sybil through their latency-diversity property. Therefore, honest nodes are more likely to increasingly introduce and be introduced to other honest nodes without requiring a trust relationship with the introducing node.

**Attackers must create Sybils with unique RTTs** to be able to suppress honest nodes (governed by the threshold parameter $\Delta$). This suppression implies that Sybils must wholly occupy the accepted list of a node, which is filtered using latency diversity. By estimating what RTTs will be measured to honest nodes, an attacker can interfere with latency measurements to have its Sybils appear to match the estimations. This causes the honest nodes to be filtered out when they are eventually found. However, every diverse RTT of an honest node requires a new Sybil group. For example, to obtain a Sybil presence in a network that contains two honest nodes that differ in latency more than $\Delta$, an attacker must create two groups of Sybils that have RTTs within $\Delta$ of each honest node (doubling the number of required Sybils by using latency diversity).

**Attackers require low latency to targets** to compete with honest nodes. To perform an Eclipse attack, Sybils must have an overpowering presence within a latency of $\Delta$ of any honest node. However, the only attack on RTT measurements is to introduce additional delay, as they cannot be decreased by an attacker. Therefore, if an honest node exists that has a measured RTT that differs more than $\Delta$ from the node that operates Sybils, it is impossible to suppress this honest node using Sybils. As a consequence, honest nodes that are in close physical proximity cannot be attacked without an attacker buying hardware close to the targets. For a network-wide attack an attacker would have to buy hardware close to all honest nodes in the network, which makes a network-wide attack possible, but infeasible.

**Open networks (Web3) cannot be efficiently attacked** as the network participants and their latencies are not known beforehand. Web3 technology is characterized by participants from all over the world that continuously join and leave the system. In contrast, in networks with known participants, it is feasible for an attacker to measure and predict latencies between other network participants that are online for extended periods. In order for an attacker to disrupt connectivity in such a network, it must occupy each unique latency-diverse slot in the accepted node list. Therefore, all possible latencies must be preempted and have a majority of Sybils to suppress newly joining honest nodes with a unique latency. For example, with a latency measurement timeout of 5 seconds and a diversity threshold of $\Delta = 5ms$, a majority of Sybils has to be present for each of these 1000 distinct latencies. With a sufficient number of honest nodes (that may occupy any of these 1000 slots), the required number of Sybils has to be increased thousand-fold to suppress honest nodes in comparison to an equal Sybil attack without latency diversity.

**Low Sybil counts must be used per node** to decrease the chance of a connection being dropped. As the number of identities per node grows, it becomes increasingly likely that a connection to any of these identities is dropped. Therefore, a significant Sybil attack must be spread out over multiple nodes.

## 3.6 Enhanced SybilSys: implementation

In this section we discuss the implementation details of Enhanced SybilSys: the data structure that stores node identities, how flow joining is brought about, how it is detected when flows do join, maintaining and refreshing the node storage data structure, and the parameterization of Enhanced SybilSys. To implement flow recognition and data structure maintenance multiple options are presented, which are evaluated in Section 3.7.

### 3.6.1 The peer discovery tree

Every newly joining node in the system creates a peer discovery tree as its environment to operate in with itself as the root. Using the peer sampling process, it first selects an initial set of latency-diverse nodes, which we call the *bootstrap set*, as its children in this tree. Each of these children then helps the creator by building a (linear) branch of the tree consisting of latency-diverse nodes, starting with itself, of a pre-specified maximum length. In fact, they initiate a random walk, reporting back to the creator the identities of the nodes sampled along the branch and discarding those that are found by the creator to be not latency-diverse with any of the previous nodes in the branch. In the latter case, another node is sampled. As a consequence of this construction, the nodes in the bootstrap set and the nodes in each of the branches separately are latency diverse, but nodes in different branches may not be. An example of a possible peer discovery tree is given in Figure 3.5.

TrafficJamTrigger must be applied to all pairs of nodes that require latency diversity. However, because neighborhoods are of limited size, this is not a problematic requirement. Furthermore, the pairs that require diversity are not all pairs of nodes in the peer discovery tree. For example, given the ten initially sampled latency-diverse nodes that grow branches of four diverse child nodes (as in Section 3.4.2), only 105 pairs need to be probed: 45 pairs for the ten initial nodes and 6 pairs for each of the 10 branches of four nodes. We continuously apply TrafficJamTrigger to random pairs in the peer discovery tree after the initial filtering using latency diversity. We discuss how a tree is updated when node pairs classify as Sybils in Section 3.6.4.

### 3.6.2 Message flow joins and bursts

TrafficJamTrigger sends bursts of RTT measurement messages to two identities to decide whether they share a node. However, it may not be sufficient to send two flows of measurement requests to two identities one after the other. For example, a node close to a

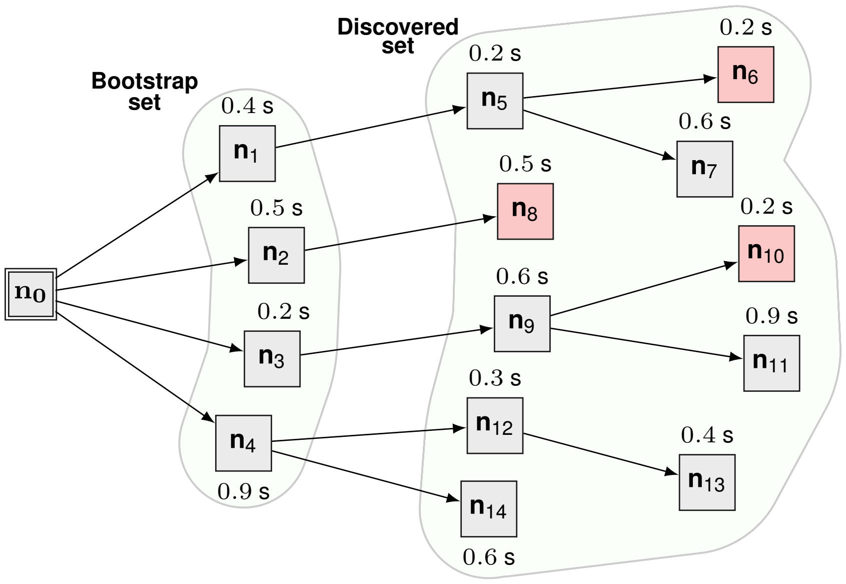


Figure 3.5: Example of a peer discovery tree for a node $n_0$ with a maximum depth of 3 and $\Delta = 0.1$ s. Nodes with dark (red) shading violate the latency diversity (i.e., their difference in latency is strictly smaller than $\Delta$).

measuring node may already have responded to all latency probes before a node that is further away has even started responding. This example would not lead to two message flows joining in a queue and therefore lead to the honest nodes being labeled as Sybils. To this end, our probe sending strategy consists of alternating short bursts of messages to the two identities instead.

When measuring two identities, TrafficJamTrigger sends a burst of messages as fast as possible. A burst starts with messages to the identity with the higher RTT, which is immediately succeeded by messages to the other identity. The intent of this “slowest first” ordering is to account for the case when an honest node has a low latency and responds before the measuring node even finishes sending out its latency probes to the node with the higher RTT. In practice, we use the heuristic of alternatingly sending the same number of bursts (of 20 messages each) to both identities without additional delay, one for every 200 ms of RTT of the slower identity (for example, two bursts to each identity if the highest RTT is 234 ms). Despite it being a heuristic, we show that this strategy works well enough to detect attackers in Section 3.7, and we leave further optimization to future work.

### 3.6.3 Recognizing message flow joins

Any binary classifier that is able to detect a sudden jump in a burst of measured RTTs, shown in Section 3.5.2, for the node with the highest initial RTT (Section 3.6.2) should suffice for TrafficJamTrigger. However, the steady increase or decrease of measured RTTs makes it more difficult to detect this jump. Queues may be in the process of being filled or emptied due to other messages (Section 3.5.2). For example, an RTT jump of $0.2ms$

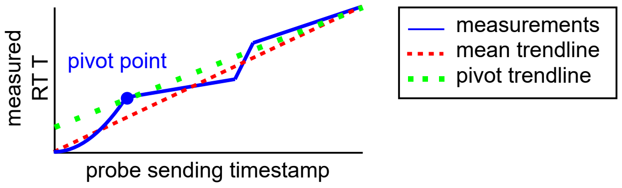


Figure 3.6: Example construction of the pivot point, mean trendline, and pivot trendline, for an imaginary measurement curve.

between two measurements over a steady increase of $1ms$ leads to an RTT difference of $1.2ms$ between them, while the jump is actually of equal magnitude compared to a difference of $0.2ms$ when there is no steady increase.

A linear interpolation, of two specifically selected measurements of the same burst, is constructed to compensate for the increase (or decrease) during a burst. We call the linear interpolation a *trendline* and we explain the two ways in which we construct it shortly. A burst's measurements are recalculated as their difference to the value of their corresponding trendline at the time of sending a measurement probe.

We consider two types of trendlines: the *mean trendline* and the *pivot trendline*. The mean trendline starts with the first measurement of a TrafficJamTrigger burst and ends with its last measurement. The pivot trendline passes through the *pivot point* instead. The pivot point intuitively corresponds to the point where a queue has finished filling and streams have not joined yet (the fifth point in Figure 3.4a, just before the RTT jump between the sixth and the seventh point). In other words, the first point in the burst after which the increase of the RTT does not increase. We formally define this pivot point in a burst as the last measurement $m_t$ (where $t > 2$) in the sequence of timestamps starting at $t = 1$ for which it holds that $m_u - m_{u-1} > m_{u-1} - m_{u-2}$ for all $3 \leq u \leq t$. As a visual reference, the two trendlines and the pivot point are shown for an imaginary measurement curve in Figure 3.6.

We now present six classifiers for detecting the RTT-jump pattern, given in Table 3.1. The first three classifiers are based on the nearness of the RTT measurements to their trendline. We express this *nearness* as the mean squared error (MSE) of RTT measurements being smaller than a parameter $\varepsilon$. Two identities are classified as Sybils if the TrafficJamTrigger measurements of the slower node have an MSE smaller than $\varepsilon$. The difference between the three MSE-based classifiers lies in the RTT measurements that are included in the computation of the MSE: the *MSE* classifier uses all measurements, the *MSE pre-pivot* classifier uses all measurements up to the pivot point, and the *MSE post-pivot* uses only the measurements after the pivot point.

Two further classifiers are based on the shape of the measurement progression, taking no parameters. The *log-like* classifier is based on whether the majority of the measurements are higher than the trendline. The *wave-like* classifier decides if nodes are Sybils if the TrafficJamTrigger measurements do not cross the trendline exactly once, going from values smaller than the trendline to values larger than those of the trendline.

Our final classifier is the *baseline increase* classifier, which exploits the latency increase phenomenon mentioned in Section 3.5.2. This classifier deems two identities to be Sybils if

Table 3.1: Binary classifiers that detect artificial delay using the RTT burst measurements of the node with the highest initial RTT.

| **Classifier** | **Attacker classification condition** |
|---|---|
| MSE | MSE of all RTT measurements is smaller than $\varepsilon$. |
| MSE pre-pivot | MSE of RTT measurements before the *pivot* is smaller than $\varepsilon$. |
| MSE post-pivot | MSE of RTT measurements after the *pivot* is smaller than $\varepsilon$. |
| Log-like | Most of the measurements are above the trendline. |
| Wave-like | RTT measurement progression does not go from below to above the trendline. |
| Baseline increase | The average of the measurements is a given percentage higher than the initial RTT measurement before the burst. |

the identity with the highest RTT exceeds a given increase in average RTT during a burst compared to its initial RTT. The results of Figure 3.3 show that this method increases in efficacy as the number of Sybils on a device grows.

### 3.6.4 On accepted node removal

Enhanced SybilSys may require the removal of accepted nodes. Nodes may go offline, they may physically move (and no longer be latency-diverse), and TrafficJamTrigger may classify identity pairs as Sybils. Secondarily, nodes should be continuously removed (and resampled) to avoid the situation in which a successful attack could ever *permanently* hold all of the spots in a node's peer discovery tree if Sybils go undetected. In all of these cases a node's peer discovery tree has to be updated.

We consider three (binary) aspects that can be combined for a total of eight possible configurations for node removal from the peer discovery tree. First, we explore what node pairs to select for classification. We distinguish the selection of node pairs formed by a node in the bootstrap set combined with a node in its respective branch (the *local* pairs) and the selection of a pair from any two nodes in the entire peer discovery tree (*all* pairs). Secondly, we investigate what node pairs to remove for continuous resampling, the churn strategy. We consider periodically removing the node pair with the highest (*worst*) classifier score (though it may not be above a classifier's threshold), and removing a *random* node pair. Lastly, we inspect whether to either *remove* all following nodes in a branch when a node is removed or to only remove an intermediate node in a branch and *keep* its descendants by linking its child to its parent. In Section 3.6.5 we evaluate all eight configurations.

### 3.6.5 Configuration for deployment

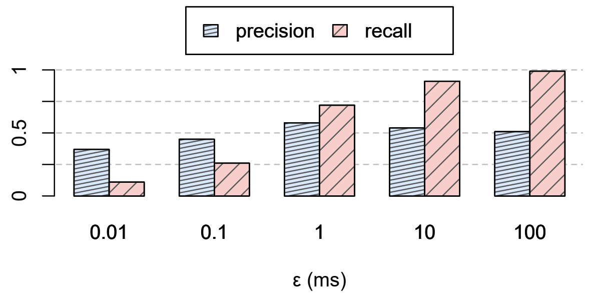
We configure Enhanced SybilSys to seamlessly integrate with our existing Tribler network in order to deploy and evaluate it with real users of Tribler (Section 3.7). We adopt the parameterization of Tribler's peer sampling mechanism as much as possible, namely the total number of nodes and the periodicity of the peer discovery algorithm. First, we derive the peer discovery tree's bootstrap set size and its maximum branch length (see Section 3.6.1) from the total number of nodes. The remainder of this section presents small-scale experiments to configure parameters that cannot be derived from Tribler: the parameterization of the classifiers (given in Section 3.6.3) and the configuration of Enhanced SybilSys' node removal (Section 3.6.4).

We configure Enhanced SybilSys' peer discovery tree based on the peer sampling mechanism of Tribler. We adopt the total number of 20 as the target size of the peer discovery tree. Given the target number of 20 nodes, we parameterize SybilSys with 10 branches that grow to a branch length of 4 (if not stopped at 20 nodes, the tree would grow to a size of 40 nodes).

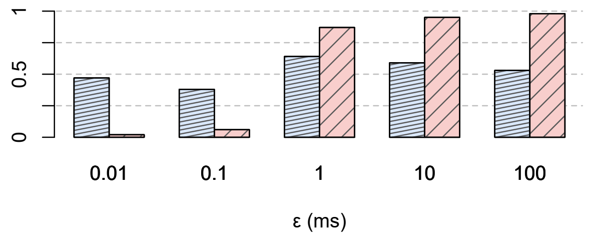
The experiments to parameterize the classifiers emphasize classifier quality over the speed of node discovery. The setup of our configuration experiments uses two nodes that each operate 25 Sybil identities and we sample 50 honest nodes from the Tribler network for each run (equal to the setup of Section 3.5.2). We run 60-second experiments using $\Delta = 0.05s$, a bootstrap set size of 6, a maximum peer discovery tree branch length of 4 and a total tree size of 20 for each node. With less branches than in actual deployment (which grows 10 branches from 10 boostrap nodes), the quality of classifiers is stressed more.

We parameterize our classifiers (also known as "training" in Machine Learning) to provide a balance between recall and precision. On the one hand, the recall of any binary classifier corresponds to the chance that whatever entity is being classified, Sybils in our case, is classified as that entity. As Enhanced SybilSys uses classifiers to mark nodes for removal, the recall corresponds to the chance that a Sybil is (eventually) removed. The recall should therefore be as high as possible. On the other hand, the precision of a binary classifier corresponds to the fraction of entities that are correctly classified over all the entities. A precision lower than 0.5 causes Enhanced SybilSys to mostly mark honest nodes for removal. Inversely, because nodes share their neighbors with each other, a precision of over 0.5 means that nodes predominantly resample nodes from a set of nodes that is more likely to contain honest nodes. Therefore, if the precision is at least over 0.5, this resampling ensures that honest nodes are predominantly connected to other honest nodes.

We use our setup to configure the MSE-based classifiers and the baseline increase classifier. We show the impact of different values of $\varepsilon$ on the precision and recall for both the mean trendline in Figure 3.7a and the pivot trendline in Figure 3.7b for the MSE classifier. The results of Figure 3.7 show a trade-off between the classifier's recall and its precision. We pick a value that lies between the collapse of the precision and the collapse of the recall, with a precision of at least 0.5. Therefore, for both trendlines, we configure the MSE classifier to use a value of $\varepsilon = 10ms$. The pre-pivot and post-pivot parameterization have a similar collapse and we again choose values between their collapses, which we find at $\varepsilon = 10ms$ and $\varepsilon = 0.01ms$ respectively. In the same fashion, for the baseline increase classifier we pick a 20% increase in latency to classify Sybils.



(a)



(b)

Figure 3.7: The precision and recall for different values of $\varepsilon$ for the MSE classifier using the mean trendline (a) and the pivot trendline (b).

Table 3.2: The number of honest nodes found using the node removal strategies of Section 3.6.4 averaged over 30 experiment runs (higher is better).

| **Node Removal Strategy** | | | **Number of Honest Nodes** | | | |
|---|---|---|---|---|---|---|
| **Pairs** | **Churn** | **Descendant** | **Mean** | **Median** | **Min** | **Max** |
| *local* | *worst* | *remove* | 9.18 | 9 | 4 | 17 |
| *local* | *worst* | *keep* | 8.88 | 9 | 5 | 14 |
| *local* | *random* | *remove* | 9.39 | 9 | 5 | 15 |
| *local* | *random* | *keep* | 8.56 | 9 | 5 | 12 |
| *all* | *worst* | *remove* | 9.57 | 10 | 5 | 16 |
| *all* | *worst* | *keep* | 8.91 | 9 | 5 | 13 |
| *all* | *random* | *remove* | 9.95 | 10 | 6 | 17 |
| *all* | *random* | *keep* | **10.24** | 10 | 6 | 16 |

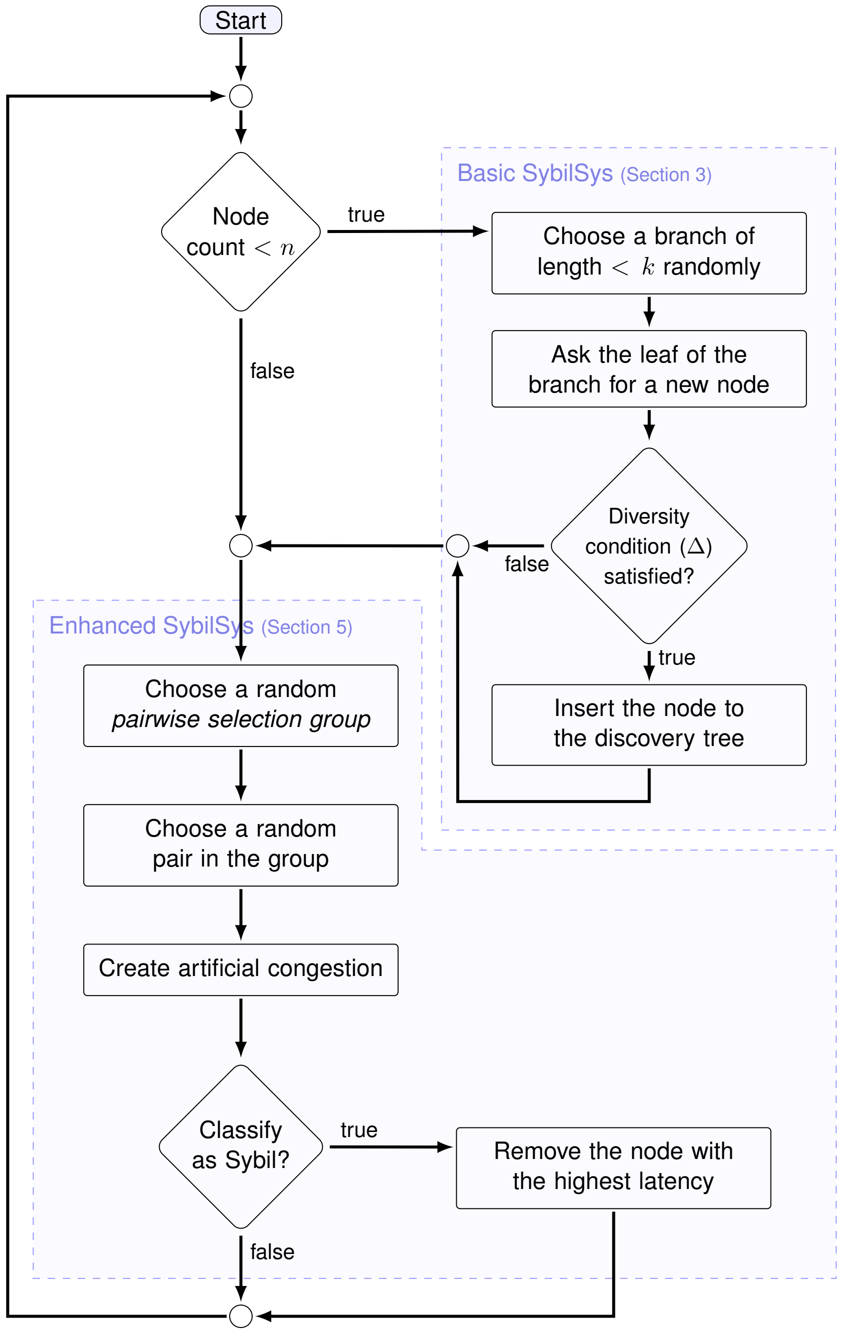

Start
Node count $< n$
true
false
Basic SybilSys (Section 3)
Choose a branch of length $< k$ randomly
Ask the leaf of the branch for a new node
Diversity condition ($\Delta$) satisfied?
false
true
Insert the node to the discovery tree
Enhanced SybilSys (Section 5)
Choose a random *pairwise selection group*
Choose a random pair in the group
Create artificial congestion
Classify as Sybil?
true
false
Remove the node with the highest latency

Figure 3.8: Activity diagram of our Enhanced SybilSys for the requested total node count $n$, and the maximum branch length $k$.

In order to configure the node removal of Enhanced SybilSys, we evaluate the strategies of Section 3.6.4 using our experiment setup with the MSE classifier. We present the number of honest nodes each node-removal strategy yields in Table 3.2. Firstly, the result of applying classification over all pairs in the peer discovery tree is similar to that of classification within each branch and the bootstrap set, though classifying all pairs performs slightly better. Secondly, enforcing churn through continuously removing the "most Sybil" 2-tuple also does not lead to any big differences in honest node counts compared to random node removal, performing worse in all cases but one. Lastly, we find that—in all but one case—removing all descendants in a branch leads to more honest node retention than only removing the particular Sybil node and linking its child and parent node. Our results show that the configuration that retains the most honest nodes consists of applying classification over all node pairs, periodically removing random nodes and linking descendants to the parents of removed nodes in a branch.

### 3.6.6 Overview of Enhanced Sybilsys

Omitting the passive latency measurements (Section 3.4), bootstrap set construction (Section 3.6.1), and peer maintenance (Section 3.6.4), the activity diagram of Figure 3.8 serves as a summary of our proposed algorithm. The periodicity of this algorithm consists of each node in the network running it once every 0.5 seconds (the default periodicity of Tribler's peer sampling).

## 3.7 Enhanced SybilSys: evaluation

We now evaluate both our flow joining detection classifiers and our complete Enhanced SybilSys algorithm (configured for deployment as described in Section 3.6.5) through two experiments. First, we construct several data sets of TrafficJamTrigger measurements to which we apply the six classifiers defined in Section 3.6.3. Second, we evaluate Enhanced SybilSys using the best-performing classifier to mimic a best-effort defense by honest nodes. Finally, based on the experimental results, we estimate the cost of attacking Enhanced SybilSys.

Both experiments of this section use real users of our Tribler network and we introduce Sybils into the network that actively attack Enhanced SybilSys using the latency measurement interference attack. We denote the node that uses Enhanced SybilSys in order to filter out Sybils as the "measuring node" and we set its latency diversity threshold $\Delta = 5ms$, based on the real-world deployment results of Section 3.4.2.

### 3.7.1 RTT-based classifiers evaluation

To evaluate our binary classifiers (Section 3.6.3), we create four data sets consisting of TrafficJamTrigger measurements, one for pairs of honest nodes and three for pairs of Sybils. We create more sets for Sybils to evaluate the impact of running Sybils on three different numbers of nodes, which should make them harder to detect (Section 3.5.2). The setup for all four of these sets consists of a measuring node that connects to all identities in a given set and performs TrafficJamTrigger measurements for all pairs of identities it is connected to. The data sets only contain the measurement bursts which are used for classification, those of the node with the highest RTT for each pair (Section 3.6.3). We do not perform

Table 3.3: Overview of the data sets used for the evaluation of the classifiers.

| Data set | Nodes | Identities per node | Measured pairs |
|---|---|---|---|
| honest | 500 | 1 | 124750 |
| Sybil (1) | 1 | 100 | 4950 |
| Sybil (2) | 2 | 50 | 4950 |
| Sybil (3) | 4 | 25 | 4950 |

TrafficJamTrigger measurements, and we do not create datasets, for pairs consisting of an honest node and a Sybil. The reason is that we deem it unethical to Sybil attack unsuspecting users of Tribler. This limitation may skew our results. For reproducibility and further research, we have made our anonymized data sets publicly available [213].

Our honest set was created using the nodes of real Tribler users found with a network crawler that assumed the role of the measuring node. These users were running real workloads and had real network congestion during their measurement. All users had unique IP addresses and remained online for the duration of our experiment. Over the course of three days, our network crawler had eventually connected to 500 nodes and the experiment was started. TrafficJamTrigger measurements were performed by the crawler for all possible pairs of these 500 nodes, leading to 124 750 unique sequences of measurement bursts. The mean size of a sequence of bursts was 99 messages, and the median was 39 messages (to reiterate, we send 20 messages per 200 ms of RTT). Due to network effects, there were messages that did not receive a response. Out of 124 750 sequences of bursts, only 7 414 had *all* of their individual messages responded to. Despite these network effects, Section 3.7.2 shows that a classifier can be used for effective Sybil filtering.

The Sybil sets capture artificially delayed Sybil attackers. We establish three data sets for different attacker setups, where the attacker nodes are in different physical locations: (1) a single node operating 100 Sybil identities, (2) two nodes operating 50 Sybil identities each, and (3) four nodes operating 25 Sybil identities each. Each setup operates in its own overlay network partition in order to keep Sybil attackers from communicating with Tribler users. TrafficJamTrigger measurements were performed for all possible Sybil pairs, including those running on different nodes. All three setups have 100 identities, each resulting data set includes 4 950 TrafficJamTrigger sequences of measurement bursts per setup. For reference, the summary of our data sets is given in Table 3.3.

We evaluate the binary classifiers (parameterized using the training set of Section 3.6.5) using a test set comprised of our honest set and our Sybils sets. However, our honest set and Sybils sets are not of equal sizes (124 750 bursts sequences for honest nodes and three sets of 4 950 bursts sequences for Sybils) and this *imbalance* would skew the precision and recall metrics [123]. To make up for the size differences, we use statistical bootstrapping, which consists of random resampling with replacement from the (smaller) Sybil sets to match the size of the (largest) honest set. Other methods can be used to equalize the set sizes (e.g., taking a subset of the honest set), though their estimation errors should all converge for a sufficiently-large smaller set [170]. Opinions on what constitutes a

Table 3.4: Sybil classifier performance for both trendline types.

| | **Mean** | | **Pivot** | |
|---|---|---|---|---|
| **Classifier** | ***Precision*** | ***Recall*** | ***Precision*** | ***Recall*** |
| Random | 0.5021 | 0.4982 | 0.4984 | 0.4974 |
| MSE | 0.5372 | 0.9119 | 0.5895 | 0.9479 |
| MSE pre-pivot | 0.5319 | **0.9139** | 0.5860 | **0.9544** |
| MSE post-pivot | 0.4428 | 0.5877 | 0.1661 | 0.1037 |
| Log-like | 0.5457 | 0.4683 | **0.6577** | 0.9069 |
| Wave-like | **0.6283** | 0.4761 | 0.6544 | 0.9071 |
| Baseline increase | 0.5951 | 0.7350 | 0.5965 | 0.7392 |

sufficiently-large set differ, but typically this is in the order of hundreds of data points (our smallest data set has 4 950).

The precision and recall, calculated using our test set, for all of the RTT classifiers are presented in Table 3.4 for both trendline methods (see Section 3.6.3). For reference, we include the *random* classifier, which simply classifies an identity as Sybil with a 50% chance. We only include this reference to show that our statistical bootstrapping method is correctly implemented and leads to 50% precision and 50% recall.

Our results indicate that a jump in latency measurements is best detected by the MSE pre-pivot classifier, which classifies measurements while intermediate queues are filling before the pivot point. In contrast, the MSE post-pivot classifier shows both very poor precision and very poor recall, showing it is very difficult to detect Sybils after the pivot point using the Mean Squared Error, i.e., after queues are filled. This lack of detection aligns with the intuition that there can be no jump (up) in RTT measurements if all RTT probes already suffer the maximum queuing time. The MSE classifier, which considers both the pre-pivot and post-pivot measurements, achieves similar results to the pre-pivot classifier in spite of the inclusion of the post-pivot measurements. The log-like and wave-like classifiers have a relatively high precision of over 0.5 and are also usable with Enhanced SybilSys. These two classifiers benefit greatly from the pivot trendline, nearly doubling in recall value. The baseline increase classifier does not use a trendline and we have simply evaluated it twice, with similar results. All in all, all classifiers except the MSE post-pivot classifier are able to effectively filter Sybils with Enhanced SybilSys. Nevertheless, as recall is the decisive metric for convergence of the set of accepted nodes to a set of honest nodes, we use the MSE pre-pivot classifier for our final algorithm in the following evaluation.

## 3.7.2 Real-world evaluation of Enhanced SybilSys

Our second evaluation consists of combining the best performing Sybil classifier, MSE pre-pivot, with our Enhanced SybilSys peer discovery mechanism and evaluating it in a network with Sybil nodes. Again, every honest node is running a peer-to-peer file sharing client, leading to real-world congestion effects. The aim of our evaluation is to show that honest nodes are still able to find each other in these extremely challenging conditions. We explore two aspects of how an honest node finds other honest nodes: (1) the time it takes until it finds the first other honest node, and (2) how the number of honest nodes it has found progresses over time.

For both aspects, we create a setup with in total 100 identities (both honest and Sybils). Our setup consists of a measuring node that is connected to a number of honest nodes from the Tribler network and up to four attacker nodes that run 25 Sybil identities each. To create networks with a given Sybil fraction, we randomly sample identities from these honest nodes and from the attackers' Sybil identities. For example, a network with a Sybil fraction of 0.75 contains 25 random honest nodes and 75 random Sybils, each of which may run on any of the four attacker nodes.

We determine the number of Sybils per node based on the results shown in previous sections. In Section 3.5.2 (the second setup) shows the relationship between the number of identities per node and how easily Sybils are detected using an initial RTT measurement, corresponding to the baseline increase classifier. As the latency of all identities on a single node grows with the number of identities on the node, so does the efficacy of our baseline increase classifier (see Section 3.6.3). In other words, creating too many Sybils per node disallows an attacker to effectively influence its measured latencies (by defenders) and this would thereby trivially increase the effectiveness of Enhanced SybilSys. Therefore, we use the setup of Section 3.6.5 to derive that the largest number of Sybils per node that does not increase the recall of our baseline increase classifier. We find a number of 25 Sybil identities to be the maximum (reaching a recall of 0.92 at 50 Sybil identities, equaling the MSE pre-pivot classifier).

We present the time until half of the honest (measuring) nodes have found another honest node. The reason for using this metric is that when there is at least one connection to another honest node, peer sampling is successful for Web3 applications (see Section 3.3). For brevity, we call the neighborhood of an honest node (excluding the measuring node itself) that includes at least one honest node *robust*. The results we present are averaged over 20 runs of at least 10 minutes.

The results of our first exploration given in Figure 3.9 show that up to a Sybil fraction of 0.95 (with only 5 honest nodes for 100 identities), the neighborhoods of honest nodes are robust nearly instantly. This robustness is grounded in the chance of sampling at least one honest node with 20 samples, the target number of nodes (see Section 3.6.5), from a set of 100 identities that include 5 honest nodes (essentially binomial sampling). Our results underline that, up to a Sybils fraction of 0.95, Sybils can be trivially avoided by opening a sufficiently large number of connections. Nevertheless, Enhanced SybilSys is useful when the initial random sample of nodes does not contain an honest node and when the number of connections cannot be scaled up to match the Sybil presence in a network (e.g., due to security or hardware constraints). For the Sybil fractions of 0.97 and 0.99, an honest node using Enhanced SybilSys is expected to find another honest node after 275 and 335 seconds, respectively, with a granularity of five seconds.

We now zoom in on how the number of honest nodes found progresses over time for a setup with a fraction of Sybils of 0.99, which is the most challenging case for 100 identities. It forces Enhanced SybilSys to filter out latency-diverse Sybils using TrafficJamTrigger in order to find the single honest node. For comparison, we use two benchmarks,

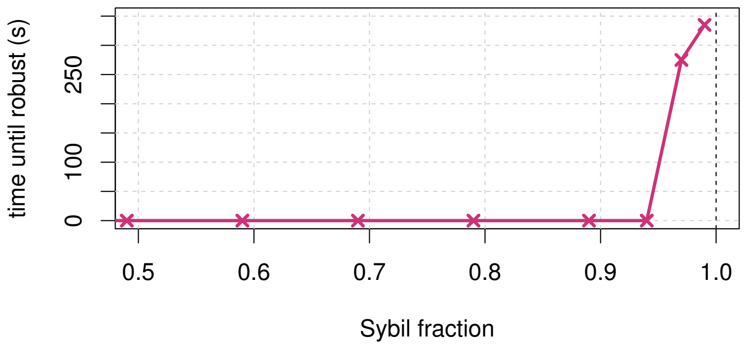


Figure 3.9: Time until half of the honest nodes has found another honest node, i.e., until their neighborhoods are robust, as the Sybil fraction increases.

the theoretical upper bound on the number of honest nodes that can be found (e.g., by a perfect classifier of precision and recall of 1.0) and the expected number of honest nodes in a random sample (e.g., from a rendezvous server). Any Sybil-avoiding peer sampling mechanism is upper bounded by the chance to discover an honest node through iterative random sampling, which follows a geometric distribution. In contrast, a random and uniformly selected sample of 20 nodes (the configured number of target nodes) leads to 0.2 honest nodes on average for a network that contains one honest node out of 100 nodes.

The results depicted in Figure 3.10 show that Enhanced SybilSys does not reach the theoretical maximum honest node count. However, our results do show that the average number of connected honest nodes reaches over 0.5 after 6 minutes, at which point the majority of experiment runs have found and retained the singular honest node. Therefore, Enhanced SybilSys outperforms the benchmark of random sampling and is a viable solution to form a robust overlay network for this particular setup.

### 3.7.3 On the cost of attacking

We evaluate the monetary cost of delaying the discovery of honest nodes by six minutes (the time we found in Section 3.7.2 for a network to become robust). To put this into perspective, we use the statistics of Tribler, which is estimated to have 50 000 concurrent users. In Section 3.7.2, it is shown that the ratio of the number of Sybil identities to the number of honest nodes should be at least 99:1 to delay honest node discovery by six minutes. Therefore, using Section 3.7.2's setup of one node per 25 identities, a successful attack is required to use more than 198 000 nodes in different physical locations. Even if attackers use free Wi-Fi and old \$20 Raspberry Pi's, a Sybil attack on this network would cost just under 4 million dollars. At this cost, attacks other than the Sybil attack are economically more viable to block communication. For example, cheaper attacks are a man-in-the-middle attack on all users (about 50 000 × 20\$, excluding labor costs) and rented virtual private servers (given a \$10 price tag per server, leading to 198 000 × 10\$).

Using the same assumptions as for the estimate using our own network, we estimate

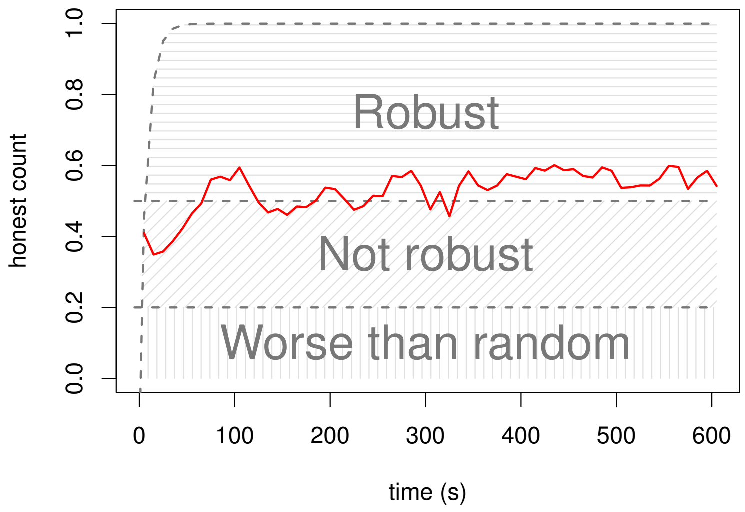


Figure 3.10: The number of honest nodes found by the measuring node over time (the dark, red, line) in a 99% Sybil network. A random sample of 20 nodes leads to 0.2 found honest nodes, a robust neighborhood contains more than 0.5 honest nodes, and a perfect classifier upper bounds the number of found honest nodes with a geometric distribution.

Table 3.5: The estimated cost of performing a Sybil attack with $20 hardware to delay honest node connections by 6 minutes, depending on the network size.

| **Network** | **Year** | **Online Nodes** | **Attack Cost** |
|---|---|---|---|
| Bitcoin | 2017 | 10k [65] | $0.79M |
| Ethereum | 2018 | 15k [117] | $1.19M |
| Kazaa | 2002 | 30k [141] | $2.38M |
| *Tribler* | 2020 | 50k [this work] | $3.96M |
| Napster | 2000 | 500k [160] | $39.6M |
| Napster | 2001 | 1.57M [133] | $124.3M |

the attack cost for several other popular peer-to-peer technologies. The costs are calculated using the estimated number of online nodes for each technology. The resulting cost to delay connections between honest nodes (for six minutes) is given in Table 3.5. For instance, we calculate a cost of $124.3M for the 1.57 million online nodes that Napster had in 2001. However, our estimate is based on measurements that were done for a file-sharing application and its heavy bandwidth load decreases classifier effectiveness, as observed in Section 3.7.1. We predict that applications that have lighter bandwidth loads will have more success in detecting Sybils. Therefore, using Sybils to attack networks where users have lighter bandwidth loads is expected to have a higher cost.

## 3.8 Related Work

Decentralized techniques to avoid Sybil identities have been studied in many contexts, including computational puzzles [46, 53, 137, 192], graph-based approaches [45, 227, 243, 244] and reputation mechanisms [30, 111, 158, 242].

In the case of computational puzzles, computational resources of nodes are challenged to limit the influence of attackers. SybilControl [137] proposes admission control that requires each identity to periodically solve computational puzzles, suppressing the influence of attackers who failed to solve a puzzle. To protect honest identities from devoting excessive computational effort, da Costa Cordeiro et al. use computational puzzles that have adaptive difficulty [53] and that are energy-efficient [54]. An issue with the computational puzzle approach is that attackers practically use the computational disparity between consumer devices and their hardware to take control over peer-to-peer networks, in particular the networks of cryptocurrencies. Besides, in the field of cryptocurrencies, these attacks are not just for the sake of vandalism: these attacks are profitable [201]. In contrast, our approach does not depend on the computational resources of attackers, but the routing infrastructure in between an attacker and a target, which is unlikely to be fully in the hands of an attacker given the global structure of the Internet.

Graph-based approaches leverage the sparse connections between Sybils and honest identities. SybilGuard [243] and SybilLimit [244] rely on random walks for ranking nodes to filter out attackers. However, both approaches suffer from a large number of false negatives as well as computational complexity due to the requirement of testing all suspect nodes. SybilRank [45] is less prone to false negatives and reduces the computational complexity of SybilGuard and SybilLimit. However, these graph-based approaches rely on a set of globally known honest nodes called *trust seeds*. In our approach, none of the components assume a-priori trusted nodes.

Reputation mechanisms rely on assigning numerical scores to each identity based on reported interaction histories [30, 111, 158, 242]. However, addressing the problem of establishing trust relationships, through some form of decentralized reputation system, is also beholden to a plethora of attacks [124]. Creating trust requires a circular dependency between the creation of trust and the mitigation of attacks, which easily leads to reputation building with fake histories [209]. We have sought to avoid this problem in our own work by only mitigating attacks based on first-hand knowledge, completely avoiding both requiring centralized infrastructure and the need to establish trust.

**Latency as a Sybil classifier.** Many studies have attempted to use triangulation for detecting and avoiding Sybils, using latency to pinpoint physical coordinates [21, 204] or (locally) by using signal strength [8, 64, 94, 104, 145, 233]. However, the approach of signal strength, while highly similar, is not applicable for interactions over the Internet. Bazzi and Konjevod [21] exploit the geometric properties of network latency to test the distinctness of identities. However, this work builds on the assumption that the latency of nodes has strong geometric properties and that these latency measurements are symmetric. Empirical studies provide sufficient evidence to reject the validity of both assumptions [130, 131, 153]. In our work, we do not rely on those assumptions and consider artificially delayed latency measurements. Sherr et al. propose Veracity [204], a decentralized service for securing network coordinate systems, which also considers the case where an attacker artificially delays the latencies. However, it mainly addresses the coordinate system protection problem and has limited guarantees against malicious identities, providing results for a structured network (DHT) consisting of up to 40% active attackers, using 500 simulated nodes. We show in our experiments that the combination of Basic SybilSys and TrafficJamTrigger can handle up to 99% active attackers, using 500 real users, running real workloads.

**Digital Identity.** Detection and prevention of fake identities plays an essential role in identity management systems [54, 155, 215]. In a recent study, Stokkink et al. [215] have created a decentralized digital identity solution that requires no intermediation by third parties, suitable to even replace physical passports. This may seem to suggest a lack of communication to third parties and therefore to forego the problem of the Sybil attack. Indeed, as pointed out by Maram et al. [155], the building of identity information is Sybil-resistant. However, for the overall solution, this claim of Sybil-resistance would be false since these systems still require communication with others [215]. These solutions require decentrally established communication channels, which can be Sybil attacked. However, once communication to honest nodes is established, these solutions may be a great complement to our technology.

## 3.9 Conclusion

To aid in avoiding Sybil attacks on Web3 applications, this chapter presented Basic SybilSys, a Sybil-avoiding peer discovery mechanism based solely on network latency. Secondarily, this chapter introduced TrafficJamTrigger, a network latency measurement strategy, to allow for detection of attacks that counter our Basic SybilSys mechanism. The combination of Basic SybilSys and TrafficJamTrigger forms Enhanced SybilSys.

Enhanced SybilSys promises to establish connections between users while avoiding Sybils without the need for trust or a centrally governed infrastructure. As Enhanced SybilSys only serves to establish connections, any of the numerous existing Sybil avoidance algorithms and mechanisms (e.g., computational puzzles, social network inference, reputation mechanisms and digital identities) can be used to complement Enhanced SybilSys. Whereas Enhanced SybilSys already avoids Sybils, future work may explore the complementary technology it enables to further filter Sybils and to achieve even less Sybils in a node's neighborhood. Furthermore, to improve Enhanced SybilSys, future work may also investigate other means of detecting Sybils using TrafficJamTrigger measurements.

Through tests with Enhanced SybilSys, it was shown to what extent network latency is able to ensure connections to non-Sybils over the Internet and what the expected monetary cost of undermining our Sybil avoidance is. It has been shown how non-Sybil nodes are still found even when a network consists of 99% Sybils. Using the results for the Tribler network, it is estimated that an attacker would have to invest millions of dollars to obtain a significant presence in a real overlay network to perform a successful (albeit temporary) Eclipse attack. Therefore, SybilSys succeeds in driving up the cost of the previously cheapest and easiest attack on the communication layer of Web3 solutions.

# 4
# Reputation-Based Data Carrying for Web3 Networks

*Web3 networks are emerging to replace centrally-governed networking infrastructure. The integrity of the shared public infrastructure of Web3 networks is guaranteed through data sharing between nodes. However, due to the unstructured and highly partitioned nature of Web3 networks, data sharing between nodes in different partitions is a challenging task.*

*In this chapter we present the Timely Sharing with Reputation Prototype mechanism, which approaches the data sharing problem by having nodes audit each other to enforce carrying of data between partitions. We use reputation as an analogue for the likelihood of nodes interacting with nodes from other partitions in the future. The number of copies of data shared with other nodes is inversely related to the nodes' reputation. Our experiments use emulations to verify the basic behavior of making more copies available when reputations are low and vice-versa. We use real-world traces of both Bitcoin and Twitter to show how our implementation can converge to an equal number of copies as structured approaches.*

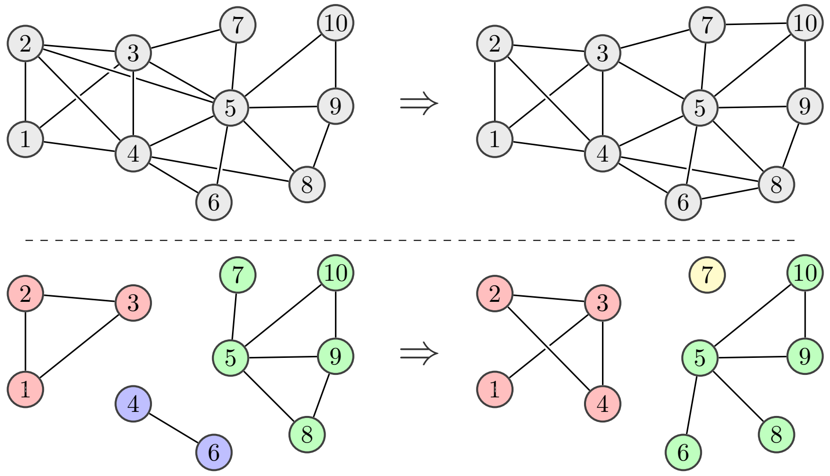


Figure 4.1: Connection graph evolution: *(top)* a typical peer-to-peer network model and *(bottom)* a Web3 network model.

## 4.1 Introduction

The "Web" is turning anarcho-capitalist. Centrally-governed infrastructure is no longer required to share files (BitTorrent), to communicate (OppNets), to transfer currency (Bitcoin), or even to identify at passport strength (TCID [215]). The "Web3" movement goes beyond classical peer-to-peer solutions, seeking to turn our digital world into collaboratively maintained infrastructure [231]. However, as individualistic users only opportunistically interact with others, the classical issue of network partitioning resurges. A partitioned (split) network is incompatible with even the weaker consistency models, e.g., eventual consistency [16]. We present an emergent algorithm for Web3 networks to probabilistically guarantee that data is shared between network partitions by using reputation.

The reality of Web3 networks does not fit the connectivity model of peer-to-peer networks. The traditional way of thinking of peer-to-peer networks is that at least one path exists between any two nodes, allowing a rumor-mongering protocol to deliver information to all nodes. However, we observe a shift away from this connectivity assumption. Even a relatively small network like that of Bitcoin, in the order of 10 000 nodes [174], cannot keep up with the vast volume of user interactions and is seeing temporary partitioning, using aggregated "off-chain" interactions (e.g., the lightning network [181]). Beyond Bitcoin, partitioning has also been observed for Web3 digital assets in the interaction graph of Non-Fungible Tokens [163]. Web3 connection graphs should not be thought of as relatively static networks that periodically update connections. Instead, we should consider opportunistic networks with highly mobile users that frequently migrate between network partitions, exemplified in Figure 4.1.

Web3 networks find themselves in a precarious balance between nodes' ability to maintain the integrity of the data forming their public infrastructure and lowering communication costs. On the one hand, all data should be available to all nodes, allowing them to identify when the infrastructure degrades and to correct it (used in Bitcoin for instance).

On the other hand, the communication and computation requirements of Web3 applications are being increasingly scrutinized, which can be resolved by optimizing an overlay network for locality [157]. However, if overlays are optimized too much and data is never shared between network partitions, these networks become vulnerable to attacks again. Instead of requiring over 50% of the network to be malicious in order to degrade the infrastructure, an attacker would require only over 50% of a partition's size. Some data has to periodically be shared between partitions.

The main problem that we address in this chapter is the need for data sharing between partitions in split networks with highly mobile nodes. We leverage reputation mechanisms to select nodes that are likely to traverse partitions to carry—and share—data. A node's reputation reflects the willingness of other nodes to interact with it. Therefore, nodes with a high reputation are more likely to share data with others than low-reputation nodes and make for prime carriers to share data between partitions. In contrast, more low-reputation nodes are needed to achieve a comparable data sharing probability. Our contributions are as follows:

- We detail the requirements for Web3 data sharing solutions, and how they can be attacked (Section 4.2).
- We motivate the design of our *Timely Sharing with Reputation Prototype* (TSRP) in Section 4.3.
- We use emulations to show how more copies of data become available as reputations change (Section 4.4).
- We apply our prototype to real-world traces of Bitcoin and Twitter and show how the number of copies converges to the number for structured networks (Section 4.5).

## 4.2 Requirements analysis

We now derive requirements using three typical Web3 assumptions. Firstly, Web3 applications assume that over half of the network's nodes are not malicious. Secondly, nodes do not enter the system with prior trust for other nodes. Lastly, it is assumed that there exists no node with special authority.

**An emergent algorithm is required** to fit our assumptions. In a Web3 system there is both a lack of trust between nodes and no node has authority over any other node. This translates to the physical world in that the users, each of whom controls a node, cannot be told to form a physical connection to certain other nodes. For example, by interpreting an application-layer interaction graph it may seem reasonable for the application to request another interaction with a specific node, but this may involve physically moving devices around the world to form a physical connection due to the high mobility of Web3 nodes. In general, we believe it to be nonsensical for an application layer to force interactions on the network layer. Physical connections emerge and any algorithm should fit this.

**Load-balancing over nodes is required** to meet Web3's incentives. Without external regulation, peer-to-peer interactions lead to rich-get-richer dynamics [121, 220]. Therefore, if interactions are used to select nodes to carry data, nodes with more interactions become disproportionally burdened with data. To select nodes to carry data our

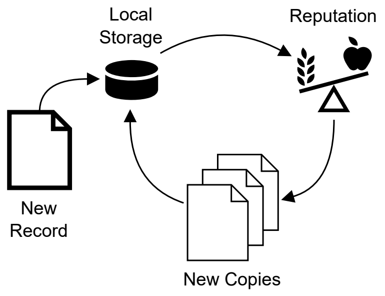


Figure 4.2: New records and new copies from others are carried in a node's local storage between partitions based on reputation.

proposal uses reputation, which is based on nodes' interactions. Therefore, simply selecting high-reputation nodes as carriers for data places an unequal burden on these nodes, effectively punishing them for a high reputation score. In turn, unequal burden leads to unequal reciprocity between nodes, which is not compatible with peer-to-peer incentive mechanisms without payment [81]. Nevertheless, we do expect nodes with a high reputation to interact more frequently with other nodes and, therefore, to be more effective as carriers for data between partitions.

**All nodes have a partial view of a Web3 system** due to the lack of trust in data obtained from others. However, nodes do share logic that allows them to verify that the data they receive complies with a system's rules. This approach was pioneered by Bitcoin, in which every node adopts the longest chain of correctly hashed data (known as "Nakamoto consensus"). In a general sense, this corresponds to a Web3 system design pattern that consists of providing all nodes with an algorithm to verify the integrity of their common public infrastructure. However, nodes are never forced into accepting data and they can subjectively deviate from the shared logic.

**Records are required to both be locally stored on nodes and be shared to avoid network partitioning** in our system model. We use the term *record* to refer to all data that is usable by a given Web3 application. A record may be encrypted, for which reputation mechanisms exist for our solution [7, 98, 198]. Every instance of the same record is what we refer to as a *copy*. Both records proposed by a node and those copied from other nodes are stored in a node's *local storage*. The implementation of the local storage can be a simple database and may serve a higher-order data structure, like a blockchain.

Our system model consists of three components that form the feedback cycle shown in Figure 4.2. First, any newly introduced record becomes part of the local storage of a node. A node decides which other nodes to share a record with based on its shared verification logic and by using its stored copies. Other nodes that receive a record verify that they should store a copy of the record in their own local storage using the same shared logic. The feedback cycle continues until nodes no longer want to, have to, or have space to, store a copy of the record.

**The refusal to store records is the main attack** a data sharing solution is required to resolve. Any node that does not store records should be detectable by any other node through shared verification logic. Nodes may then choose to not further interact with a node that does not store what it must store. We do not distinguish between nodes with malicious intent and non-malicious nodes that have simply run out of storage space, as both necessitate additional copies of a record in the network. However, specifically for our solution, the manner of calculating the change in reputation due to not storing a copy can differ per node. Nevertheless, the reputation of a node is still based on the copies it shares with other nodes.

## 4.3 Design

Our *Timely Sharing with Reputation Prototype* (TSRP) is designed to select nodes to carry copies of records based on nodes' reputation. Using both the identity and the reputation of a node, TSRP provides the shared logic to decide whether the node "must store" a given record (i.e., locally store a copy).

### 4.3.1 Overview

Every node using TSRP follows a simple two-step decision flow to determine whether a node (including the node itself) should act as a carrier for a given record. First, the distance between an identifier, which can be any arbitary string (likely a public key), and a record is calculated using a distance function over their hash values, which we call the *binding score*. Finally, the binding score is compared to the reputation of a node to decide whether it "must store" the given record. The decision flow of whether a node must store a copy of a record is schematically given in Figure 4.3.

The distance between an identifier and a record, i.e., the binding score, is used in a similar fashion to how DHTs decide which identifiers should store certain records. Typically, DHTs, like Kademlia [156], use XOR functions to bind nodes' identifiers to the records they should store. In particular, TSRP uses the Hamming distance, which is part of the class of symmetric XOR functions [33]. The Hamming distance function has well-known statistical properties that we use to construct a proof for the number of copies of records in Section 4.3.4. The binding score is used to provide the initial association of records with nodes and has nothing to do with the value, quality or importance of a record.

The binding score is made to resist index poisoning attacks, which consist of creating fake identities until one with a high binding score is found [147, 210]. The score is grounded in values produced by a pseudo-random number generator (PRNG). The PRNG is seeded using (a binary string representation of) both a record and a node identifier. Thereby, the generated values are always the same, regardless of which node evaluates it and when it is evaluated. Assuming attackers cannot pre-empt the creation of a record, this disallows index poisoning attacks through fake identities.

The reputation of each node is calculated individually by nodes it interacts with. As a result, the calculated reputations may differ wildly between nodes, depending on the node that calculates it. As the reputation of a node is a factor in deciding whether or not it should store a given record, nodes using TSRP may not come to the same decision on whether a node should store a particular record. Nevertheless, we later prove and show, through both emulations and real-world traces, that nodes are expected to store copies.

A node with a high reputation should be avoided when choosing carriers for records to achieve load balancing. In a peer-to-peer setting, the reputation of nodes is a reflection of

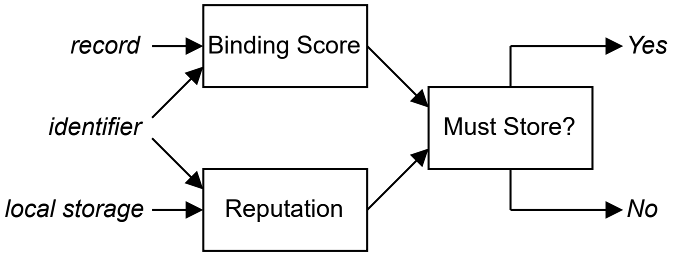


Figure 4.3: How a node uses its local storage to decide if a copy of a record must be stored by another node, based on its identifier.

the "work" they have done that has gone unreciprocated [214]. Therefore, selecting high reputation nodes to perform more "work", in the form of carrying records for TSRP, is not compatible with the incentives of Web3 applications (as presented in Section 4.2). To keep incentive compatibility, TSRP makes a node's reputation inversely related to the probability of it storing any record. New records are more frequently coupled to nodes with a low reputation. Because these low-reputation nodes are untrusted, they are given more records to carry: a lack of reputation requires nodes to perform more "work". Therefore—assuming nodes wish to do less "work"—nodes are incentivized to have a high reputation.

### 4.3.2 Auditing

To detect nodes in the network that do not follow the shared storage logic, the main attack on TSRP (Section 4.2), we use a witness protocol, also known as auditing. To this end, the decision flow of Figure 4.3 is again used to determine the records that a node is required to store. Each audit consists of a request for proof of an audited node storing the required records, based on their estimated reputation and the auditor's known records.

Audit proofs can be naively implemented by showing the record belonging to a hash. However, if needed, these proofs can also be implemented through verification in the encrypted domain [248]. We use the naive auditing approach using hashes within the scope of this chapter.

Failing an audit (i.e., not being able to show that a required record is stored locally) negatively affects the reputation of the audited node. Nodes that fail audits are punished through ostracization by setting their reputation to 0 until they store the required records.

As Web3 users are expected to be egocentric and records are expected to become stale, nodes are both expected and incentivized to remove records that are no longer interacted with. For example, in Bitcoin this staleness occurs after blocks have six descendants, at which point blocks are considered to be probabilistically finalized [191] and no longer require frequent auditing. As shown in PopCache [219] and StreamCache [140], basing local caching of information on staleness (i.e., interaction rate) greatly reduces communication costs.

### 4.3.3 Formalization

We now formalize our decision flow that leads to records being more frequently stored by nodes with lesser reputation. In order to prove that our decision flow leads to availability

Table 4.1: Symbols

| **Symbol** | **Description** |
|---|---|
| $n$ | The size of the set of all node identifiers. |
| $i$ | A randomly-assigned node identifier (can be any binary string, most commonly a public key). |
| $\mathscr{R}(i)$ | Reputation function for a node identifier $i$. |
| $\Delta$ | Hamming distance function. |
| $H_Q$ | A uniformly distributed secure hash function that produces a binary string of $Q$ bits. |
| $Q$ | Length in bits of the hash function $H_Q$ output. |
| $Z(x)$ | Function to map a uniformly distributed random variable on $[0,Q]$ to a distribution on $[0,1]$. |
| $D$ | The minimum storage factor between 0 and $Q-1$. |
| $d$ | Record (possibly encrypted). |
| $r(d,i)$ | A pseudo-random function that produces a binary string of $Q$ bits (the pseudo-randomness is seeded with the node id $i$ and record $d$). |
| $S(d,i)$ | A Boolean function indicating whether node id $i$ should store record $d$. |

of records through random carriers, we provide the concrete definitions of its elements. Our formalization uses the symbols captured in Table 4.1. For any node $i$ it can be calculated whether or not $i$ should store a specific record $d$ (a node may also decide this for its own identifier). To do so, the *binding score* $\mathscr{B}(d,i)$ is calculated. Basically, a node $i$ must store a record $d$ if its reputation $\mathscr{R}(i)$ is smaller than its binding score $\mathscr{B}(d,i)$ for $d$. If a node does not store what it must store, we consider it to be a malicious node. We formulate the store condition function $S$ as follows:

$$S(d,i) \equiv \mathscr{R}(i) \leq \mathscr{B}(d,i) \tag{4.1}$$

This carrier-selection function $S$ is deterministic. When given the same reputation $\mathscr{R}(i)$ and binding score $\mathscr{B}(d,i)$, all nodes will have the same output for $S$. However, nodes do not necessarily agree on the reputation $\mathscr{R}(i)$ of a node $i$.

The binding score $\mathscr{B}(d,i)$ is calculated for a record-identifier pair $(d,i)$ in four transformations. First, the PRNG, seeded with the concatenation of the binary strings representing $d$ and $i$, is used to generate a binary string of $Q$ bits. Second, a uniformly distributed secure hash function $H_Q$ is used on the binary string representing the node identifier $i$, shortening the representation without violating its probabilistic distribution. Third, the Hamming distance is calculated between the PRNG-generated binary string and the hash value (i.e., two binary strings of the same length $Q$). Lastly, the resulting distance is mapped to the interval $[0, \frac{Q}{Q-1}]$ with a mapping function $Z$. In short, these four transformations are formalized as follows:

$$\mathscr{B}(d,i) = Z\Big(\Delta\Big(H_Q(i), r(d,i)\Big)\Big) \tag{4.2}$$

The mapping function changes the relationship between a node's reputation and its probability to store any particular record. For example, $Z$ can be chosen to have a doubling of reputation lead to a quarter of the storage requirements. In the context of this chapter, we use $Z(x) = x/(Q - D)$, where $Q$ and $D$ are integers where $Q > D \geq 0$. Both the length of the hash $Q$ and the minimum storage factor $D$ can be chosen to guarantee a minimum expected number of copies of a record (expanded upon in Section 4.3.6). Other choices are left to future work.

## 4.3.4 Lower bound on available copies

We now discuss TSRP's probabilistic guarantee on the number of available copies of a record. Given our definition of TSRP and the function $Z(x) = x/(Q - D)$, we show that any record $d$ has an expected lower bound of $n \times \left(\frac{1}{2}\right)^Q$ copies.

By definition of $S$, nodes store records if $\mathscr{R}(i) \leq \mathscr{B}(d, i)$. The least number of nodes that will store a copy of a given record $d$ occurs when the reputation $\mathscr{R}(i)$ of every node $i$ is equal to the maximum value of 1. By expansion of $\mathscr{B}(d, i)$, we derive $1 \leq Z\left(\Delta\left(H_Q(i), r(d, i)\right)\right)$. The smallest number of copies are now stored if the function $Z$ is minimized, by setting $D$ to 0. Our storage function is now simplified to $Q \leq \Delta\left(H_Q(i), r(d, i)\right)$. As both $H_Q(i)$ and $r(d, i)$ are bit strings of length $Q$, the inequality can only be satisfied if the distance between both strings is maximized: equal to the chance that both strings are exactly equal, which we use for more compact notation. Because its inputs, $r$ and $H_Q$, are both uniformly distributed and independent, it follows that the Hamming distance $\Delta$ follows a binomial distribution $B(Q, \frac{1}{2})$. The probability to store any record $r$ *on a particular node* $i$ is given by $P(H_Q(i) = r(d, i)) = P(X = Q \mid X \sim B(Q, \frac{1}{2})) = \binom{Q}{Q}\left(\frac{1}{2}\right)^Q (1 - \frac{1}{2})^0 = \left(\frac{1}{2}\right)^Q$. Finally, all $n$ nodes together are expected to store at least $n \times \left(\frac{1}{2}\right)^Q$ copies.

We stress that this lower bound is an expected value, which gives the average number of copies. Therefore, some records may even have no copies stored when all nodes have the maximum reputation value. If this is an issue for a system, TSRP should be parameterized to disallow this and we discuss how to do this in Section 4.3.6.

## 4.3.5 Estimating reputation

TSRP requires a reputation mechanism that allows a node to approximate the system-wide reputation score of a node, within a known error $\varepsilon$. Suppose a node $i$ checks another node $j$. If the reputation of $j$ is higher than its binding score for a record, $j$ removes the record from its local storage. However, if $i$ underestimates the reputation score of $j$, $i$ concludes that $j$ fails to comply with the storage logic. These types of erroneous conclusions can be resolved for any reputation mechanism $\mathscr{R}$ with a known estimation error of $\varepsilon$. Increasing the reputation score to $min(\mathscr{R}_{max}, \mathscr{R}(j) + \varepsilon)$ trivially guarantees that $i$ does not mistakenly conclude that $j$ fails to comply with the storage mechanism for the given error $\varepsilon$.

A node may estimate its own reputation incorrectly for a variety of reasons. For example, due to the differing reputation mechanisms employed by other nodes or their malicious behavior, views of a node's reputation may be inconsistent. In these cases, a "correct" estimation of a node's reputation does not exist. In practice and in the following experiments, we therefore scale the current reputation estimate of a node's own reputation estimation by the amount of interaction rejections by other nodes. Upon a successful interaction, the scalar for the node's own reputation estimation increases and upon a rejection the scalar decreases.

It may be desirable for a node to only allow interactions with nodes that calculate a similar reputation score to what the node calculates for itself. This similarity enforces other nodes to obtain a sufficiently equal view of the network to obtain the same reputation score, thereby increasing the effort required by other nodes to interact with the local network partition.

### 4.3.6 Parameterization

Choosing appropriate values for $Q$ and $D$ is an important design choice. The influence that the reputation mechanism has on the number of copies corresponds to $Q$. Higher values of $Q$ lead to changes in reputation causing more dramatic changes in the number of copies. The mapping of the malicious fraction of nodes in the network, over the range of $Q$, is reflected by the value of $D$. We assume that any use case will, at the very least, wish to cover random node failure (i.e., making at least 2 copies of a record available at all times). We now discuss the appropriate choice of $Q$ and $D$ in more detail for the lower bound of two records.

We first derive the choice of $Q$ that allows for scaling between 2 and $n$ copies in the network. In this case, $Q$ has to be chosen according to the condition $n \times \left(\frac{1}{2}\right)^{Q} \geq 2$ to assure the minimum of 2 copies. This leads to the restriction $Q \leq log_2(n) - 1$. Though this seemingly implies a small value for $Q$, failure to choose an appropriately *large* $Q$ for the network size gives rise to additional copies. For example, in a network of 50 000 nodes, choosing $Q = 1$ leads to a probabilistic lower bound of 25 000 copies and $Q = 14$ leads to a lower bound of 3.05 copies: a difference of 24996.95 copies per record. Of course, the more copies of a record that are available in the network, the harder it will be for attackers to perform a poisoning attack. The network size $n$ can be estimated from local interactions [114, 200] and, as the error of the estimation scales with the network size, fits well with the logarithmic operation to calculate $Q$.

The estimated malicious node fraction in the network leads to the value of $D$. The idea of $D$ is to change $Q$, through $Z$, in such a way that the expected lower bound of copies per record is larger than the number of malicious nodes in the network. Because the fraction of malicious nodes known to any node converges to the fraction in the system [38], we can estimate an appropriate value for $D$ given the current estimated network size. Given the (locally known) fraction $m$ of nodes that is verified to not follow the shared storage logic, each node sets the adaptive value $D = Q - log_2(mn)$. In the remainder of this chapter we assume this adaptive value of $D$.

## 4.4 Storage based on carrier quality

The purpose of this section is to show that TSRP dynamically makes copies available according to the available nodes and their estimated carrier quality (i.e., their reputation). We evaluate the emergent carrier selection of TSRP on networks where nodes have continuous peer-to-peer interactions. We perform three emulations to explore TSRP's carrier selection through emergent behavior, based on the number of copies in the network. The first two emulations show the impact of the network size and the record volume. In our last emulation, we present the impact of the number of malicious nodes.

In our emulation setup, we use real latency data from the Internet (sampled from the King data set [55], which has a maximum latency of 800 ms). Each node in our experiments is mapped to a node in the latency data set, which contains the latencies to all other nodes in the network. For each of the experiments, we impose a network topology where each node has bidirectional communication with up to 30 random other nodes, regardless of the network size. For all emulations we use a machine with 64 physical cores (clocked at 2.4 GHz) with 528 GB DDR4 RAM (clocked at 2933 MHz) available.

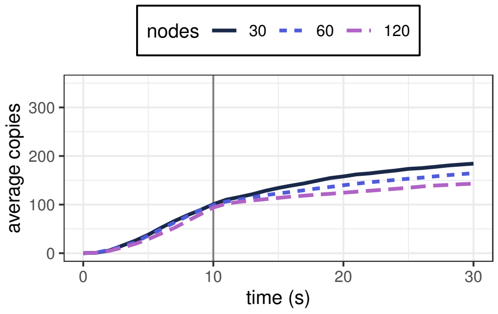
Our methodology consists of having *nodes continuously generate records*, to match our problem description. For record generation, each node generates a string of characters and shares it with a randomly selected node. We fix the record generation rate per node to two records per second. The contents of these records consist of 20 randomly-generated bytes (which would be replaced with application-defined data when used in a real application). Our first two emulations do not introduce malicious nodes. Even so, every node still runs the full TSRP protocol, performing audits before each interaction to ensure others follow the shared storage logic.

**We determine the impact of the network size** on carrier selection when the reputations of all nodes in the system change. In order to have control over the—otherwise unpredictable—reputation values, we force all nodes to use the same reputation values for other nodes in the network. We change all of the reputations at the fixed time of 10 seconds into the experiments. We call this sudden change in reputation a *flip*. We perform a flip of the reputation values of all nodes between the minimum possible value (0) to the maximum possible value (1). After the flip in reputations, nodes continue generating and disseminating records for another 20 seconds.

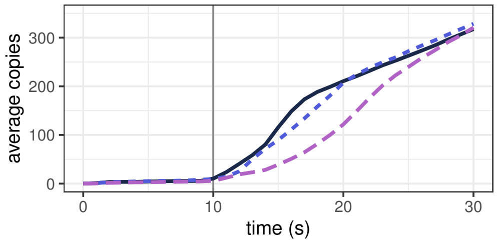
In Figure 4.4a we show the average number of copies (regardless of the associated records) stored per node when reputations flip from 0 to 1. Due to our overlay network topology setup (30 connections for each node), the larger network of 120 nodes takes longer to receive records. Intermediate nodes need to forward records in larger networks, but eventually the same average record storage rate is reached as for smaller networks. For all three shown experiments, at the 10 second mark each node starts storing distinctly less records. As all nodes are now deemed to be high quality carriers, nodes store less copies overall when reputations flip to a value of 1.

The number of copies in the network as reputations flip from 1 to 0 is shown in Figure 4.4b. In this case, all nodes have to collect additional records. As the network size increases, nodes also require more time to detect that their own reputation estimate is too high (as nodes are not interacting with them anymore due to our ostracization strategy). Furthermore, retrieving the missing records is slower in larger networks. Nevertheless, we conclude that TSRP correctly selects more carriers to store copies when reputations are low and vice-versa.

**The impact of the number of the locally missing records** is what we investigate next, as retrieving records is the most difficult case to recover from. We fix the node count to 60 nodes. Like before, we present the impact of flipping node reputations at different times on the carrier selection.

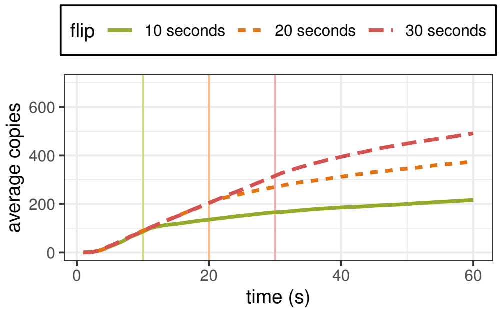


(a)

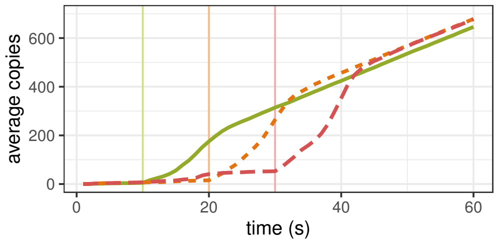


(b)

Figure 4.4: The average number of copies stored per node when at 10 seconds all reputations flip from 0 to 1 (a) and 1 to 0 (b). Every node generates two records per second.



(a)



(b)

Figure 4.5: The average number of copies stored per node when at 10, 20, and 30 seconds all reputations flip from 0 to 1 (a) and 1 to 0 (b). Every node generates two records per second.

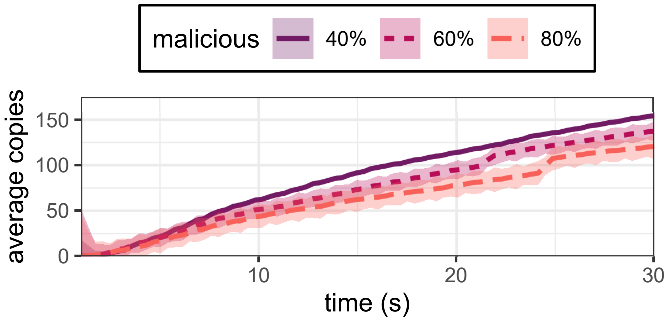


Figure 4.6: Average copies per record over time using PageRank for malicious node fractions (shaded 95% confidence interval).

We visualize the flip of reputations from 0 to 1 in Figure 4.5a. As there are no records to catch up on, all nodes reduce their stored copies nearly instantly. In contrast, Figure 4.5b shows the effect of generating more records before the reputation flip from 1 to 0. The longer we wait before we change the reputation, the larger the number of records that will have to be redistributed in the network. As a result, it will take more time for all nodes to gather all the records. Even so, we observe the emergent effect of adapting to known reputations.

**We now introduce malicious nodes to the network and exhibit their impact** on the number of copies. We consider malicious behavior to be the refusal to store any records. We again measure the average number of copies, this time for different fractions of malicious nodes. Since the number of nodes only serves to delay the time to converge to a stable record storage rate, we fix the node count in the experiment to 60. We use *incremental personalized PageRank* [13] as the reputation mechanism for all nodes. As opposed to random selection of interaction partner in the previous experiment, we mimic rational users, corresponding to our problem description. Each node samples another node from a categorical distribution formed by their PageRank scores.

We now examine the number of copies in the network for the malicious node fractions of 40%, 60%, and 80%. Intuitively, all honest nodes should hold a high reputation, which, in turn, would lead to an average number of copies close to the required minimum. However, the results, presented in Figure 4.6, show that TSRP makes more than 2 copies per record available for all three malicious node fractions. As the fraction of malicious nodes increases, the average number of copies drops. The discrepancy between intuition and these results is explained by the fact that reputations calculated by PageRank are normalized and personalized. This normalization causes higher relative differences in reputation in the presence of malicious nodes, but equalizes reputation as the fraction of honest nodes becomes higher. The end result is that PageRank causes TSRP to store records more sparsely among honest nodes as the fraction of malicious nodes increases. Though it is left out of scope, for this reason the carrier-selection behavior can be changed through the mapping function $Z$.

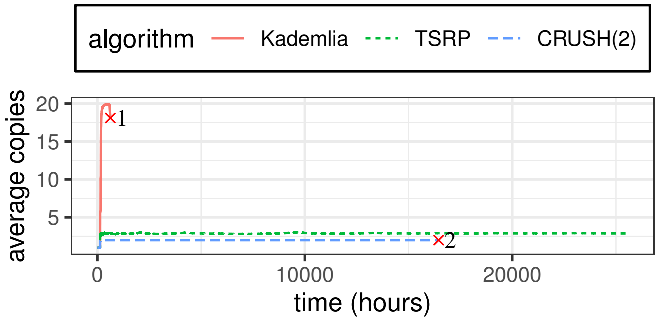


Figure 4.7: Average number of copies over time (since 1 March 2009) for several storage algorithms applied to Bitcoin. The numbered crosses correspond to the crashes of Table 4.2.

Table 4.2: Crash causes for the Bitcoin experiment.

| # | Cause | Crash time (hours) | Median copies |
|---|---|---|---|
| 1 | Out of memory | 622 | 19.80 |
| 2 | Segmentation fault | 16452 | 2.00 |

## 4.5 Real-world traces

We apply TSRP to real-world traces of both Bitcoin and Twitter to show that the emergent storage of TSRP does not necessarily lead to many more copies of records in comparison to storage schemes for structured peer-to-peer networks.

We replay traces using our 64-physical-core machine (clocked at 2.4 GHz) with 528 GB DDR4 RAM (clocked at 2933 MHz). The Bitcoin trace is replayed at 2000× the original speed, while the Twitter speed remains untouched. To provide a benchmark for the number of emergent copies of TSRP, we choose two representative solutions for respectively DHT solutions and data allocation solutions: Kademlia [156] (introduced in Section 4.3.1) and CRUSH [237]. The former is widely deployed [211] and the latter is part of the distributed file system Ceph [236].

We use the default parameterization of both Kademlia and CRUSH as much as possible. By default, Kademlia attempts to allocate 20 copies per record. Crush operates on a given amount of desired copies, which we will set to a value of 2 to provide crash fault tolerance and denote as `CRUSH(2)`. This is typical for allocation and provisioning schemes, like the default replication factor of 3 in HDFS [37] and 2 to 3 for different kinds of information in ZFS [23]. We also insert an additional head node for CRUSH that monitors the storage capacity of the storage nodes and provides the storage mapping for each record.

**We replay a real-world Bitcoin data set** that includes both real records and real node identifiers. To do so, we use the first 156056 blocks (nearly 3 years of data) from the Bitcoin data set supplied by Kondor et al. [121]. We utilize this particular data set, as it also includes an address-to-node mapping which we use to avoid assigning new identifiers to the same nodes. During this replay we fix the reputation of all nodes to 1, the main attack

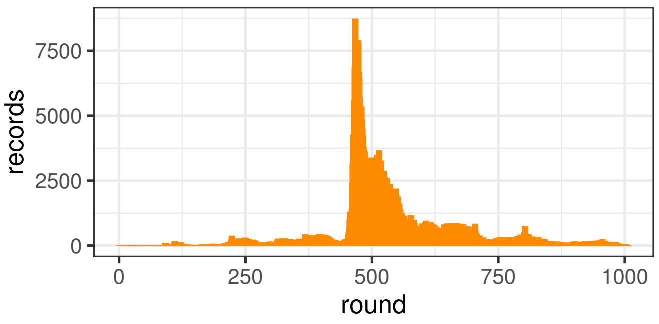


Figure 4.8: Number of records per round in the Twitter data set.

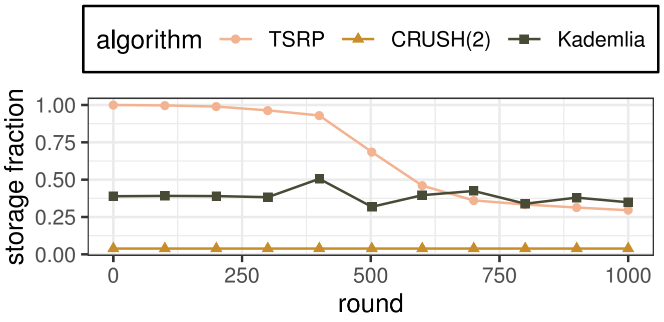


Figure 4.9: The average storage fraction (i.e., the copies per record per node) per round for the Twitter data set.

vector on reputation-based storage [172].

Figure 4.7 shows the average copies per record in the network as Bitcoin users join the network and interact with each other. For the implementations that crashed during the processing of the data set, Table 4.2 shows the number of hours an implementation managed to process and the median number of average copies stored. This average was calculated over all nodes interacting with the system at any point in time. After the first 6 Bitcoin blocks (records), enough users have joined the network for TSRP to ensure the availability of 2 copies for all records in the system. Albeit after a longer time than Kademlia and CRUSH, TSRP does converge to a constant number of average copies. However, both the sample standard deviation of the number of copies stored per particular record (1.8069) and the maximum number of copies for a record (10) are higher for TSRP at the end of this trace replay.

**We replay a Twitter data set** in such a way that data in shared between users, instead of data being distributed by a server to users. We use a Twitter data set that covers a flash crowd of tweets about the Higgs boson [60]. We construct records by mapping the “retweets”, “mentions”, and “replies”, between users to records shared between nodes. Through this construction, the trace that we replay contains 563069 records between 304691 nodes captured over 604621 seconds (roughly a week). We share records between nodes every ten minutes, to form the total of 1008 rounds of these record exchanges, for which we show the number of records per round in Figure 4.8.

The source of reputation for the Twitter data set comes from the number of "followers" per Twitter user. Reputations are calculated using the "simplified version" of the Distributed EigenTrust algorithm [111] (normalized toward the median, a localized trust value of 1 if there was a record between nodes and 0 otherwise, and using a "forget factor" of 0.2). The follower count from the data set serves as the "a priori trust value" for each node. As the trace is played, this simplified EigenTrust algorithm causes the median of all the reputations of nodes in the network go to 1 and the average to near 0.72.

The visualization of the number of copies per record per node is given in Figure 4.9, with a data point sampled every 100 rounds. As users build reputation in the network, we see that TSRP stores a similar number of records on average to Kademlia. In the final state of this experiment, in the 1008th round, only 29.5% of the network stores a particular record on average with an average reputation score of 0.72. This is a 70.5% decrease from the starting state, where 100% of all nodes stored a particular record on average. Considering the number of stored records, TSRP successfully achieves the same result as Kademlia, a system that has been carefully parameterized by networking experts for well over a decade.

## 4.6 Related Work

Our proposal of emergent carrier selection is closely related to information-centric networking, but critically releases the assumption of a connected network to fit Web3. PopCache [219] and StreamCache [140] are examples that underline the importance of the *information hit rate* to serve caching. PopCache explores serving content to users through a single node, a cascading network or a binary tree. StreamCache uses a tree topology. We go beyond the connectivity assumption of these caching solutions and show how the hit-rate of peer-to-peer interactions is sufficient to retain an arbitrary number of cached records in an unstructured and split network. Our approach is supported by the research of Bulut et al. suggesting that the friend-to-friend relationships of highly mobile users are sufficient to route data to all network participants [39].

In general, optimizing networks for node locality leads to lesser communication requirements between nodes [157]. For example, within the domain of edge storage, it has been observed that eventual consistency leads to lower communication requirements [161]. This phenomenon has also been observed for semantic overlays [51], DHTs [146], and causal multicast [61]. In our approach, we do not attempt to create a structured network to fit the emergent property of locality and, instead, we make use of the unstructured, but clustered and dynamic, network that emerges from Web3 applications.

The biggest difference between our proposal and other works is the use of storage to allow for inconsistency detection, instead of using storage to serve data. To detect inconsistencies, previous research has shown that auditing is highly effective ([95, 205]). Auditing has even been shown to be able to detect Eclipse attacks by Singh et al. [205]. Audits have also been generally employed to deter and to detect faults in state machine replication through "accountability" in PeerReview [95]. However, the assumption for systems that use audits is that *random* nodes perform the audits, instead of only nodes that interact with the data. Therefore, in order to randomly audit data, the data must be shared in such a way that it reaches random nodes. TSRP uses auditing to enforce storage by carriers of records and, in turn, ensures the auditability of the public infrastructure of a Web3 application through the mobility of its users.

## 4.7 Conclusion

Web3 networks need to be approached differently. Structure cannot be imposed over these networks. There is no authority to govern nodes and users should be assumed to be individualistic and egocentric. Connections between users are opportunistic, making a Web3 network prone to partitioning. Users do not wish to serve each other data, but rather police each other to uphold the integrity of their shared public infrastructure. However, we have shown that network-wide storage of records is unnecessary for these types of networks. Without breaking the Web3 incentives for cooperation, records can be shared efficiently between partitions. Our prototype, TSRP, allows users to audit each other to make sure that the integrity of records in a Web3 system is upheld.

# 5
# A Local-First Approach for Green Smart Contracts

*Shared code in blockchains, known as smart contracts, stands to replace important parts of our digital governance and financial infrastructure. This work evaluates permissionless execution of smart contracts that is tightly coupled to cryptocurrencies and Proof-of-Work blockchains. In this case, smart contracts inherit the environmental impact of their associated Proof-of-Work blockchain, like its energy consumption, carbon footprint, and electronic waste. The four concepts of relaxed consistency, strong identities, probabilistic consensus, and the use of liabilities instead of assets may change the status quo.*

*This chapter explores the integration of these concepts to decouple smart contracts from Proof-of-Work blockchains. By means of a local-first approach, that may expose users to non-final contract states, the architecture of smart contracts can be transformed to become green. Because the contract states that are presented to users may change, we base the interactions between users on liabilities (i.e., a change in state is a liability to a user). We propose a novel paradigm for smart contract architectures, named Green Smart Contracts, that is based on a local-first approach. Furthermore, we present and implement a prototype solution for this paradigm. We validate the need for a mechanism to resolve consistency violations by replaying the contract calls of a real smart contract. Our simulation shows that violations occur more often (13% of contract invocations) when using liabilities than when using a traditional blockchain (3% of contract invocations). However, we additionally validate that they can be avoided using a consensus mechanism, and our experiments show that a publish-subscribe messaging pattern uses the fewest messages to do so, though it may not be applicable for use cases that disallow the inherent imbalance in the messaging between peers. Our carbon emission estimation shows that a Green Smart Contract approach lowers carbon emissions by 52.31% when compared to the messaging behavior of a typical peer-to-peer blockchain with 1000 nodes.*

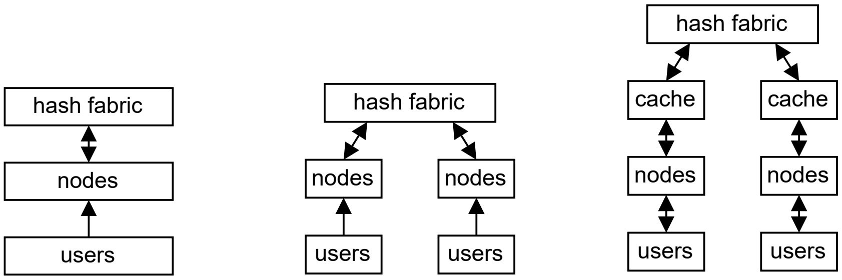
## 5.1 Introduction

By 2030 it is expected that digitization and automation will make 80% of current financial firms "go out of business, become commoditized or exist only formally but not competing effectively" [87]. Executing the newly emerging digitized and automated processes will require multiple servers that are governed by different entities that have mutual distrust (e.g., due to geopolitics). Therefore, the financial sector is looking into Web3 developments like blockchains and smart contracts to modernize its infrastructure [71]. Smart contracts, which consist of program code that is published in a blockchain and seek to provide permissionless execution, transparency, and availability of source code, can serve to meet the demands of new digitized governance and financial infrastructure [22, 49, 52, 62, 99, 184, 203, 223]. Some smart contracts are even already seeing up to 20 000 invocations per day [180]. Unfortunately, the blockchains that power smart contracts are not environmentally friendly [199]. The Chinese government has even banned cryptocurrencies due to their utilized resources, energy consumption, carbon footprint, and electronic waste [238]. This chapter seeks to integrate and leverage recent proposals to realize a paradigm shift in the architecture of smart contract systems and, thereby, provide a greener alternative for future digitization and automation.

Traditional models used by, e.g., Bitcoin [164] and previously Ethereum[1] [240] allow all their users to *observe and verify* that the program code of a smart contract is executed correctly, while collaboratively appending to the same hash fabric (e.g., a blockchain). In the traditional system model, users attempt to invoke smart contract code and nodes attempt to add these invocations to a blockchain in return for payment. When added to a blockchain, the code can be observed and executed on all nodes. Thereby, every node in such a system can calculate the state of a smart contract based on the history of invocations. As all nodes are expected to execute all invocations of the smart contract code, the responsibility for its correct execution does not depend on a single party but on the entire network. Network-wide verification allows smart contracts to achieve fairness and cooperation among competitors in a trustless context [73]. In short, the system model of the traditional approach to smart contracts, shown in Figure 5.1a, consists of a pool of users that invoke smart contracts and a pool of nodes that attempt to add these invocations to a shared hash fabric, consisting of a blockchain.

Several concepts have been proposed for the blockchain ecosystem that could be integrated to realize a paradigm shift. Generally, the metric of "throughput" (i.e., the number of contract invocations per time unit for smart contracts) is optimized and with it comes lesser communication requirements and lesser environmental impact [12]. A throughput increase is often achieved through the concepts of weaker consistency models (to capture how invocations are recorded) and weaker consensus models (the manner in which invocations are agreed upon), and typically these approaches no longer use a traditional single chain of blocks but rather a hash fabric of blocks. For example, "sharding" uses cliques of nodes, known as "shards", that communicate intensively internally but very little between each other [221], and Hashgraph opts for a Directed Acyclic Graph instead of a single chain [17]. As a result, the system models of recent proposals, shown in Figure 5.1b,

[1]As of September 15th 2022, Ethereum has switched away from their traditional model and it now uses a so-called "Proof-of-Stake" model, which we do not further discuss in this chapter.



(a) Single user pool; single node pool; single shared hash fabric.

(b) Multiple user pools; multiple node pools; multiple hash fabric branches.

(c) Multiple user pools; multiple node pools; multiple hash fabric branches; multiple ephemeral hash fabric states (caches).

Figure 5.1: System models for information flows between users' smart contract invocations and the underlying hash-based data structures: from traditional proposals (a) to recent proposals (b) and local-first (c).

have to consider multiple pools of nodes that attempt to synchronize with a shared hash fabric. However, users do not actively take part in this synchronization.

Smart contracts are not necessarily compatible with weaker consistency and consensus models. Depending on how a consistency model is weakened, the safety and liveness guarantees of a system may radically change. Users and nodes may not be able to observe (all) interactions with smart contracts, they may not be able to observe interactions in the right order, and perhaps they may only be able to observe interactions after a long delay. Furthermore, weaker consensus may lead to temporary decisions on allowable inputs and, by extension, a change in the values of the outputs of smart contracts [139], which must be retroactively corrected.

A next step, away from network-wide verification, is a local-first approach to the smart contract execution model. The approach proposes to make sharing interactions with other nodes a secondary concern [118]. The corresponding system model, shown in Figure 5.1c, requires users and nodes to actively engage in change management by merging the changes in their hash-fabric caches to the global shared hash fabric. The local-first approach was originally proposed for human users that collaboratively edit a shared data structure but produce very little consistency conflicts [118]. Thereby, local-first is a green approach to smart contract execution, as it eliminates the need for Proof-of-Work and minimizes the required communication of data. However, when integrated into existing blockchain systems, a local-first approach would still need its invocations paid for using the cryptocurrencies of Proof-of-Work blockchains. Otherwise, users would not be incentivized to store invocations, so they can be verified by other users. Furthermore, like a sharding approach, a local-first approach depends heavily on the identities of nodes to edit caches, whereas blockchains typically do not have identity management, beyond public keys.

This work enables a local-first approach by challenging the need for cryptocurrencies and arguing that identities can be used in a permissionless smart contract system. Firstly, our insight is that cryptocurrencies represent assets and, therefore, require network-wide verification. By changing from assets (i.e., what is owned) to the concept of liabilities (i.e., what is owed), network-wide verification is no longer required before smart contract execution. Secondly, our insight is that modern identity management no longer requires central governance. Identities can be stronger than just public keys: strong identities do not necessarily violate smart contracts' promise of a trustless context that enables fairness and cooperation. However, due to the application of these concepts, green architectures are necessarily subject to new design constraints. In this chapter, we derive these constraints for systems and their architectures in order to make use of novel green concepts. Thereby, these resulting novel systems can be applied to meet all of the functional requirements of smart contracts to modernize governance and financial infrastructure without the operational risk of being banned by governments due to environmental concerns.

This chapter defines a novel paradigm for smart contract execution called Green Smart Contracts (GSCs), which is based on concepts observed in proposals for the Web3 ecosystem and executes smart contracts using a local-first approach. The overall contribution of this chapter is the definition of **a novel architectural paradigm for local-first smart contract execution**. We envision our paradigm challenging Proof-of-Work blockchains as the de facto standard for execution of smart contracts, that currently require network-wide verification for each contract invocation. Our aim is to allow both system and application designers to leverage a greener alternative for contract execution. Our work offers the following contributions:

- We identify the main problem of smart contract execution and the concepts of relaxed consistency, strong identities, probabilistic consensus, and liabilities, which allow for greener smart contracts (Section 5.2).
- We derive the constraints for system architectures to leverage the four concepts for greener smart contracts (Section 5.3).
- We design a greener smart contract prototype solution (Section 5.4).
- Our simulation shows the necessity of consensus for a widely-used real smart contract and the amount of time that nodes work with a cache that is inconsistent with the global hash fabric (Section 5.5).
- Through further experiments, we show that inconsistencies in the hash fabric can be effectively resolved for multiple use cases and we create a model for $CO_2$ emissions that shows when our approach becomes a greener alternative to existing solutions (Section 5.6).

## 5.2 Concepts to decouple smart contracts from Proof-of-Work chains

The problem that smart contracts face is that they inherit a large ecological footprint by being tied to Proof-of-Work blockchains. We look for an execution model that can make use of recent trends in the blockchain ecosystem to address this problem. We present four concepts that can be leveraged for greener smart contracts, and we explain how they can be leveraged.

**Concept 1: Relaxation of the consistency model.** Weakening the consistency model will improve throughput of a system. In practice, this is why systems that weaken their consistency model to allow for independent block updates (like Directed Acyclic Graphs) have shown higher transaction throughput [15, 136]. However, network-wide consistency cannot be weakened to the point of being eliminated. The order of executed operations matters for smart contracts [154, 228]: users' reads and writes to smart contracts are interdependent. Thereby, completely forfeiting consistency enables front-running attacks through information hiding [75]. Nevertheless, basing the consistency model on application-defined locality is a winning strategy [51, 61], especially if a small group of nodes has additional influence over a data structure's contents [150]. Exploring the application of a weaker form of consistency for smart contracts remains promising.

*Examples of solutions that leverage weak consistency are Avalanche [191], Hashgraph [17], and the Tangle [182].* The commonality between these solutions is that they use Directed Acyclic Graphs that see their transactions interlinked based on some form of locality. Locality is based on "transactions" for Avalanche, "actors" for Hashgraph, and "sites" for the Tangle. We propose taking the concept of relaxing consistency based on locality to the extreme and investigate local-first consistency.

**Concept 2: Using strong identities to detect forks.** Individual countries and the European Union are creating passport-level identity solutions for use in the blockchain ecosystem, known as Self-Sovereign Identity solutions [14, 212, 215]. If strong identities were used for smart contracts, there would be no way for users to interfere with contract operations (i.e., "writes") of other users. However, despite a lack of writing contention, even when grounded in natural persons, identity does not guarantee validity. Secondly, a strong identity does not imply any special permissions (like in a permissioned blockchain). A strong identity is not necessarily a trusted identity. A well-identified user may still produce a conflict with itself (e.g., to fool other users and through bugs), which is the classic blockchain forking problem and still needs consensus. Nevertheless, strong identities and public key infrastructure have been shown to greatly improve the detection efficiency of information that users attempted to hide [134].

*Examples of solutions that leverage strong identities are Corda [101], Ebay, and Uber.* Corda proposes ownership of contract applications based on public keys and the usage of identity to assign legal weight to documents on its ledger. Identity is leveraged both for privacy (not all nodes have to know of all transactions) and efficiency (not all nodes need to process all transactions). Ebay and Uber have a centrally governed platform that provides strong user identities, which are a legal requirement, vital for employee management (Uber), and required for the selling of goods (Ebay) and services (Uber) between their respective users. In short, for Ebay and Uber strong identity is used for accountability. We investigate the application of strong identities to gain both efficiency and accountability without central governance.

**Concept 3: Using probabilistic consensus for smart contract invocations.** In our case, we define probabilistic consensus as a form consensus that reaches probabilistic finality of decisions (i.e., decisions may be overturned). A probabilistic approach achieves high throughput and hardens its probabilistic guarantees over time [139]. Using a small-world assumption, detection of violations of agreements made through probabilistic consensus becomes more efficient with witness and audit protocols [44, 95]. Just like for the consistency model, the highest throughput is achieved when the *locality* of the data that consensus is formed over is close to the nodes that are selected to form consensus. Practically, the consensus mechanism should be tightly coupled to the consistency mechanism. For instance, there is a large increase in throughput for smart contracts if locality of Ethereum is optimized, known as "sharding" [221].

*Examples of solutions that leverage probabilistic consensus are Avalanche [191], Bitcoin [164], and Ethereum [240].* One of the key innovations of Bitcoin is its probabilistic consensus, known as Proof-of-Work, requiring no communication between nodes in order to reach consensus (simply deciding on the longest known chain). Ethereum's "sharding" further leverages locality and adopts a model of communication between cliques of nodes. Avalanche even proposes "metastable consensus", relying on majority votes in overlapping localities. We note that probabilistic consensus, especially when exploiting locality, is mostly focused on making *any decision*, which is not necessarily the *best decision* for some use case or application. Therefore, we enable the use of these probabilistic consensus mechanisms but we do not pick a single mechanism, in order to remain use case agnostic.

**Concept 4: Postponing consensus using liabilities.** Liabilities can materialize as tokens in contracts and can represent many different things, like assets, trades, and loans, to support a token economy [231]. An example of a liability is payment through credit, where a user clears a transaction regardless of the user's balance, versus payment through debit, where the user must have the required sum beforehand. In other words, liabilities support systems that depend on authorization instead of ownership. However, even though liabilities can represent assets, to avoid double-spending of assets (like currency) a system requires some form of network consensus. Nevertheless, consensus is then only required after executing the smart contract and it can be postponed (or possibly even avoided) for use cases that depend on liabilities.

Liabilities stand to change the nature of a token economy, when used in combination with the concepts of probabilistic consensus and weak consistency. Instead of serving one network-wide token (e.g., a token that serves as a cryptocurrency), a unique token can be used for the locality of a contract that does not require the whole network to verify it. Thereby, every contract would have its own exchange rate against currency from other contracts (akin to exchanges between fiat currencies). Therefore, the vision of a general token economy, or digital currency, is transformed into one that exists only in the multiple localities of nodes: token microeconomies.

*Examples of solutions to leverage tokens are Ethereum Request for Comment 20 (ERC-20) tokens [230], non-fungible tokens (NFTs) [149], and Basic Attention Token [148].* The ERC-20 standard exists to capture fungible tokens and the NFT standard was made for non-fungible tokens (i.e., assets). Both standards operate from smart contracts on the Ethereum blockchain and tokens that derive from them can represent the breadth of use cases for digital liability and asset management. For example, the Basic Attention Token is an ERC-20 token that uses a user's ad viewing time as the basis for its value. Instead of treating tokens as something to be implemented in a smart contract, we investigate the treatment of tokens as the primitive to power smart contracts.

**Combining concepts.** Any combination of the four concepts, that we have previously presented, can be leveraged to adapt a hash fabric to a given application domain.

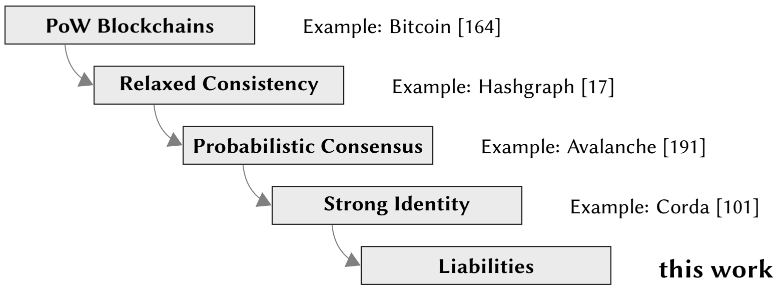


Figure 5.2: The addition of concepts to Proof-of-Work blockchains and examples of solutions that implement all concepts up to that point.

However, these concepts can also be viewed as the evolution toward individual accountability and decentralization of governance to replace network-wide verification. When viewed in this manner, shown in Figure 5.2, we see the communication costs of solutions lessen as individual accountability is increased. As communication costs are tied to the environmental impact of blockchains [12], it follows that the concept of liabilities is the next logical step to investigate, which we do in this work.

The four concepts we present are highly reminiscent of digitized governance and financial infrastructure in the physical world and, therefore, applicable to digitize these processes. For example, a person may use their physical credit card (an identity) to withdraw physical money (a liability) at a physical bank (a node). Of course, to limit risk, the bank will disallow the person to withdraw more money than their credit card limit allows. If the transaction succeeds, banks will engage in clearing and settlement with each other (logic that needs consensus) and record the overall exchange of money between banks (consistency) in their ledgers. Another example is the casting of votes: a person may show their passport (an identity) to cast a vote (a liability) at government office (a node). The different offices then check whether duplicate votes have been cast (logic that needs consensus) and tally the votes (consistency) to record election results.

## 5.3 Design constraints to enable a local-first approach

We now determine to what extent the concepts presented in Section 5.2 can be applied and to what extent their application changes the smart contract execution model. In order for a system to integrate these concepts into a local-first system model (Figure 5.1), we derive the design constraints which we use to create a prototype solution. The findings are summarized in Table 5.1.

Any system that executes smart contracts requires incentive compatibility. In general, no rational user performs more work than necessary. Inherently, a decentralized system like a smart contract system is prone to freeriding, which must be alleviated by implementing either a payment or a reciprocity scheme [81]. Traditional blockchains opt for the former, requiring payment for both proposing—and interacting with—a contract. However, the payment approach provides an unfortunate link between contract execution and payment through cryptocurrency, coupling the low throughput of cryptocurrencies to contracts. Therefore, the dependency on payments can only be broken by using reci-

Table 5.1: Differences between GSCs and smart contracts that use traditional (PoW) blockchains.

| **Property** | **Smart Contracts** | **GSC** |
|---|---|---|
| *incentive* | payment | reciprocity |
| *fork detection* | majority | probabilistic |
| *impartiality* | majority | randomness |
| *use case* | assets | liabilities |
| *finality* | probabilistic consensus | configurable |

procity (its implementation is discussed in Section 5.4), leading to *Constraint 1: Green smart contract execution and dissemination should only depend on reciprocity between users, not on payment.*

A fork detection mechanism requires scalability with respect to the number of users. Smart contracts may have a large volume of interactions, e.g., up to 20 000 interactions per day have been observed [180]. Of course, multiple contract invocations fit inside a block but Proof-of-Work blockchains both have a limited number of blocks per day (e.g., Bitcoin roughly sees one block per 10 minutes) and require the majority of the network to observe and accept each newly proposed block. To this end, to overcome the limitations of Proof-of-Work, relaxations to the consistency and consensus models of blockchains have been proposed (Section 5.2). However, these relaxations change the nature of the smart contract invocations, which can no longer be assumed to be finalized through consensus but should be assumed to be tentative and part of an ephemeral state. Therefore, in order to leverage these novel probabilistic approaches, our second constraint is formulated as *Constraint 2: Green smart contract execution should build on probabilistic fork detection, not consensus.*

A green smart contract system should ensure that the processing of invocations remains impartial to their contents. Traditionally, blockchains assume that the majority of nodes in a network is sufficient to quell any individual nodes that are partial to the contents of invocations (e.g., incentivizing the acceptance of valid invocations with a block-mining bounty in Bitcoin [164]). However, issues with impartiality may arise even when the majority of nodes in a network is used [203]. Furthermore, depending on the explicit involvement of a majority of nodes conflicts with Constraint 2. Instead, more recent proposals trust in the random selection of nodes to ensure impartiality. For example, Verifiable Random Functions have been proposed to elect verifiably random quorums [159] and randomly selected nodes (witnesses) can be used for fault detection [95]. For invocation processing to remain impartial, without violating Constraint 2, we impose *Constraint 3: Green smart contracts should use a random selection of nodes to ensure impartiality, not a majority.*

Greener smart contract systems can be decoupled from cryptocurrencies. Cryptocurrencies inherently require a system for asset management. To solve "double-spending" of the assets (i.e., transferring ownership of a single asset to more than one user), a form of consensus needs to be used. To avoid assets in their entirety, a liability-based interaction model (Concept 4) can be used to power smart contracts. By switching to liabilities as the underlying primitive, the system model of smart contracts (Figure 5.1a) necessarily changes into that of a local-first approach (Figure 5.1c). Asset management now becomes a part of the application layer instead of the underlying substrate for contracts. We explore the resulting solution space by imposing *Constraint 4: Green smart contract execution should be based on liabilities, not on assets.*

A greener smart contract system can exploit a relaxed consistency model. For example, even Bitcoin uses a gossip network to share contract invocations (and blockchain blocks in general) between nodes and later forms consensus on the finality of transactions based on the longest chain of blocks [164]. By explicitly separating the consistency mechanism from the consensus mechanism, recent works have shown the benefits of immediate availability of data in the system, fewer exchanged messages, and a higher throughput [9, 196]. However, the downside is that the invocations of users may not be deemed valid on the application layer at a later time and they may be rolled back. For Bitcoin, the probabilistic finality of transactions of six blocks (about one hour) [191] is sufficient for digital currency. However, the finality requirement may change depending on the application layer. For example, the Corda whitepaper [101] argues that a PDF document is binding even if it is not even on a blockchain, as long as it was signed by an authority. Therefore, in order to remain application agnostic, we postulate the following *Constraint 5: Green smart contracts should allow for configuration of their consensus mechanism, not provide a single fixed mechanism.*

## 5.4 A design for locality-based "green" execution

A greener architecture design that makes use of the concepts that we have presented is different from traditional blockchain architectures. We present the architecture of our GSC prototype, that satisfies the design constraints presented in Section 5.3. Our prototype architecture has five components: (1) a *hash fabric* storing smart contracts and their operations as an immutable history, (2) a *consensus* component to detect and resolve forks, (3) a *runtime* for users to interact with contracts, (4) a *virtual machine* that runs the code and operations captured in the hash fabric on every node, and (5) a networking protocol for *contract discovery*. We now explain how these components interact and we discuss their necessity, with Figure 5.3 as a visual reference.

**The hash fabric** is the central component of GSC architectures, persisting (i.e., both storing and sharing between users) the content and operations of smart contracts (that run in the virtual machine). The hash fabric replaces what would traditionally necessarily be a single "main" (block)chain. For instance, traditionally, in Ethereum all executed user code ends up in one chain. In contrast, due to weaker consistency and consensus models, the hash fabric can represent a data structure that is no longer a single chain. For example, next to a traditional single chain, the hash fabric may also use a mesh or a Distributed Acyclic Graph. Nevertheless, the data structure that is used by the hash fabric should form an immutable history.

As users interact with GSC architectures through an interface, they generate so-called *proposals* to modify the underlying data structure. These proposals capture and hide the semantics of the smart contract information in the hash fabric. Proposals are shared between users using a consistency mechanism of the hash fabric. The primary function of

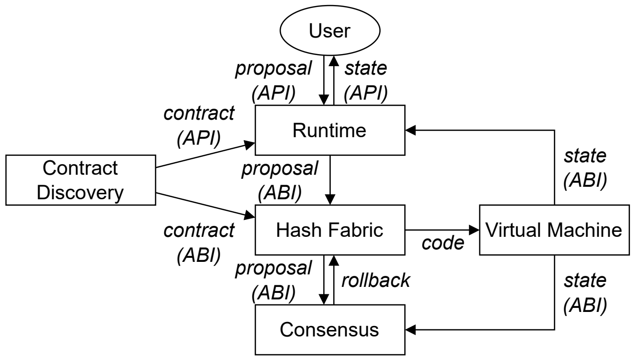


Figure 5.3: Main components of GSC architectures and their interactions.

the consistency mechanism is to synchronize new proposals with the network and to apply received proposals. Secondarily, the consistency mechanism may also be forced to change the data structure itself, a rollback that invalidates proposals in the hash fabric, in case of consistency violations. In short, proposals lead to the temporary—and possibly inconsistent—states that are typical for a local-first approach.

Our prototype defines simple push and pull gossip messages to share proposals in network overlays. Each proposal contains fields for its *code*, the *consensus round* it belongs to, its *proposal type* (i.e., contract creation, operation, or rollback), its *base address* in the virtual machine, and its *block number* in the hash fabric. Consensus rounds are necessary when the hash fabric needs to decide between inconsistent proposals using a consensus mechanism, which we describe shortly.

Figure 5.4 shows an example of how our prototype transforms proposals, received through a network overlay, into blocks in its hash fabric. The first block contains the contract creation proposal, followed by interactions with the contract. This first proposal (*Proposal 1*) places its contract code at address `0x00000000` in the virtual machine component, claiming block number 1 and participating in consensus round 0. Proposals are rejected if they are proposed for a consensus round that has already finished. An arbitrary consensus mechanism may decide to accept or reject blocks that are proposed in a certain consensus round. Of course, one of the blocks necessarily has to be rejected if the order of applying the proposals (*Proposal 2* and *Proposal 3*) leads to different states. However, a new ordering may be chosen in a consensus round, like *Proposal 3* being placed in a new *Block 3* in round 1.

**The consensus** mechanism is tasked with evaluating the structure and semantics of the hash fabric to select the *dominant* history from the valid histories captured by the hash fabric. For example, one may select the “longest chain” as the dominant history (known as *Nakamoto consensus* [241]) from multiple forks of valid blocks in a blockchain. Though forming consensus on an entire history is certainly possible (e.g., in a relational database or a simple log of executions), it is more efficient to only form consensus on new entries in an append-only log as the data structure grows. Within the scope of this work, probabilistic consensus is considered (Constraint 2), which may lead to multiple conflicting

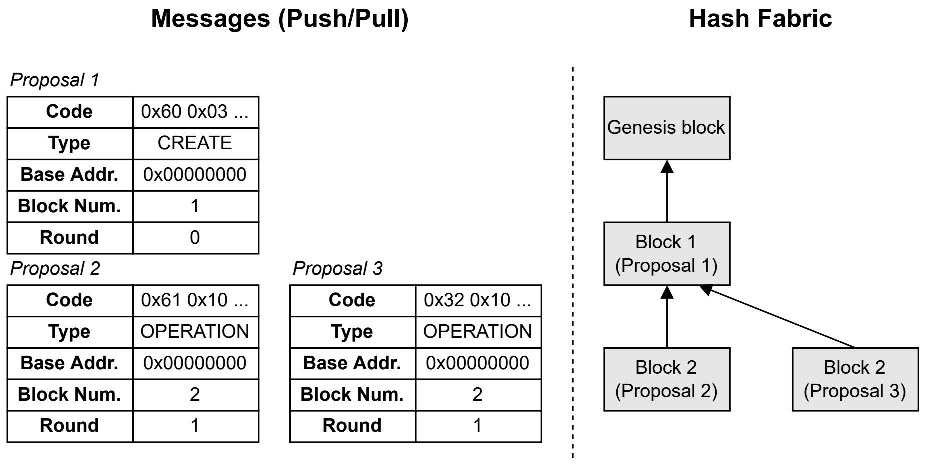


Figure 5.4: How proposals (push/pull gossip messages) are captured in the hash fabric. Depending on the proposals’ contents and the chosen hash fabric, the depicted blocks may cause a fork or form a legitimate state.

histories. The consensus mechanism determines the currently valid proposals and their corresponding causal history (i.e., the “head of the chain” in traditional blockchains).

Not all contract interactions require consensus (e.g., inspecting the value of a variable in a smart contract’s state). What interactions do need consensus can be derived through the hash fabric and the virtual machine. Firstly, in the case that the hash fabric fails to apply a proposal (e.g., when two proposals define the same block number when using a single chain) consensus is needed. Secondly, in the case that the virtual machine fails to apply a proposal (e.g., when two proposals write a different value to the same memory address) consensus is once again required. In both cases, consensus is used to select a single dominant history.

What proposals a user is required to store depends on the chosen consensus mechanism. However, what users end up storing also depends on the trust between users. For example, in theory, Bitcoin requires that all nodes wishing to add blocks to a blockchain store the entire longest chain. In practice, “light nodes” may store only a subset of all blocks and trust in other nodes that store the entire chain [88]. Within the scope of this work, we acknowledge that more efficient storage schemes exist that exploit reciprocity and randomness (Constraints 1 and 3) and trust, e.g., Timely Sharing with Reputation Prototype [216], and we assume that they are leveraged by users to obtain the necessary data to form consensus.

**A runtime** is needed for users to create contract code. This is a common approach, for example found in Bitcoin and Ethereum [250], to make contracts version-independent and portable. The hash fabric only provides the history of operations on the contract. From its history, the runtime derives the current *human-readable state* of a contract to present to the user. The state is calculated by applying the API abstractions to the result of executing the operations captured in the dominant history of the hash fabric.

The GSC architecture compiles contract code written in a Domain Specific Language.

In our prototype, contracts are written in the Solidity language. The creation of a new contract causes two proposals, one for the hash fabric's language API and one for its compiled equivalent, the application binary interface (ABI). We make the distinction between the source contract and the actual compiled contract as the high-level language implementation may not produce the same compiled code for different virtual machines. However, the API is still practically necessary for human interaction, as the ABI exposes users to low-level details that are difficult to work with [206].

**A virtual machine optimistically locally executes** all compiled (ABI) code that is received through a user's network. Essentially, this is no different from how blockchain solutions normally offer their transactions to their respective virtual machines [165, 221]. However, in contrast to normal execution of virtual machine instructions, GSC systems explicitly maintain the state of all forks of the hash fabric (Constraint 2). In our prototype implementation, we use the Ethereum Virtual Machine (EVM), which maintains a "main chain" to execute contract code on (that is normally stored in blockchain blocks) [102]. To execute code from any arbitrary preceding state, we select a previous state as the main chain and execute code from that point.

New proposals, regardless of their semantics (both contract creation and operations on contracts), consist of virtual machine instructions that have been compiled from an API call. The virtual machine is responsible for retrieving the state from a given header and applying given instructions to potentially persist a new state and header and—in case of state changes—may return code to share with other users, just like in Ethereum [102].

The biggest difference between GSC's proposals and traditional smart contract execution is the lack of currency (e.g., Ethereum's "gas" [102]). In GSC systems, depending on the chosen hash fabric and consensus mechanism, code is not necessarily pushed to strangers that do not have an intrinsic benefit to run code (i.e., users that have no causal relationship). Therefore, there is no need for currency to power a contract. However, our prototype does still use the currency mechanism of the underlying EVM to protect against infinite loops [102]. To do so, every execution is supplied with an ample amount of artificial currency (equivalent to several millions of dollars).

**Contract discovery**, consistency, and consensus can be based on locality for GSC systems, and blockchains in general [229], to make a system scalable. Whether or not an application allows for this depends on the system configuration (Constraint 5). In other words, if desired, contracts and their interactions may only be discovered by third parties when they are interacted with. The ability to exploit locality depends on the second concept for GSCs: strong decentralized identities. These strong identities make it possible for users to determine that a particular contract belongs to the user presenting it, beyond reasonable doubt (i.e., using cryptography [57]). The locality-based approach ensures that network-wide consensus is not strictly necessary (but can still be applied) to establish ownership of—and interactions with—a contract. The absence of network-wide consensus can make GSCs more energy-efficient ("green") than traditional blockchains [12].

What constitutes locality may differ from application to application and governs the permissible underlying networking technology [186]. For example, when a the GSC paradigm is used to manage contracts that govern physical systems, networking technology like Bluetooth or Wi-Fi Direct may suffice. Over the Internet, this locality may be a common application-driven interest of multiple peers (e.g., a particular file in Bittorrent).

## 5.5 Consistency violations in a real-world smart contract

One of the key concepts of local-first software is that humans do not produce many “conflicts” when interacting with each other [118]. In smart contracts these conflicts would materialize as forks. We explore the need for fork resolution in Green Smart Contracts by running a real-world trace of an Ethereum smart contract, focusing on the time to resolve inconsistencies between nodes in a network. The smart contract used in these simulations is from the “CryptoKitties” game, which allows users to generate and trade cartoonish pictures of cats. Conflicts can arise on a high abstraction level when users attempt to “breed” with each other’s cats or transfer their ownership, but also on a lower level when the contract writes to a shared memory address in the virtual machine.

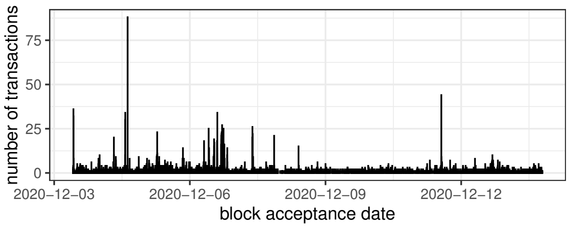
### 5.5.1 Data set

The CryptoKitties game consists of five smart contracts: the “Core” contract, “GeneScience” contract, “Offers” contract, “SalesAuction” contract, and “SiringAuction” contract [110]. The latter three contracts are used to support the exchange of cat pictures and the “GeneScience” contract is used to support generation of entirely new pictures. Apart from these supporting contracts, the core game logic is implemented in the “Core” contract, which we focus on. We query `bitquery.io`, a website for blockchain analytics, for the first 5 839 blocks of the CryptoKitties “Core” smart contract (defined at address `0x06012c8cf97bead5deae237070f9587f8e7a266d`). These blocks contain the first 9 769 transactions, not evenly spread over the blocks (shown in Figure 5.5a), between 543 unique addresses. The transactions contain twelve distinct contract calls defined by the CryptoKitties contract, for which we give the number of occurrences in the data set in Figure 5.5b.

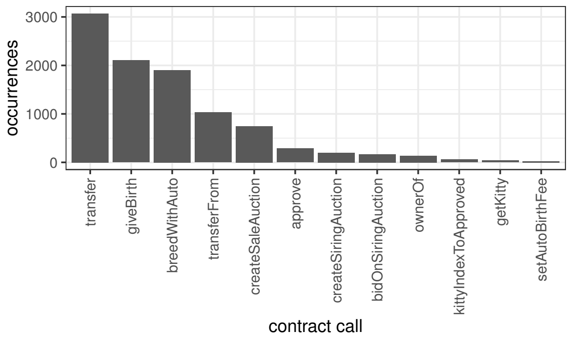
Due to privacy concerns, `bitquery.io` omits the actual argument values of contract calls. We do not attempt to circumvent this omission, but we replace the arguments with random values. Of course, the referenced memory addresses do not change and conflicts due to concurrent writes to these addresses are preserved. However, the random values may lead to different logic being executed. For example, an “if”-statement may only trigger a write to a memory address if a value meets a certain condition. Consequentially, the smart contract may execute different instructions in the EVM. Therefore, the state of the smart contract in our simulation is not expected to be equal to that of the real CryptoKitties contract on Ethereum. To mitigate this limitation, our simulation does not keep track of the instructions that were invoked but rather the number of times that the EVM was called to execute any set of instructions.

### 5.5.2 Setup and methodology

Our data set consists of addresses calling methods on the CryptoKitties contract code, mapped to a network of nodes that propose these method calls to a consensus mechanism. We create a node for each address in our data set. We use a fully connected network of nodes and we fix the latency for all messages between nodes to $50ms$. This choice is rooted in the network protocol of Bitcoin, in which up to 1000 peers can be discovered in a single message without further communication, well more than the 543 users in the data set [194]. Similar to a real blockchain, we use a list of blocks as the data structure for the hash fabric and we adopt the oldest-known value to reach consensus.



(a)



(b)

Figure 5.5: The number of transactions per block (a) and the number of occurrences per contract call type (b) in the CryptoKitties data set.

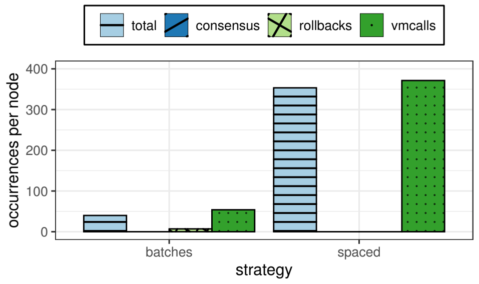


Figure 5.6: Average number of occurrences of the four metrics per node for the CryptoKitties data set replay.

Our data set is replayed using a *batches* strategy that introduces transactions using the data set timestamps and a *spaced* strategy that introduces transactions with a spacing of 250*ms*. For the former strategy, every node in our network proposes its transactions using the timestamps defined in the data set. The batches strategy leads to nodes that have transactions in the same block attempting to claim the same block number in our list data structure. Therefore, if any calls conflict, the consensus mechanism must select one of the conflicting calls. For the spaced strategy, consecutive contract calls in the data set are spaced by 250*ms*, causing all preceding transactions to be finalized before new ones are initiated. Thereby, the second strategy maintains the order of the original transactions, but not their timing. The intention of these strategies is to show the difference between real contract interactions and artificial workloads.

We capture a total of five metrics. The first four metrics are occurrences relative to the node count to expose any non-linear message complexity. Firstly, we keep track of the total number of received messages. Included in the total number of messages are the messages that may be needed to form consensus, to push messages, and to pull specific messages (due to rollbacks, a single message may need to be pulled more than once). The messages needed for consensus are also counted as the second metric, which is always zero in this simulation but they will play a role in Section 5.6. Thirdly, the number of rollbacks of (part of) the hash fabric is recorded. We also keep track of the required EVM calls, as opportunistic execution implies higher CPU loads, which may be restrictive to CPU-limited devices. The fifth metric is the time that nodes are in an inconsistent state, which we call the *convergence time*. The convergence time is the time between when the first consensus message is received for a particular consensus round (which corresponds to its block height, see Figure 5.4) and the time at which the last consensus message is received.

### 5.5.3 Results

We first discuss the metrics related to the number of messages, presented for both of our strategies in Figure 5.6. As mentioned before, no consensus messages are sent due to the choice of consensus mechanism. Even so, the replay of a real data set leads to relatively few rollbacks, with 7 rollbacks for 54 EVM calls for the batches strategy and no required rollbacks when the hash fabric is finalized between contract invocations. Therefore, we

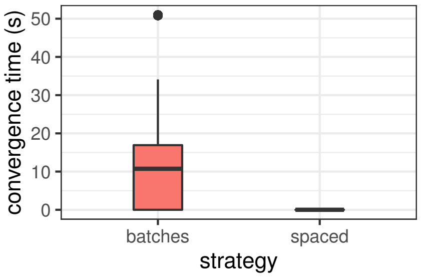


Figure 5.7: Box plot of the time until contract interactions have converged, when nodes interact concurrently (“batches”) and when interactions are spaced out (“spaced”).

conclude that real smart contracts do not necessarily have a lot of conflicting interactions. Regarding the different strategies, we see much fewer messages and EVM calls for the batches strategy as opposed to the spaced strategy. This absence of messages occurs due to conflicting interactions not being forwarded and applied in the network, which could be resolved by a retry mechanism.

In Figure 5.7, we show the time it takes for contract calls to converge for both of our strategies. Our results show that the batches strategy causes the convergence time to reach into the order of several seconds, up to almost a minute. This is explained by nodes attempting to claim the same list index in the list of blocks, necessitating some other non-trivial consensus mechanism to select only one of the conflicting calls for the index. Our second observation is that the convergence time goes to 0 seconds if each call is proposed $250ms$ after the last call finished (the spaced strategy). The chosen spacing allows each node to receive the previous contract call and propose a new contract call with a list index that does not conflict with the preceding transaction. Clearly, when a smart contract has very little (or no) conflicting calls, no consensus mechanism is required for a consistent state for all nodes.

### 5.5.4 Modeling conflicts

Our results support that little to no conflicts in the consistency layer enable a system in which network-wide consensus is unnecessary. However, it may be unrealistic to assume that no conflicts occur. For example, out of the proposed transactions to Ethereum an estimated 3% fails [169]. In contrast, in our CryptoKitties replay we observed that 13% of the proposed transactions fails due to consistency violations when using the batches strategy. Furthermore, in our replay, the resolution of consistency violations is based on the time a call was made. Currently, we do not know of technology that allows timestamps to be (unconditionally) verified. Therefore, we do not have evidence for any system being able to exist to benefit from that finding.

As, to our knowledge, systems that forego consistency violations in the hash fabric do not exist, some form of consensus mechanism is required. Supported by the results of Section 5.6.3, we believe using identities to provide conflict resolution is the next-best option for real smart contracts. However, as mentioned in Section 5.3, using the GSC paradigm and depending on identities is not compatible with all use cases that smart contracts are currently serving. For asset-based use cases, the consensus model of traditional smart contract execution is still the only viable option.

We use a simple model to validate the use of our local-first approach. Given the period of time an invocation is vulnerable to conflicts $f$, the number of new invocations per second $r$, and the probability of conflict between two transactions $p$, the expected number of conflicts experienced by a single transaction is $p \times r \times f$. Our results yield $p = 0.13$ in the worst case of the *batches* strategy and our dataset has $r = 0.011296$ invocations per second. Therefore, invocations succeed without conflict more often than not (when $p \times r \times f < 0.5$) if the time to reach finality for each invocation is $f < 340.49$ seconds. From our results from the *spaced* strategy, we observe that $f < 0.25$ seconds. Therefore, the inequality is satisfied and, at least in the case of CryptoKitties, invocations will succeed without conflict more often than not, validating the use of a local-first approach. In fact, in comparison to the 3% failures of Ethereum, i.e., 0.03 expected conflicts, our approach leads to only a fraction of this, with 0.00036712 expected conflicts when $p = 0.13$, $r = 0.011296$, and $f = 0.25$. Furthermore, given $p = 0.13$ and $f = 0.25$, our approach is applicable up to $r = 15.38$ invocations per second per contract. As the most popular smart contracts receive 20 000 invocations per day [180], i.e. $r = 0.23$, our approach offers 66.44 times the necessary invocations per second.

## 5.6 Resolving consistency violations

Inconsistencies do occur in real smart contracts (Section 5.5), which are exacerbated in a green local-first approach. GCS architectures must leverage a mechanism to decide a dominant history (Section 5.4) and, depending on the use case, different consensus mechanisms may be used. For example, if a single leader is permissible, a publish-subscribe communication pattern can be used. If a completely leaderless mechanism is required, a metastable consensus mechanism like Snowflake [190] can be used. In this section we conduct experiments in order to provide insight into the consequences of deploying a selection of different consistency and consensus mechanisms.

### 5.6.1 Setup

Four different mechanisms are used as the consensus mechanism with our GSC prototype: Raft[2] [171], metastable consensus (similar to Snowflake [190]), a publish-subscribe mechanism, and adopting the oldest-known value. The chosen algorithms represent the breadth of approaches to peer-to-peer agreements and we discuss them in further detail later.

To determine a node count for our experiments, we note that Raft was tested with five servers [171], Snowflake was tested with up to 2000 nodes [190], and the remaining two mechanisms depend on the limits of the communication substrate. We pick a middle-ground of up to 1000 nodes for our experiments and we simulate networks of 10, 100, and 1000 nodes. We again use a fully connected network topology, rooted in the same rationale as in Section 5.5 (even 1000 nodes can be discovered in a single message). We measure the four metrics as in Section 5.5 that pertain to message handling ("total", "consensus",

[2]Specifically `https://github.com/streed/simpleRaft`

```
pragma solidity ^0.5.11;
contract SimpleContract {
     uint value = 0;
     function setValue(uint _value) external {
          value = _value;
     }
     function getValue() external view returns(uint) {
          return value;
     }
}
```

Figure 5.8: A simple Solidity smart contract.

"rollbacks", "vmcalls"). The smart contract shown in Figure 5.8, created by S. Verma[3], is used to run our experiments.

Two different mechanisms for consistency are used. Firstly, the simple list requires all received blocks to have a unique list index. When two blocks attempt to occupy the same list index, the consensus mechanism is invoked. Secondly, a conflict-free replicated data type (CRDT) [202] represents the other extreme for consistency mechanisms and it is the recommended data structure for a local-first approach [118]. The CRDT data structure adds incoming blocks to the current index (a "merge" in a *Sequence CRDT* [166]) until two blocks are found that are order-dependent and, therefore, require a decision from the consensus mechanism (as discussed in Section 5.4). If the consensus mechanism requires the nodes to take an initial vote, every one of them votes for the oldest block it knows of.

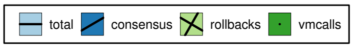
### 5.6.2 Methodology

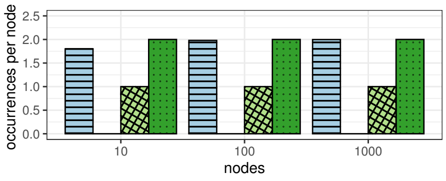
Our methodology consists of introducing a conflict in the simulated network and waiting until all nodes have accepted a new block, resolving the conflict. Our experiment follows three synchronized phases: sharing the initial contract, creating the conflict and waiting for it to be resolved.

The first phase of our experiment consists of sharing the contract code. Blocks are created for two indices: the API code ("block 1") and the ABI code ("block 2"). We then wait for all nodes to receive both blocks, forming consensus according to the consensus mechanism (without conflicts all nodes accept both blocks). We start counting toward our metrics after this first phase has completed.

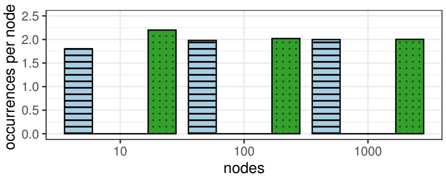
In the second phase we introduce a conflict using two nodes that invoke the API of the contract (Figure 5.8). One node invokes `setValue(11)` and one node invokes `setValue(13)`, which would leave the system in an inconsistent state if both transactions were applied without ordering them (some nodes would have 11 and some nodes would have 13 as the return value of `getValue()`). After introducing this conflict, we resume communication between nodes and end the experiment (the final phase) by waiting for all nodes to accept either value.

[3]https://medium.com/better-programming/part-1-brownie-smart-contracts-framework-for-ethereum-basics-5efc80205413

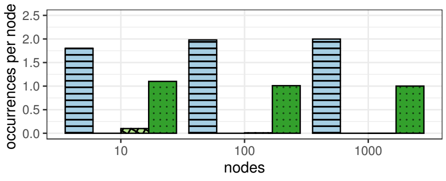




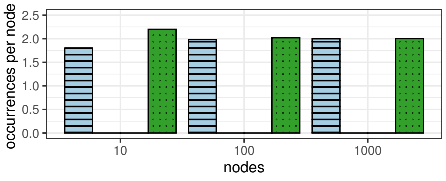
(a) List consistency and the oldest-known value.



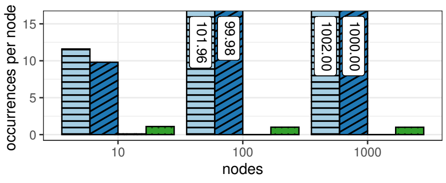
(b) CRDT consistency and the oldest-known value.



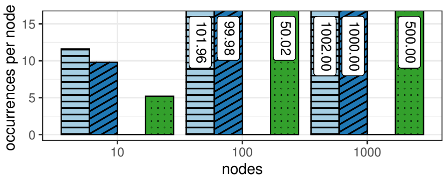
(c) List consistency and a single publisher.



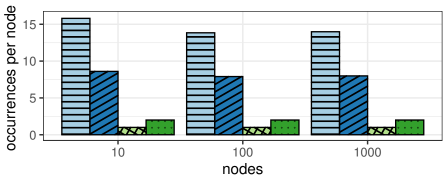
(d) CRDT consistency and a single publisher.



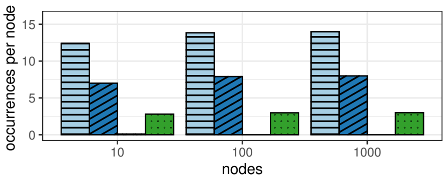
(e) List consistency and metastable consensus.



(f) CRDT consistency and metastable consensus.



(g) List consistency and Raft consensus.



(h) CRDT consistency and Raft consensus.

Figure 5.9: The number of occurrences of the “total”, “consensus”, “rollbacks”, and “vmcalls”, metrics versus the number of nodes, for the eight combinations of consistency and consensus mechanisms. Bars that go beyond the vertical plotting range are labeled.

## 5.6.3 Results

The consensus mechanism of adopting the oldest-known value is discussed first. Every node, except for the two originators, only receives a total number of two messages, the lowest number of messages out of all experiments. For the list consistency (Figure 5.9a) the second message conflicts with the first message, which requires a rollback, leading to two executed EVM calls. For the CRDT (Figure 5.9b) nodes attempt a merge of three possible blocks: `setValue(11)`, `setValue(13)` and the set `{setValue(11), setValue(13)}`. The former two proposals are made by the originators, while all other nodes forward the latter form. The latter form is reevaluated by the originators, leading to the average of just over two EVM calls.

The results for a single publisher, given in Figure 5.9c and Figure 5.9d, are largely similar to the results of the oldest-known value. For the list consistency (Figure 5.9c) the required number of rollbacks in the system is now equal to one. Only one of the conflicting messages is published by the originating identity and the other one is rolled back on the node that produced it. For the CRDT consistency layer, the conflicting messages can still exist with the same identifier and this is only later corrected by the consensus mechanism.

Metastable consensus is the first non-trivial consensus protocol, shown in Figure 5.9e and Figure 5.9f (note the change in the vertical axis range). The number of messages increases as the number of nodes increases, which is a direct consequence of the design choice to avoid a leadership election protocol. Due to state deduplication in the EVM calls and the consensus layer being able to freely add and remove messages for a given index, the EVM call count is inflated.

Raft is the final consensus mechanism that we evaluate and its results, given in Figure 5.9g and Figure 5.9h, show a decrease in message count as opposed to metastable consensus, largely due to its leadership election protocol [171]. Electing a single identity to resolve a conflict after its detection requires fewer messages, as opposed to having all nodes converge to a single value over time, as with metastable consensus. However, when then number of messages is a concern, having a pre-established leader (i.e., a single publisher) is still superior.

All of the combinations of consensus mechanisms and consistency mechanisms successfully resolve conflicts, though their environmental impact may be different. The choice of CRDT consistency mainly causes a higher number of EVM calls and—from the perspective of environmental impact—is therefore less desirable due to the higher CPU utilization. More traditional consensus mechanisms are known to have a higher environmental impact [12]. Therefore, the choice of consensus mechanism is key to make GSCs environmentally friendly. Applications should strive to make use of the concepts of GSCs as much as possible, if they wish to reduce their environmental impact.

From the user's point of view a CRDT solution may still be desirable. Even though the higher CPU utilization leads to more environmental impact, this would mean that the user is not confronted with rollbacks. A rollback is visible in the interface of a user's application, whereas higher CPU usage is not.

## 5.6.4 Model for environmental impact

We now define a model for environmental impact by estimating $CO_2$ emissions, based on our observed messaging behaviors and their associated expected conflicts. We derive

Table 5.2: Behavior of our four metrics to resolve a conflict between two contract invocations given the number of nodes (n), for the measured consistency and consensus mechanisms.

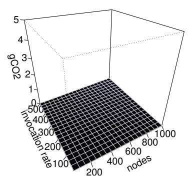
| Consistency | Consensus | Total Msgs. | Consensus Msgs. | Rollbacks | EVM Calls |
|---|---|---|---|---|---|
| List | Oldest-Known | $2n$ | 0 | $n$ | $2n$ |
| CRDT | Oldest-Known | $2n$ | 0 | 0 | $2n$ |
| List | Publish-Subscribe | $2n$ | 0 | 1 | $n$ |
| CRDT | Publish-Subscribe | $2n$ | 0 | 0 | $2n$ |
| List | Metastable | $n^2$ | $n^2$ | 0 | $n$ |
| CRDT | Metastable | $n^2$ | $n^2$ | 0 | $\frac{1}{2}n^2$ |
| List | Raft | $14n$ | $8n$ | $n$ | $2n$ |
| CRDT | Raft | $14n$ | $8n$ | 0 | $3n$ |

the behaviors of our four metrics from our results, shown in Table 5.2. The behaviors we define are a simplification of our actual results as they change with the number of nodes. For example, though we model the total number of messages for CRDT and Raft as $14n$, our results are actually $12.4n$ for 10 nodes, $13.84n$ for 100 nodes, and $13.98n$ for 1000 nodes. Our defined behaviors become more accurate as the number of nodes grows. As related work suggests that the exchanged number of messages of a method is the primary driver for environmental impact [12], we focus on three functions to describe the distinct messaging behaviors (Table 5.2): $b_1(n) = 2n$, $b_2(n) = 14n$, and $b_3(n) = n^2$.

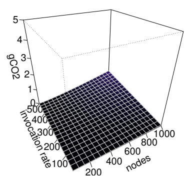
For each conflict that is introduced, its total number of messages $b_i(n)$ are added to the aggregate total number of messages for the entire network. In Section 5.5.4, we determined that the number of conflicts is given by $p \times r \times f$. Therefore, the aggregated total number of messages is given by $p \times r \times f \times b_i(n)$. For fully connected networks, $f$ is simply the maximum latency $l$ between nodes, while random graphs may have $f = O(log(n) \times l)$ and linear topologies have $f = O(n \times l)$. By assuming, without loss of generality, that $r$ is measured per time unit $l$, we can eliminate $l$ from our equations. Thereby, we obtain the following three functions for the aggregated total number of messages: $O(m_1) = p \times r \times O(1) \times b_i(n)$, $O(m_2) = p \times r \times O(log(n)) \times b_i(n)$, and $O(m_3) = p \times r \times O(n) \times b_i(n)$.

To tie the messaging behavior to $CO_2$ emissions, we require estimates for the energy expenditure of transmitted messages and the emissions of the expended energy. Firstly, a realistic upper bound for the energy expenditure per gigabyte is $0.2kWh/GB$ [50]. Secondly, though the $CO_2$ emissions vary per region, we use the United States average of $0.603kgCO_2/kWh$ to model emissions [10]. We use the Ethereum maximum smart contract invocation size of $24kB$ as the message size, assuming that the overhead data needed for consensus is negligible. Given these assumptions, we obtain a result of $0.0028944\ gCO_2\ /message$ to estimate our carbon emissions.

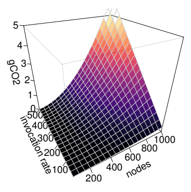
We evaluate our model to calculate $CO_2$ emissions for the nine different combinations of $b_i$ ($b_1$, $b_2$, and $b_3$) and $m_j$ ($m_1$, $m_2$, and $m_3$). To compare between the different behaviors, we fix the value $p = 0.13$ to match our CryptoKitties results. Our results are visualized in Figure 5.10 and they highlight that a low node count or a low invocation rate trivially keep carbon emissions down. However, when the node count and the invocation rate are both



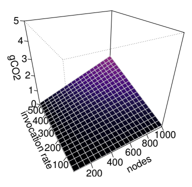
(a) $b_1$, $m_1$



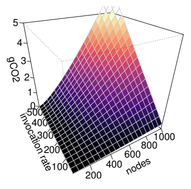
(b) $b_1$, $m_2$



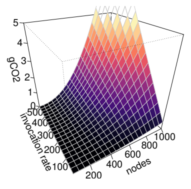
(c) $b_1$, $m_3$



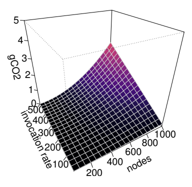
(d) $b_2$, $m_1$



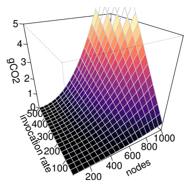
(e) $b_2$, $m_2$



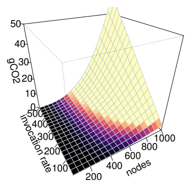
(f) $b_2$, $m_3$



(g) $b_3$, $m_1$



(h) $b_3$, $m_2$



(i) $b_3$, $m_3$

Figure 5.10: Estimated carbon emissions as a function of the invocation rate and the number of nodes, given functions $b_i$ for messaging behavior and functions $m_j$ for the aggregated total number of messages.

increased, the choice of consensus mechanism and messaging topology starts to matter. With a local-first approach of fully connected nodes ($m_1$), the choice of consensus mechanism is hardly an influence. For a more loosely-connected messaging topology ($m_2$ and $m_3$), a fixed publisher or predetermined finalization strategy like "oldest-known" ($b_1$) is preferential. Given a typical peer-to-peer network with logarithmic message propagation time and Raft consensus (i.e., $b_2$, $m_2$), even the worst-performing consensus mechanism with local-first (i.e., $b_3$, $m_1$) has only 47.69% of the carbon emissions at $n = 1000, r = 500$. In the majority of these cases, the Green Smart Contract approach is a greener approach.

To put these numbers into perspective, China produces roughly 12 gigatonnes of $CO_2$ per year. Even in our worst presented case (using 1000 nodes that produce 500 invocations per second with 13% chance of conflict), the Green Smart Contract approach only produces $1.58 \times 10^{-6}$ gigatonnes of $CO_2$ per year. However, if the same configuration is used for a billion nodes with 500 invocations per second, the estimated emissions would be—a much less trivial—$1.88 \times 10^{12}$ gigatonnes of $CO_2$ **per second**. In contrast, our best presented case would produce 0.00013 gigatonnes (130 million kilograms) of $CO_2$ per second for a billion nodes and 500 invocations per second, or 342 times the emissions of China per year.

## 5.7 Related work

Throughout this chapter we have highlighted the individual works that are closely related to the topics we addressed. We now position our contributions on a coarser scale. On the coarsest scale, one can consider using solar panels to power the hardware of nodes [144], which necessitates additional physical hardware, and the trade of emission certificates [40, 105], which requires a secondary market. For a more focused discussion, we regard work that proposes changes to software architecture.

One of the key works that precedes our work and falls within our definition of a "recent proposal" (Figure 5.1b) is the Hedera Hashgraph [18]. The Hedera Hashgraph proposes both a weaker consistency model (in the form of a Directed Acyclic Graph data structure) and a weaker consensus model (which they call "asynchronous Byzantine Fault Tolerance"). Furthermore, the consensus layer is optionally permissioned (i.e., configurable). Whereas Hedera is similar to our Green Smart Contracts in its relaxations, it falls short of going to the extreme of a local-first model and it uses the traditional blockchain model for users to offer interactions to nodes. Our work goes one step further and actively embraces a new paradigm for user interactions with smart contracts.

Our work is closely related to the "local-first software" proposal by M. Kleppmann et al. [118], that mainly focuses on change-based updates to shared data structures. Our work builds on their findings and uses a local-first approach to the consistency layer in smart contracts. However, the work of Kleppmann et al. is mainly focused on how humans interact to form a shared data structure. One of their conclusions is that "conflicts are not as significant a problem" because "users have an intuitive sense of human collaboration and avoid creating conflicts". Whereas we have found that there are certainly less conflicts in a real-world smart contract than in a lab setting (see Section 5.5), with a 13% failure rate, we believe conflicts are a significant problem for smart contract execution.

Smart contract execution is the next evolution after the state machine replication movement (which came after shared memory systems). Even though smart contract execution does away with any trust assumptions between nodes, Byzantine Fault Tolerance (BFT) in its execution model is still shared with the domain of state machine replication. Very few works explore execution without a critical dependency on BFT consensus. Of particular note is Eve [112], proposing state machine replication with tunable fault tolerance to speed up execution. Their proposal is to first agree on an order of operations, then to execute those operations, and lastly to verify the resulting state. In contrast, our work proposes to execute operations opportunistically, verify the resulting state, and then to agree on an order only when a conflict is found. However, just like our work, Eve exposes a design space of "nondeterminism introduced by allowing parallel execution". Our work argues this design space consists of liability-based applications.

Our proposal of making contracts the central point of interaction is closely related to the publish-subscribe communication pattern. However, regarding the execution of program code, most works only consider the publish-subscribe pattern to distribute executable tasks to nodes (for instance, works by M. Sadoghi et al. [195] and by J. Dayal [59]). In their work, L. Jehl et al. explore a publish-subsribe-based state machine model, based on broadcasts in the presence of Byzantine failures [106]. In contrast to the aforementioned work (and in agreement with local-first software), we believe a single publisher (a contract) and its subscribers (the users that interact with the contract) naturally emerge from smart contract use and do not require additional reliable broadcast.

## 5.8 Conclusion

Green Smart Contracts are able to challenge Bitcoin and Ethereum as ubiquitous technology, to serve all applications. This chapter has identified the concept of liabilities, to replace the established model of cryptocurrencies serving as the primitive to support smart contract execution. By depending on liabilities as a primitive, Green Smart Contracts are able to maximally leverage weak consistency and probabilistic consensus to form a novel green "local-first" architectural paradigm for smart contract execution. The requirement of stronger identity management for such architectures can be overcome without violating the context of permissionless and trustless smart contract execution. The benefit of our local-first approach is that the requirements of applications that use liabilities are severely lowered, as opposed to applications that depend on assets. Going forward, smart contract applications should carefully examine their application domain to potentially make use of the communication improvements offered by the Green Smart Contract paradigm. Future digitization and automation can become greener and more performant.

# 6 Conclusion

This thesis has sought to answer what technology is appropriate to enable Self-Sovereignty for a Web3 ecosystem. We considered the three domains of identity, public infrastructure, and shared code, and we introduced four more research questions to help answer our thesis-overarching research question. We summarily enumerate our conclusions, one for each research question, that relate to the necessary technology to enable Self-Sovereign existence of users and Self-Sovereign collaboration between these users. We then conclude if, and how, technology can enable this existence and collaboration for Web3 ecosystems. This chapter ends with the enumeration of open questions, future work, that we have uncovered in the pursuit of Self-Sovereign Web3 technology.

## 6.1 Conclusions

The conclusions of our four supporting research questions are as follows:

1. In Chapter 2 we presented the deployed and matured solution of TrustChain IDentity (TCID) for truly Self-Sovereign identity. Our solution is practical, modular, and has sufficiently low overhead for practical use. We have exposed that Self-Sovereign Identity has been relatively over-explored from the angle of cryptography and under-explored in its network layer. We conclude that it is indeed possible to create a Self-Sovereign Identity solution that addresses even the most stringent use case of serving as a passport analog.

2. In Chapter 3 we introduced the SybilSys mechanism, which avoids Sybils by exposing their lack of network latency diversity. We have shown that SybilSys is capable of significantly increasing the monetary cost of performing Sybil attacks without the need for trust or a centrally governed infrastructure. Using real users, our research has shown that latency can serve as a reliable first-hand metric to avoid Sybils. We conclude that SysbilSys allows peers to avoid Sybils in the network layer using only network latency.

3. In Chapter 4 we have presented our carrier-selection mechanism of Timely Sharing with Reputation Prototype (TSRP) to maintain public infrastructure. These carriers

are peers that store, and share, the data needed to maintain public infrastructure. We have provided a proof that carriers can be selected, using their reputation, in such a way that public infrastructure is maintained. The TSRP mechanism successfully scales the number of copies carried by peers with respect to the reputations of the locally-known peers. Furthermore, we have shown how our mechanism does not necessarily introduce many more copies of data in the network than structured (centralized) approaches would. We conclude that our TSRP mechanism allows public infrastructure to leverage locally-calculated reputations to be efficiently maintained.

4. In Chapter 5 we introduced the Green Smart Contracts (GSC) paradigm for smart contract execution. We have shown how probabilistic consensus mechanisms, weak consistency models, and decentralized strong identities can be leveraged to execute smart contracts. Our local-first approach to the execution of smart contracts significantly decreases the number of messages sent between peers. We derived that our approach leads to a novel model for (digital) Web3 economies that is based on liabilities instead of assets. We conclude that our GSC paradigm allows peers to execute shared code in an environmentally-friendly, and local-first, fashion.

We derive the following high-level conclusions for these domains based on our earlier conclusions:

5. Humans can exist and interact in the digital world in the same Self-Sovereign fashion as they exist in the physical world. We have shown that identity wallet technology can enable even the most stringent use case of serving as a passport analog (Conclusion 1). Even the most notorious attack on peer-to-peer software, the Sybil attack, can be mitigated in a Self-Sovereign fashion, using a mechanism that depends only on network latency measurements (Conclusion 2).

6. Self-Sovereign users can share and maintain public infrastructure to reliably execute program code. We have shown a mechanism that only locally calculates reputations to maintain public infrastructure (Conclusion 3). Program code can even be collaboratively executed between Self-Sovereign individuals while sparing the environment (Conclusion 4).

All in all, this thesis has showcased Self-Sovereign systems for the three domains of identity, public infrastructure, and shared program code execution. Thereby, we have shown what technology can be used to enable Self-Sovereignty for a Web3 ecosystem.

## 6.2 Future Directions

This thesis has provided an exploratory analysis of the solution space of Self-Sovereignty. Therefore, there is still much to be explored beyond what is presented in this thesis. We now provide several promising directions for future work per chapter:

1. In Chapter 2 we presented the architecture of our TCID Self-Sovereign Identity solution. In order to provide anonymity on the network layer, we presented a communication substrate that uses intermediary peers to route data. We motivated

that this communication substrate is a necessity for the Internet. However, future works may explore different substrates that guarantee network anonymity without any additional anonymization measures. Considering the focus on mobile computing devices, future 6G (or even 7G) implementations may serve to reduce the Self-Sovereign Identity stack complexity if anonymity is guaranteed by the substrate. Therefore, the research question that we pose is: *"How can a communication substrate like the Internet be both anonymous and efficient?"*

2. In Chapter 3 we introduced SybilSys to avoid connections to Sybils using only network latency. As our focus was only on network latency, we have left the exploration of other complementary mechanisms out of scope, like those that use social networks to infer if identities are Sybils. Future works may explore coupling other mechanisms to SybilSys to improve Sybil avoidance and the first research question that we pose is: *"To what extent can Sybil-avoiding mechanisms reinforce each other to avoid Sybils?"* Furthermore, Enhanced SybilSys uses a heuristic in an attempt to cause messages to occupy the buffers of routing hardware at the same time, while treating the network topology as opaque. The heuristic could be enhanced by using network-topology-aware methods to guarantee message flow joins and the second research question that we pose is: *"How can the structure of peer-to-peer network topologies be discovered without using data from other peers or third-party hardware?"*

3. In Chapter 4 we introduced TSRP for reputation-based carrier selection to maintain public infrastructure through the replication of records. We have proven that the manner in which reputation is established is inconsequential for record replication. However, the mapping of reputation to a number of records in a system remains unexplored. Therefore, future works should explore both how reputation is calculated and how many copies should be made available in relation to the known reputations. The research question that we pose is: *"In what manners can the carrying, storage and sharing, of data be incentivized using reputation?"*

4. In Chapter 5 we showed how the GSC paradigm can greatly reduce the number of messages in a smart contract system. Our results are based on experiments with applications (and their traces) that were not created for liability-based economies. The impact of creating smart contracts that are specifically made for this use case remains unexplored and the first research question that we pose is: *"To what use cases can smart contracts based on liabilities be applied?"* Furthermore, we have not explored hybrid models for consensus. Within our model, it is possible for different smart-contract calls to form consensus in a different way in a network. For example, some calls may need to be only agreed upon with a local clique in the network whereas other calls may need a majority vote in the entire network. Therefore, the second research question that we pose is: *"How can decision making in program code be characterized to derive its applicable consensus procotols?"*

# Bibliography

## References

[1] Andreas Abraham, Felix Hörandner, Olamide Omolola, and Sebastian Ramacher. Privacy-preserving eid derivation for self-sovereign identity systems. In *International Conference on Information and Communications Security*, pages 307–323. Springer, 2019.

[2] Rafael Accorsi. Automated privacy audits to complement the notion of control for identity management. In *Policies and Research in Identity Management*, pages 39–48. Springer, 2008.

[3] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. Bootstrapping trust in distributed systems with blockchains. *USENIX; login*, 41(3):52–58, 2016.

[4] Christopher Allen. The path to self-sovereign identity, April 2016. URL `http://www.lifewithalacrity.com/2016/04/the-path-to-self-soverereign-identity.html`.

[5] Gergely Alpár, Fabian van den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. Irma: practical, decentralized and privacy-friendly identity management using smartphones. In *10th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2017)*, pages 1–2, 2017.

[6] Mansour Alsaleh, Abdulrahman Alarifi, Abdul Malik Al-Salman, Mohammed Alfayez, and Abdulmajeed Almuhaysin. Tsd: Detecting sybil accounts in twitter. In *2014 13th International Conference on Machine Learning and Applications*, pages 463–469. IEEE, 2014.

[7] Elli Androulaki, Seung Geol Choi, Steven M Bellovin, and Tal Malkin. Reputation systems for anonymous networks. In *International Symposium on Privacy Enhancing Technologies*, pages 202–218. Springer, 2008.

[8] Arthanareeswaran Angappan, TP Saravanabava, P Sakthivel, and KS Vishvaksenan. Novel sybil attack detection using rssi and neighbour information to ensure secure communication in wsn. *Journal of Ambient Intelligence and Humanized Computing*, 12(6):6567–6578, 2021.

[9] Alex Auvolat, Davide Frey, Michel Raynal, Francois Taiani, et al. Money transfer made simple: a specification, a generic algorithm, and its proof. *Bulletin of EATCS*, 3(132), 2020.

[10] Inês M Lima Azevedo, M Granger Morgan, and Lester Lave. Residential and regional electricity consumption in the us and eu: How much will higher prices reduce co2 emissions? *The Electricity Journal*, 24(1):21–29, 2011.

[11] DS Baars. Towards self-sovereign identity using blockchain technology. Master's thesis, University of Twente, 2016.

[12] Abigael Okikijesu Bada, Amalia Damianou, Constantinos Marios Angelopoulos, and Vasilios Katos. Towards a green blockchain: A review of consensus mechanisms and their energy consumption. In *2021 17th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, pages 503–511. IEEE, 2021.

[13] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[14] Johannes Bahrke and Charles Manoury. Commission proposes a trusted and secure digital identity for all europeans. European Commission, June 2021. URL `https://ec.europa.eu/commission/presscorner/detail/en/IP_21_2663`.

[15] Chong Bai. State-of-the-art and future trends of blockchain based on dag structure. In *International Workshop on Structured Object-Oriented Formal Language and Method*, pages 183–196. Springer, 2018.

[16] Peter Bailis and Ali Ghodsi. Eventual consistency today: Limitations, extensions, and beyond. *Communications of the ACM*, 56(5):55–63, 2013.

[17] Leemon Baird. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. *Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep*, 34, 2016.

[18] Leemon Baird, Mance Harmon, and Paul Madsen. Hedera: A public hashgraph network & governing council. *White Paper*, 1, 2019.

[19] Paul Baran. On distributed communications networks. rand corporation. *P-2626), Santa Monica, September 1962, 40 pp*, 32:168–267, 1962.

[20] Paulo C Bartolomeu, Emanuel Vieira, Seyed M Hosseini, and Joaquim Ferreira. Self-sovereign identity: Use-cases, technologies, and challenges for industrial iot. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1173–1180. IEEE, 2019.

[21] Rida A Bazzi and Goran Konjevod. On the establishment of distinct identities in overlay networks. *Distributed Computing*, 19(4):267–287, 2007.

[22] Roman Beck, Christoph Müller-Bloch, and John Leslie King. Governance in the blockchain economy: A framework and research agenda. *Journal of the Association for Information Systems*, 19(10):1, 2018.

[23] Nicole Lang Beebe, Sonia D Stacy, and Dane Stuckey. Digital forensic implications of zfs. *digital investigation*, 6:S99–S107, 2009.

[24] Tal Be'ery. Ethology: A safari tour in ethereum's dark forest, 2020. URL `https://zengo.com/ethology-a-safari-tour-in-ethereums-dark-forest/`.

[25] Juan Benet. IPFS-content addressed, versioned, p2p file system (draft 3). *arXiv preprint arXiv:1407.3561*, pages 1–11, 2014.

[26] Timothy J Berners-Lee. Information management: A proposal. Technical report, CERN, 1989.

[27] Patrik Bichsel, Carl Binding, Jan Camenisch, Thomas Groß, Tom Heydt-Benjamin, Dieter Sommer, and Greg Zaverucha. Cryptographic protocols of the identity mixer library. In *Technical Report*, volume 99740, page 3730. 2009.

[28] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[29] Kenneth P Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, and Yaron Minsky. Bimodal multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[30] Alex Biryukov and Daniel Feher. Recon: Sybil-resistant consensus from reputation. *Pervasive and Mobile Computing*, 61:101109, 2020.

[31] Alex Biryukov and Ivan Pustogarov. Bitcoin over tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*, pages 122–134. IEEE, 2015.

[32] Alex Biryukov and Sergei Tikhomirov. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *2019 IEEE European symposium on security and privacy (EuroS&P)*, pages 172–184. IEEE, 2019.

[33] Eric Blais, Joshua Brody, and Badih Ghazi. The information complexity of hamming distance. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2014)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2014.

[34] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 329–349, 2019.

[35] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[36] Yogita Borse, Anushka Chawathe, Deepti Patole, and Purnima Ahirao. Anonymity: A secure identity management using smart contracts. In *Proceedings of International Conference on Sustainable Computing in Science, Technology and Management (SUSCOM), Amity University Rajasthan, Jaipur-India*, 2019.

[37] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53(1-13): 2, 2008.

[38] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: Byzantine resilient random membership sampling. *Computer Networks*, 53(13):2340–2359, 2009.

[39] Eyuphan Bulut and Boleslaw K Szymanski. Exploiting friendship relations for efficient routing in mobile social networks. *IEEE Transactions on Parallel and Distributed Systems*, 23(12):2254–2265, 2012.

[40] Umit Cali, Komal Khan, Shammya Shananda Saha, Tamara Hughes, Farrokh Rahimi, Leonard C Tillman, Islam El-Sayed, Pablo Arboleya, and Sri Nikhil Gupta Gourisetti. Smart contract as an enabler for the digital green transition. In *2022 IEEE PES Transactive Energy Systems Conference (TESC)*, pages 1–5. IEEE, 2022.

[41] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In *International Conference on Security in Communication Networks*, pages 268–289. Springer, 2002.

[42] Kim Cameron and Michael B Jones. Design rationale behind the identity metasystem architecture. In *ISSE/SECURE 2007 Securing Electronic Business Processes*, pages 117–129. Springer, 2007.

[43] Robson A Campêlo, Marco A Casanova, Dorgival O Guedes, and Alberto HF Laender. A brief survey on replica consistency in cloud environments. *Journal of Internet Services and Applications*, 11(1):1–13, 2020.

[44] Ming Cao and Chai Wah Wu. Topology design for fast convergence of network consensus algorithms. In *2007 IEEE International Symposium on Circuits and Systems*, pages 1029–1032. IEEE, 2007.

[45] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 197–210, 2012.

[46] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, and Dan S Wallach. Secure routing for structured peer-to-peer overlay networks. *ACM SIGOPS Operating Systems Review*, 36(SI):299–314, 2002.

[47] Eric Y Chen, Yutong Pei, Shuo Chen, Yuan Tian, Robert Kotcher, and Patrick Tague. Oauth demystified for mobile application developers. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 892–903, 2014.

[48] Usman W Chohan. Are cryptocurrencies truly trustless? In *Cryptofinance and Mechanisms of Exchange*, pages 77–89. Springer, 2019.

[49] Lin William Cong and Zhiguo He. Blockchain disruption and smart contracts. *The Review of Financial Studies*, 32(5):1754–1797, 2019.

[50] Vlad C Coroama and Lorenz M Hilty. Assessing internet energy intensity: A review of methods and results. *Environmental impact assessment review*, 45:63–68, 2014.

[51] Arturo Crespo and Hector Garcia-Molina. Semantic overlay networks for p2p systems. In *International Workshop on Agents and P2P Computing*, pages 1–13. Springer, 2004.

[52] Pierluigi Cuccuru. Beyond bitcoin: an early overview on smart contracts. *International Journal of Law and Information Technology*, 25(3):179–195, 2017.

[53] Weverton Luis da Costa Cordeiro, Flávio Roberto Santos, Gustavo Huff Mauch, Marinho Pilla Barcelos, and Luciano Paschoal Gaspary. Identity management based on adaptive puzzles to protect p2p systems from sybil attacks. *Computer Networks*, 56(11):2569–2589, 2012. ISSN 1389-1286. doi: https://doi.org/10.1016/j.comnet.2012.03.026. URL `https://www.sciencedirect.com/science/article/pii/S1389128612001417`.

[54] Weverton Luis da Costa Cordeiro, Flávio Roberto Santos, Marinho Pilla Barcellos, Luciano Paschoal Gaspary, Hanna Kavalionak, Alessio Guerrieri, and Alberto Montresor. Making puzzles green and useful for adaptive identity management in large-scale distributed systems. *Computer Networks*, 95:97–114, 2016.

[55] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A decentralized network coordinate system. In *ACM SIGCOMM Computer Communication Review*, volume 34(4), pages 15–26. ACM, 2004.

[56] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[57] Anwitaman Datta, Manfred Hauswirth, and Karl Aberer. Beyond ”web of trust”: Enabling p2p e-commerce. In *EEE International Conference on E-Commerce, 2003. CEC 2003.*, pages 303–312. IEEE, 2003.

[58] Sinclair Davidson, Primavera De Filippi, and Jason Potts. Economics of blockchain. *Available at SSRN 2744751*, 2016.

[59] Jai Dayal, Drew Bratcher, Greg Eisenhauer, Karsten Schwan, Matthew Wolf, Xuechen Zhang, Hasan Abbasi, Scott Klasky, and Norbert Podhorszki. Flexpath: Type-based publish/subscribe system for large-scale science analytics. In *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 246–255. IEEE, 2014.

[60] Manlio De Domenico, Antonio Lima, Paul Mougel, and Mirco Musolesi. The anatomy of a scientific rumor. *Nature Scientific reports*, 3:2980, October 2013. doi: 10.1038/srep02980. URL `https://doi.org/10.1038/srep02980`.

[61] Rubén de Juan-Marín, Hendrik Decker, José Enrique Armendáriz-Íñigo, José M Bernabéu-Aubán, and Francesc D Muñoz-Escoí. Scalability approaches for causal multicast: a survey. *Computing*, 98(9):923–947, 2016.

[62] Rodrigo Couto de Souza, Edimara Mezzomo Luciano, and Guilherme Costa Wiedenhöft. The uses of the blockchain smart contracts to reduce the levels of corruption: Some preliminary thoughts. In *Proceedings of the 19th Annual International Conference on Digital Government Research: Governance in the Data Age*, pages 1–2, 2018.

[63] M.A. de Vos. *Decentralization and Disintermediation in Blockchain-based Marketplaces.* PhD thesis, Delft University of Technology, 2021.

[64] Murat Demirbas and Youngwhan Song. An rssi-based scheme for sybil attack detection in wireless sensor networks. In *2006 International symposium on a world of wireless, mobile and multimedia networks (WoWMoM'06)*, pages 5–pp. ieee, 2006.

[65] Varun Deshpande, Hakim Badis, and Laurent George. Btcmap: mapping bitcoin peer-to-peer network topology. In *2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6. IEEE, 2018.

[66] Yvo Desmedt. Abuses in cryptography and how to fight them. In *Conference on the Theory and Application of Cryptography*, pages 375–389. Springer, 1988.

[67] Rachna Dhamija and Lisa Dusseault. The Seven Flaws of Identity Management: Usability and Security Challenges. *IEEE Security & Privacy Magazine*, 6(2):24–29, March 2008. ISSN 1540-7993. doi: 10.1109/MSP.2008.49.

[68] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[69] Darcy DiNucci. Design & new media: Fragmented future-web development faces a process of mitosis, mutation, and natural selection. *PRINT-NEW YORK-*, 53:32–35, 1999.

[70] John R Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[71] Randall E Duran and Paul Griffin. Smart contracts: will fintech be the catalyst for the next global financial crisis? *Journal of Financial Regulation and Compliance*, 2019.

[72] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, San Francisco, CA, USA, May 2012. IEEE. ISBN 978-1-4673-1244-8 978-0-7695-4681-0. doi: 10.1109/SP.2012.28.

[73] Helen Eenmaa-Dimitrieva and Maria José Schmidt-Kessen. Creating markets in no-trust environments: The law and economics of smart contracts. *Computer law & security review*, 35(1):69–88, 2019.

[74] O. Ersoy. *Incentives and Cryptographic Protocols for Bitcoin-like Blockchains.* PhD thesis, Delft University of Technology, 2021.

[75] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 170–189. Springer, 2019.

[76] Obinna Ethelbert, Faraz Fatemi Moghaddam, Philipp Wieder, and Ramin Yahyapour. A json token-based authentication and access management schema for cloud saas applications. In *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*, pages 47–53. IEEE, 2017.

[77] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[78] E Ezhilarasan and M Dinakaran. A review on mobile technologies: 3g, 4g and 5g. In *2017 second international conference on recent trends and challenges in computational models (ICRTCCM)*, pages 369–373. IEEE, 2017.

[79] José G Faísca and José Q Rogado. Decentralized semantic identity. In *Proceedings of the 12th International Conference on Semantic Systems*, pages 177–180, 2016.

[80] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. *Measurement and Analysis of Computing Systems*, 2(2):1–35, June 2018. ISSN 2476-1249. doi: 10.1145/3224424.

[81] Michal Feldman and John Chuang. Overcoming free-riding behavior in peer-to-peer systems. *ACM sigecom exchanges*, 5(4):41–50, 2005.

[82] Bryan Ford, Pyda Srisuresh, and Dan Kegel. Peer-to-peer communication across network address translators. In *USENIX Annual Technical Conference, General Track*, pages 179–192, 2005.

[83] The SelfKey Foundation. Selfkey, September 2017. URL `https://selfkey.org/wp-content/uploads/2019/03/selfkey-whitepaper-en.pdf`.

[84] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866*, 2018.

[85] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. The limitations of federated learning in sybil settings. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 301–316, 2020.

[86] P. Gao, B. Wang, N. Z. Gong, S. R. Kulkarni, K. Thomas, and P. Mittal. Sybilfuse: Combining local attributes with global structure to perform robust sybil detection. In *2018 IEEE Conference on Communications and Network Security (CNS)*, pages 1–9, May 2018. doi: 10.1109/CNS.2018.8433147.

[87] Gartner. Digitalization will make most heritage financial firms irrelevant, 10 2018. `https://www.gartner.com/en/doc/338356-digitalization-will-make-most-heritage-financial-firms-irrelevant`.

[88] Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 326–335. ACM, 2014.

[89] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68. ACM, 2017.

[90] Paul A. Grassi, Michael E. Garcia, and James L. Fenton. Nist special publication 800-63-3: Digital identity guidelines. 2017. doi: 10.6028/NIST.SP.800-63-3.

[91] Jesper Grolin. Corporate legitimacy in risk society: The case of brent spar. *Business Strategy and the Environment*, 7(4):213–222, 1998.

[92] Andreas Grüner, Alexander Mühle, and Christoph Meinel. An integration architecture to enable service providers for self-sovereign identity. In *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pages 1–5. IEEE, 2019.

[93] Roch Guérin and Vinod Peris. Quality-of-service in packet networks: basic mechanisms and directions. *Computer networks*, 31(3):169–189, 1999.

[94] Gilles Guette and Bertrand Ducourthial. On the sybil attack detection in vanet. In *2007 IEEE international conference on Mobile Adhoc and sensor systems*, pages 1–6. IEEE, 2007.

[95] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. Peerreview: Practical accountability for distributed systems. *ACM SIGOPS operating systems review*, 41(6): 175–188, 2007.

[96] Abdelatif Hafid, Abdelhakim Senhaji Hafid, and Mustapha Samih. Scaling blockchains: A comprehensive survey. *IEEE Access*, 8:125244–125262, 2020.

[97] Jan Hajny and Lukas Malina. Unlinkable attribute-based credentials with practical revocation on smart-cards. In *International Conference on Smart Card Research and Advanced Applications*, pages 62–76. Springer, 2012.

[98] Omar Hasan, Lionel Brunie, Elisa Bertino, and Ning Shang. A decentralized privacy preserving reputation protocol for the malicious adversarial model. *IEEE Transactions on Information Forensics and Security*, 8(6):949–962, 2013.

[99] Samer Hassan and Primavera De Filippi. The expansion of algorithmic governance: from code is law to law is code. *Field Actions Science Reports. The journal of field actions*, (Special Issue 17):88–90, 2017.

[100] Hideaki Hata, Mingyu Guo, and M Ali Babar. Understanding the heterogeneity of contributors in bug bounty programs. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 223–228. IEEE, 2017.

[101] Mike Hearn and Richard Gendal Brown. Corda: A distributed ledger. *Corda Technical White Paper*, 2016, 2016.

[102] Everett Hildenbrandt, Manasvi Saxena, Nishant Rodrigues, Xiaoran Zhu, Philip Daian, Dwight Guth, Brandon Moore, Daejun Park, Yi Zhang, Andrei Stefanescu, et al. Kevm: A complete formal semantics of the ethereum virtual machine. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*, pages 204–217. IEEE, 2018.

[103] Vincent C Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. Guide to attribute based access control (abac) definition and considerations. 2014. doi: 10.6028/NIST.SP.800-162.

[104] Yong Huang, Wei Wang, Yiyuan Wang, Tao Jiang, and Qian Zhang. Lightweight sybil-resilient multi-robot networks by multipath manipulation. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications*, pages 2185–2193. IEEE, 2020.

[105] Fabien Imbault, Marie Swiatek, Rodolphe de Beaufort, and Robert Plana. The green blockchain: Managing decentralized energy production and consumption. In *2017 IEEE International Conference on Environment and Electrical Engineering and 2017 IEEE Industrial and Commercial Power Systems Europe (EEEIC/I&CPS Europe)*, pages 1–5. IEEE, 2017.

[106] Leander Jehl and Hein Meling. Towards byzantine fault tolerant publish/subscribe: A state machine approach. In *Proceedings of the 9th Workshop on Hot Topics in Dependable Systems*, pages 1–5, 2013.

[107] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 79–98. Springer, 2004.

[108] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, and Maarten Van Steen. Gossip-based peer sampling. *ACM Transactions on Computer Systems (TOCS)*, 25(3):8–es, 2007.

[109] Jing Jiang, Zifei Shan, Wenpeng Sha, Xiao Wang, and Yafei Dai. Detecting and validating sybil groups in the wild. In *2012 32nd International Conference on Distributed Computing Systems Workshops*, pages 127–132. IEEE, 2012.

[110] Xin-Jian Jiang and Xiao Fan Liu. Cryptokitties transaction network analysis: The rise and fall of the first blockchain game mania. *Frontiers in Physics*, page 57, 2021.

[111] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th international conference on World Wide Web*, pages 640–651, 2003.

[112] Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All about eve: Execute-verify replication for multi-core servers. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 237–250, 2012.

[113] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917, 2012.

[114] Liran Katzir, Edo Liberty, and Oren Somekh. Estimating sizes of social networks via biased sampling. In *Proceedings of the 20th international conference on World wide web*, pages 597–606. ACM, 2011.

[115] Anne-Marie Kermarrec, Alessio Pace, Vivien Quema, and Valerio Schiavoni. Nat-resilient gossip peer sampling. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 360–367. IEEE, 2009.

[116] Dmitry Khovratovich and Jason Law. Sovrin: digital identities in the blockchain era. 2017.

[117] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. Measuring ethereum network peers. In *Proceedings of the Internet Measurement Conference 2018*, pages 91–104, 2018.

[118] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. Local-first software: you own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pages 154–178, 2019.

[119] Tadayoshi Kohno, Andre Broido, and Kimberly C Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, 2005.

[120] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*, pages 3478–3487. PMLR, 2019.

[121] Dániel Kondor, Márton Pósfai, István Csabai, and Gábor Vattay. Do the rich get richer? an empirical analysis of the bitcoin transaction network. *PloS one*, 9(2), 2014.

[122] Galia Kondova and Jörn Erbguth. Self-sovereign identity on public blockchains and the gdpr. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 342–345, 2020.

[123] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al. Handling imbalanced datasets: A review. *GESTS international transactions on computer science and engineering*, 30(1):25–36, 2006.

[124] Eleni Koutrouli and Aphrodite Tsalgatidou. Taxonomy of attacks and defense mechanisms in p2p reputation systems - lessons for reputation system designers. *Comp. Sci. Review*, 6(2-3):47–70, 2012.

[125] Mirja Kühlewind and Bob Briscoe. Chirping for congestion control-implementation feasibility. *Proceedings of PFLDNeT'10*, 2010.

[126] Bogdan Kulynych, Wouter Lueks, Marios Isaakidis, George Danezis, and Carmela Troncoso. Claimchain: Improving the security and privacy of in-band key distribution for messaging. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, pages 86–103, 2018.

[127] Jitendra Kurmi and Ankur Sodhi. A survey of zero-knowledge proof for authentication. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(1), 2015.

[128] Ruggero Donida Labati, Angelo Genovese, Enrique Muñoz, Vincenzo Piuri, Fabio Scotti, and Gianluca Sforza. Biometric recognition in automated border control: a survey. *ACM Computing Surveys (CSUR)*, 49(2):1–39, 2016.

[129] Ashwin Lall. Data streaming algorithms for the kolmogorov-smirnov test. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 95–104. IEEE, 2015.

[130] Raúl Landa, Joao Taveira Araújo, Richard G Clegg, Eleni Mykoniati, David Griffin, and Miguel Rio. The large-scale geography of internet round trip times. In *2013 IFIP Networking Conference*, pages 1–9. IEEE, 2013.

[131] Jonathan Ledlie, Paul Gardner, and Margo I Seltzer. Network coordinates in the wild. In *NSDI*, volume 7, pages 299–311, 2007.

[132] Jei Young Lee. A decentralized token economy: How blockchain and cryptocurrency can revolutionize business. *Business Horizons*, 62(6):773–784, 2019.

[133] Jintae Lee. An end-user perspective on file-sharing systems. *Communications of the ACM*, 46(2):49–53, 2003.

[134] Dave Levin, John R Douceur, Jacob R Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *NSDI*, volume 9, pages 1–14, 2009.

[135] Brian Neil Levine, Clay Shields, and N Boris Margolin. A survey of solutions to the sybil attack. *University of Massachusetts Amherst, Amherst, MA*, 7:224, 2006.

[136] Chenxin Li, Peilun Li, Dong Zhou, Zhe Yang, Ming Wu, Guang Yang, Wei Xu, Fan Long, and Andrew Chi-Chih Yao. A decentralized blockchain with high throughput and fast confirmation. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 515–528, 2020.

[137] Frank Li, Prateek Mittal, Matthew Caesar, and Nikita Borisov. Sybilcontrol: Practical sybil defense with computational puzzles. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pages 67–78, 2012.

[138] Harry C Li, Allen Clement, Edmund L Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 191–204, 2006.

[139] Shancang Li, Shanshan Zhao, Po Yang, Panagiotis Andriotis, Lida Xu, and Qindong Sun. Distributed consensus algorithm for events detection in cyber-physical systems. *IEEE Internet of Things Journal*, 6(2):2299–2308, 2019.

[140] Wenjie Li, Sharief MA Oteafy, and Hossam S Hassanein. Streamcache: Popularity-based caching for adaptive streaming over information-centric networks. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[141] Jian Liang, Rakesh Kumar, and Keith W Ross. The kazaa overlay: A measurement study. *Computer Networks Journal (Elsevier)*, 49(6), 2005.

[142] Jian Liang, Naoum Naoumov, and Keith W Ross. The index poisoning attack in p2p file sharing systems. In *INFOCOM*, 2006.

[143] Rolf Lindemann. The evolution of authentication. In *ISSE 2013 Securing Electronic Business Processes*, pages 11–19. Springer, 2013.

[144] Juan Liu, Jun Lv, Hasan Dinçer, Serhat Yüksel, and Hüsne Karakuş. Selection of renewable energy alternatives for green blockchain investments: A hybrid it2-based fuzzy modelling. *Archives of Computational Methods in Engineering*, pages 1–15, 2021.

[145] Yue Liu, David R Bild, Robert P Dick, Z Morley Mao, and Dan S Wallach. The mason test: A defense against sybil attacks in wireless networks without trusted authorities. *IEEE Transactions on Mobile Computing*, 14(11):2376–2391, 2015.

[146] Thomas Locher, Stefan Schmid, and Roger Wattenhofer. equus: A provably robust and locality-aware peer-to-peer system. In *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)*, pages 3–11. IEEE, 2006.

[147] Thomas Locher, David Mysicka, Stefan Schmid, and Roger Wattenhofer. Poisoning the kad network. In *International Conference on Distributed Computing and Networking*, pages 195–206. Springer, 2010.

[148] Scott Locklin. Token economics, 2018. URL `https://basicattentiontoken.org/static-assets/documents/token-econ-2018.pdf`.

[149] Matt Lockyer, N Mudge, and J Schalm. Erc-998 composable non-fungible token standard. *Ethereum Foundation (Stiftung Ethereum), Zug, Switzerland*, 2015.

[150] Alexander Löser, Felix Naumann, Wolf Siberski, Wolfgang Nejdl, and Uwe Thaden. Semantic overlay clusters within super-peer networks. In *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pages 33–47. Springer, 2003.

[151] Georgios Loukas and Gülay Öke. Protection against denial of service attacks: A survey. *The Computer Journal*, 53(7):1020–1037, 2010.

[152] Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers. Fast revocation of attribute-based credentials for both users and verifiers. *Computers & Security*, 67:308–323, 2017.

[153] Cristian Lumezanu, Randy Baden, Neil Spring, and Bobby Bhattacharjee. Triangle inequality variations in the internet. In *ACM SIGCOMM conference on Internet measurement*. ACM, 2009.

[154] Daniele Magazzeni, Peter McBurney, and William Nash. Validation and verification of smart contracts: A research agenda. *Computer*, 50(9):50–57, 2017.

[155] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1366. IEEE, 2021.

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

[157] Sigurd Meldal, Sriram Sankar, and James Vera. Exploiting locality in maintaining potential causality. In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 231–239, 1991.

[158] Michel Meulpolder, Johan A Pouwelse, Dick HJ Epema, and Henk J Sips. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8. IEEE, 2009.

[159] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[160] CNN Money. Napster: 20 million users. URL `https://money.cnn.com/2000/07/19/technology/napster/index.htm`.

[161] Seyed Hossein Mortazavi, Bharath Balasubramanian, Eyal de Lara, and Shankaranarayanan Puzhavakath Narayanan. Toward session consistency for the edge. In *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[162] Alexander Mühle, Andreas Grüner, Tatiana Gayvoronskaya, and Christoph Meinel. A survey on essential components of a self-sovereign identity. *Computer Science Review*, 30:80–86, 2018.

[163] Matthieu Nadini, Laura Alessandretti, Flavio Di Giacinto, Mauro Martino, Luca Maria Aiello, and Andrea Baronchelli. Mapping the nft revolution: market trends, trade networks, and visual features. *Scientific reports*, 11(1):1–11, 2021.

[164] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, 2008.

[165] Pezhman Nasirifard, Ruben Mayer, and Hans-Arno Jacobsen. Fabriccrdt: A conflict-free replicated datatypes approach to permissioned blockchains. In *Proceedings of the 20th International Middleware Conference*, pages 110–122, 2019.

[166] Brice Nédelec, Pascal Molli, Achour Mostefaoui, and Emmanuel Desmontils. Lseq: an adaptive structure for sequences in distributed collaborative editing. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 37–46, 2013.

[167] Sina Rafati Niya, Benjamin Jeffrey, and Burkhard Stiller. Kyot: Self-sovereign iot identification with a physically unclonable function. In *2020 IEEE 45th Conference on Local Computer Networks (LCN)*, pages 485–490. IEEE, 2020.

[168] Kieron O'hara. Smart contracts-dumb idea. *IEEE Internet Computing*, 21(2):97–101, 2017.

[169] Vinicius C Oliveira, Julia Almeida Valadares, Jose Eduardo A. Sousa, Alex Borges Vieira, Heder Soares Bernardino, Saulo Moraes Villela, and Glauber Dias Goncalves. Analyzing transaction confirmation in ethereum using machine learning techniques. *ACM SIGMETRICS Performance Evaluation Review*, 48(4):12–15, 2021.

[170] Cathy O'Neil and Rachel Schutt. *Doing data science: Straight talk from the frontline.* " O'Reilly Media, Inc.", 2013.

[171] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference (Usenix ATC 14)*, pages 305–319, 2014.

[172] Nouha Oualha and Yves Roudier. Reputation and audits for self-organizing storage. In *Proceedings of the workshop on Security in Opportunistic and SOCial networks*, pages 1–10, 2008.

[173] Lasse Øverlier and Paul Syverson. Improving efficiency and simplicity of tor circuit establishment and hidden services. In *International Workshop on Privacy Enhancing Technologies*, pages 134–152. Springer, 2007.

[174] Sehyun Park, Seongwon Im, Youhwan Seol, and Jeongyeup Paek. Nodes in the bitcoin network: Comparative measurement study and survey. *IEEE Access*, 7:57009–57022, 2019.

[175] Christos Patsonakis, Katerina Samari, Aggelos Kiayiasy, and Mema Roussopoulos. On the practicality of a smart contract pki. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 109–118. IEEE, 2019.

[176] Dave Peck and the PSL Team. An engineer's hype-free observations on web3 (and its possibilities), 2021. URL `https://www.psl.com/feed-posts/web3-engineer-take`.

[177] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[178] Kun Peng and Feng Bao. An efficient range proof scheme. In *2010 IEEE Second International Conference on Social Computing*, pages 826–833. IEEE, 2010.

[179] Liam Peyton, Chintan Doshi, and Pierre Seguin. An audit trail service to enhance privacy compliance in federated identity management. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 175–187, 2007.

[180] Andrea Pinna, Simona Ibba, Gavina Baralla, Roberto Tonelli, and Michele Marchesi. A massive analysis of ethereum smart contracts empirical study and code metrics. *IEEE Access*, 7:78194–78213, 2019.

[181] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016. URL `https://lightning.network/lightning-network-paper.pdf`.

[182] Serguei Popov. The tangle. *White paper*, 1(3), 2018.

[183] Johan A Pouwelse, Pawel Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick HJ Epema, Marcel Reinders, Maarten R Van Steen, and Henk J Sips. Tribler: a social-based peer-to-peer system. *Concurrency and computation: Practice and experience*, 20(2):127–138, 2008.

[184] Thomas Puschmann. Fintech. *Business & Information Systems Engineering*, 59(1): 69–76, 2017.

[185] Krishna Ramachandran, Irfan Sheriff, Elizabeth Belding, and Kevin Almeroth. Routing stability in static wireless mesh networks. In *International Conference on Passive and Active Network Measurement*, pages 73–82. Springer, 2007.

[186] Muhammad Raza, Venkatesh Samineni, and William Robertson. Physical and logical topology slicing through sdn. In *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–4. IEEE, 2016.

[187] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, pages 11–16, 2006.

[188] Drummond Reed, Manu Sporny, Dave Longley, Christopher Allen, Ryan Grant, and Markus Sabadello. Decentralized identifiers (dids) v1.0, April 2020. URL `https://www.w3.org/TR/did-core/`.

[189] Dan Robinson and Georgios Konstantopoulos. Ethereum is a dark forest, 2020. URL `https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/`.

[190] Team Rocket. Snowflake to avalanche : A novel metastable consensus protocol family for cryptocurrencies. 2018.

[191] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

[192] Hosam Rowaihy, William Enck, Patrick McDaniel, and Thomas La Porta. Limiting sybil attacks in structured p2p networks. In *IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications*, pages 2596–2600. IEEE, 2007.

[193] Muhammad Saad, Victor Cook, Lan Nguyen, My T Thai, and Aziz Mohaisen. Partitioning attacks on bitcoin: Colliding space, time, and logic. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1175–1187. IEEE, 2019.

[194] Muhammad Saad, Songqing Chen, and David Mohaisen. Root cause analyses for the deteriorating bitcoin network synchronization. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 239–249. IEEE, 2021.

[195] Mohammad Sadoghi, Martin Jergler, Hans-Arno Jacobsen, Richard Hull, and Roman Vaculín. Safe distribution and parallel execution of data-centric workflows over the publish/subscribe abstraction. *IEEE Transactions on Knowledge and Data Engineering*, 27(10):2824–2838, 2015.

[196] Ermin Sakic and Wolfgang Kellerer. Decoupling of distributed consensus, failure detection and agreement in sdn control plane. In *IFIP Networking 2020*, page 9, 2020.

[197] Chinmay Saraf and Siddharth Sabadra. Blockchain platforms: A compendium. In *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*, pages 1–6. IEEE, 2018.

[198] Alexander Schaub, Rémi Bazin, Omar Hasan, and Lionel Brunie. A trustless privacy-preserving reputation system. In *IFIP International Conference on ICT Systems Security and Privacy Protection*, pages 398–411. Springer, 2016.

[199] Johannes Sedlmeir, Hans Ulrich Buhl, Gilbert Fridgen, and Robert Keller. The energy consumption of blockchain technology: beyond myth. *Business & Information Systems Engineering*, 62(6):599–608, 2020.

[200] Tallat M Shafaat, Ali Ghodsi, and Seif Haridi. A practical approach to network size estimation for structured overlays. In *International Workshop on Self-Organizing Systems*, pages 71–83. Springer, 2008.

[201] Savva Shanaev, Arina Shuraeva, Mikhail Vasenin, and Maksim Kuznetsov. Cryptocurrency value and 51% attacks: evidence from event studies. *The Journal of Alternative Investments*, 22(3):65–77, 2019.

[202] Marc Shapiro, Nuno Preguiça, Carlos Baquero, and Marek Zawirski. Conflict-free replicated data types. In *Symposium on Self-Stabilizing Systems*, pages 386–400. Springer, 2011.

[203] Voshmgir Shermin. Disrupting governance with blockchains and smart contracts. *Strategic Change*, 26(5):499–509, 2017.

[204] Micah Sherr, Matt Blaze, and Boon Thau Loo. Veracity: Practical secure network coordinates via vote-based agreements. In *USENIX Annual Technical Conference*, 2009.

[205] Atul Singh et al. Eclipse attacks on overlay networks: Threats and defenses. In *In IEEE INFOCOM*, 2006.

[206] James E Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38 (5):32–38, 2005.

[207] Manu Sporny, Dave Longley, Markus Sabadello, Drummond Reed, Orie Steele, and Christopher Allen. Decentralized identifiers (dids) v1.0: Core architecture, data model, and representations. Technical report, W3C, 2022.

[208] Mudhakar Srivatsa and Mike Hicks. Deanonymizing mobility traces: Using social network as a side-channel. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 628–637, 2012.

[209] Alexander Stannat, Can Umut Ileri, Dion Gijswijt, and Johan Pouwelse. Achieving sybil-proofness in distributed work systems. In *International Conference on Autonomous Agents and Multiagent Systems*, 2021.

[210] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. Exploiting kad: possible uses and misuses. *ACM SIGCOMM Computer Communication Review*, 37(5):65–70, 2007.

[211] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. A global view of kad. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 117–122, 2007.

[212] Quinten Stokkink and Johan Pouwelse. Deployment of a blockchain-based self-sovereign identity. In *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, pages 1336–1342. IEEE, 2018.

[213] Quinten Stokkink, Can Umut Ileri, Johan Pouwelse, and Jan S. Rellermeyer. Latency collision measurements. 4TU.Centre for Research Data. Dataset. https://doi.org/10.4121/uuid:34850d65-1908-4249-b446-8e87c6d21ba0, 2020.

[214] Quinten Stokkink, Alexander Stannat, and Johan Pouwelse. Foundations of peer-to-peer reputation. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good*, pages 25–30, 2020.

[215] Quinten Stokkink, Georgy Ishmaev, Dick Epema, and Johan Pouwelse. A truly self-sovereign identity system. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 1–8. IEEE, 2021.

[216] Quinten Stokkink, Can Umut Ileri, and Johan Pouwelse. Reputation-based data carrying for web3 networks. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 283–286. IEEE, 2022.

[217] Miha Stopar, Manca Bizjak, Jolanda Modic, Jan Hartman, Anže Žitnik, and Tilen Marc. emmy–trust-enhancing authentication library. In *IFIP International Conference on Trust Management*, pages 133–146. Springer, 2019.

[218] Emilio Calvanese Strinati and Sergio Barbarossa. 6g networks: Beyond shannon towards semantic and goal-oriented communications. *Computer Networks*, 190:107930, 2021.

[219] Kalika Suksomboon, Saran Tarnoi, Yusheng Ji, Michihiro Koibuchi, Kensuke Fukuda, Shunji Abe, Nakamura Motonori, Michihiro Aoki, Shigeo Urushidani, and Shigeki Yamada. Popcache: Cache more or less based on content popularity for information-centric networking. In *38th Annual IEEE Conference on Local Computer Networks*, pages 236–243. IEEE, 2013.

[220] Karl Taeuscher. Uncertainty kills the long tail: Demand concentration in peer-to-peer marketplaces. *Electronic Markets*, 29(4):649–660, 2019.

[221] Yuechen Tao, Bo Li, Jingjie Jiang, Hok Chu Ng, Cong Wang, and Baochun Li. On sharding open blockchains with smart contracts. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1357–1368. IEEE, 2020.

[222] DeGate Team. An analysis of ethereum front-running and its defense solutions, 2021. URL `https://medium.com/degate/an-analysis-of-ethereum-front-running-and-its-defense-solutions-34ef81ba8456`.

[223] Anjan V Thakor. Fintech and banking: What do we know? *Journal of Financial Intermediation*, 41:100833, 2020.

[224] Elise Thomas, Albert Zhang, et al. *ID2020, Bill Gates and the Mark of the Beast: how Covid-19 catalyses existing online conspiracy movements.* JSTOR, 2020.

[225] Gunnar Thorvaldsen. *Censuses and census takers: A global history.* Routledge, 2017.

[226] Muoi Tran, Akshaye Shenoi, and Min Suk Kang. On the routing-aware peering against network-eclipse attacks in bitcoin. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[227] Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, and Sherman SM Chow. Optimal sybil-resilient node admission control. In *2011 Proceedings IEEE INFOCOM*, pages 3218–3226. IEEE, 2011.

[228] Petar Tsankov, Andrei Dan, Dana Drachsler-Cohen, Arthur Gervais, Florian Buenzli, and Martin Vechev. Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 67–82, 2018.

[229] Lewis Tseng. Eventual consensus: Applications to storage and blockchain. In *2019 57th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 840–846. IEEE, 2019.

[230] F Vogelsteller and V Buterin. Erc-20 token standard. ethereum foundation (stiftung ethereum), zug, switzerland (2015).

[231] Shermin Voshmgir. *Token Economy: How the Web3 reinvents the Internet*, volume 2. Token Kitchen, 2020.

[232] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks. *arXiv preprint arXiv:2007.02754*, 2020.

[233] Chundong Wang, Likun Zhu, Liangyi Gong, Zhentang Zhao, Lei Yang, Zheli Liu, and Xiaochun Cheng. Accurate sybil attack detection based on fine-grained physical channel information. *Sensors*, 18(3):878, 2018.

[234] Fennie Wang and Primavera De Filippi. Self-sovereign identity in a globalized world: Credentials-based identity systems as a driver for economic inclusion. *Frontiers in Blockchain*, 2:28, 2020.

[235] Honghao Wang, Yingwu Zhu, and Yiming Hu. An efficient and secure peer-to-peer overlay network. In *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05) l*, pages 8–pp. IEEE, 2005.

[236] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320, 2006.

[237] Sage A Weil, Scott A Brandt, Ethan L Miller, and Carlos Maltzahn. Crush: Controlled, scalable, decentralized placement of replicated data. In *SC'06: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, pages 31–31. IEEE, 2006.

[238] Moritz Wendl, My Hanh Doan, and Remmer Sassen. The environmental impact of cryptocurrencies using proof of work and proof of stake consensus algorithms: A systematic review. *Journal of Environmental Management*, 326:116530, 2023.

[239] Phillip Windley. How sovrin works. *Sovrin Foundation*, pages 1–10, 2016.

[240] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[241] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[242] Atsushi Yamamoto, Daisuke Asahara, Tomoko Itao, Satoshi Tanaka, and Tatsuya Suda. Distributed pagerank: a distributed reputation model for open peer-to-peer network. In *2004 International Symposium on Applications and the Internet Workshops. 2004 Workshops.*, pages 389–394. IEEE, 2004.

[243] Haifeng Yu, Michael Kaminsky, Phillip Gibbons, and Abraham Flaxman. Sybilguard: defending against sybil attacks via social networks. *ACM SIGCOMM Computer Communication Review*, 36(4):267–278, 2006.

[244] Haifeng Yu, Phillip Gibbons, Michael Kaminsky, and Feng Xiao. Sybillimit: A near-optimal social network defense against sybil attacks. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 3–17. IEEE, 2008.

[245] Adja Elloh Yves-Christian, Badis Hammi, Ahmed Serhrouchni, and Houda Labiod. Total eclipse: How to completely isolate a bitcoin peer. In *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, pages 1–7. IEEE, 2018.

[246] Sebastian Zander and Steven J Murdoch. An improved clock-skew measurement technique for revealing hidden services. In *USENIX Security Symposium*, pages 211–226, 2008.

[247] Niels Zeilemaker, Boudewijn Schoon, and Johan Pouwelse. Dispersy bundle synchronization. *TU Delft, Parallel and Distributed Systems*, 2013.

[248] Qingji Zheng and Shouhuai Xu. Secure and efficient proof of storage with deduplication. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 1–12. ACM, 2012.

[249] Philip R Zimmermann. *The official PGP user's guide*, volume 5. MIT press Cambridge, 1995.

[250] Aviv Zohar. Bitcoin: under the hood. *Communications of the ACM*, 58(9):104–113, 2015.

# Curriculum Vitæ

## Quinten André Stokkink

| | |
|---|---|
| 1991/03/29 | Date of birth in Leiderdorp, The Netherlands |

## Education

The following is a list of only higher eduction followed leading up to this thesis, omitting individual courses and summerschools.

| | |
|---|---|
| 2017-present | Ph.D. Computer Science<br>Dissertation title: *Systems for Digital Self-Sovereignty*<br>Distributed Systems<br>Delft University of Technology, Delft, The Netherlands |
| 2013-2017 | M.Sc. Computer Science<br>Thesis title: *Multi-core architecture for anonymous Internet streaming*<br>Parallel and Distributed Systems<br>Delft University of Technology, Delft, The Netherlands |
| 2011-2012 | Minor Strategic Management in a High Tech Environment<br>Delft University of Technology, Delft, The Netherlands |
| 2009-2013 | B.Sc. Computer Science<br>Thesis title: *Closing the gap between the Web and Peer to Peer*<br>Delft University of Technology, Delft, The Netherlands |

## Work Experience

The following is a list of all paid work during the making of this thesis.

| | |
|---|---|
| 2023-current | Researcher<br>Distributed Systems<br>Delft University of Technology, Delft, The Netherlands |

| | |
|---|---|
| 2022-2023 | Technical consultant; synthesis of visual data for audio using Artificial Intelligence<br>Emfa Music, Queensland, Australia |
| 2022 | Expert consultant; technical guidelines on security measures for providers of wallets with Blockchain<br>European Union Agency for Cybersecurity (ENISA), Athens, Greece |
| 2017-2022 | Ph.D. Candidate<br>Distributed Systems<br>Delft University of Technology, Delft, The Netherlands |

## Open Source Software Contributions

The following is a list of major open source initiatives that were contributed to during the making of this thesis, omitting smaller projects.

| **Project** | **Language** | **URL** |
|---|---|---|
| `cryptography` | Python | `https://pypi.org/project/cryptography/` |
| `py-ipv8` | Python | `https://pypi.org/project/pyipv8/` |
| `Tribler` | Python | `https://www.tribler.org/` |
| `apng` | R | `https://CRAN.R-project.org/package=apng` |

# List of Publications

📄 1. **Quinten Stokkink**, and Johan Pouwelse. A Local-First Approach for Green Smart Contracts *Distributed Ledger Technologies: Research and Practice*, ISSN 2769-6472. ACM, 2023.

📄 2. **Quinten Stokkink**, Can Umut Ileri, Dick Epema, and Johan Pouwelse. Web3 sybil avoidance using network latency. *Computer Networks*, 227:109701, 2023.

   ★ Won best poster award at Delf University of Technology EEMCS PhD Event 2019.

📄 3. **Quinten Stokkink**, Can Umut Ileri, and Johan Pouwelse. Reputation-Based Data Carrying for Web3 Networks. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*, pages 283–286. IEEE, 2022. Acceptance Rate 26.2% (26/99).

🏆📄 4. **Quinten Stokkink**, Georgy Ishmaev, Dick Epema, and Johan Pouwelse. A truly self-sovereign identity system. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 1–8. IEEE, 2021. Acceptance Rate 27.34% (35/128).

5. **Quinten Stokkink**, Alexander Stannat, and Johan Pouwelse. Foundations of peer-to-peer reputation. In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good*, pages 25–30, 2020.

6. Georgy Ishmaev and **Quinten Stokkink**. Identity management systems: Singular identities and multiple moral issues. *Frontiers in Blockchain*, 3:15, 2020.

7. **Quinten Stokkink** and Johan Pouwelse. Deployment of a blockchain-based self-sovereign identity. In *2018 IEEE international conference on Internet of Things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData)*, pages 1336–1342. IEEE, 2018. Acceptance Rate 15.3% (26/170).

   ☆ Nominated for best poster award at ICT.OPEN 2018.

📄 Included in this thesis.
🏆 Won a best paper award.