Decentralization and Disintermediation in Blockchain-based Marketplaces

de Vos, M.A.

DOI
10.4233/uuid:a4f750b6-5ac5-4709-80c5-71eb71ac7b35

Publication date
2021

Document Version
Final published version

Citation (APA)
de Vos, M. A. (2021). *Decentralization and Disintermediation in Blockchain-based Marketplaces*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:a4f750b6-5ac5-4709-80c5-71eb71ac7b35

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Decentralization and Disintermediation in Blockchain-based Marketplaces

Decentralization and Disintermediation in Blockchain-based Marketplaces

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 woensdag 16 juni 2021 om 17:30 uur

door

Marinus Abraham DE VOS

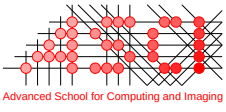
Master of Science in Computer Science,
Technische Universiteit Delft, Nederland,
geboren te Sint-Maartensdijk, Nederland.

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. A. van Deursen,	Technische Universiteit Delft
Prof.dr. A.-M. Kermarrec,	École polytechnique fédérale de Lausanne, Switzerland
Prof.dr. F. Taïani,	Université de Rennes 1, France
Dr. A. Gervais,	Imperial College London, United Kingdom
Dr. D.G.J. Bongaerts,	Erasmus University Rotterdam
Prof.dr. K.G. Langendoen,	Technische Universiteit Delft, reservelid



This work was carried out in the ASCI graduate school.
ASCI dissertation series number 420.

Keywords: decentralization, disintermediation, electronic markets, e-commerce, blockchain, decentralized exchanges, matchmaking, settlement, fraud, information management, identity management, decentralized finance, trading, money

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

Cover by: Martijn de Vos and Jeannet Stoutjesdijk. The cover shows an artistic impression of the TrustChain ledger.

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 978-94-6384-225-9

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

Contents

Summary

Marketplaces facilitate the exchange of services, goods, and information between individuals and businesses. They play an essential role in our economy. The standard approach to devise digital marketplaces is by deploying centralized infrastructure, entirely operated and managed by a market operator. In such centralized marketplaces, trusted intermediaries often provide various services to traders, such as managing market information, processing payments, and providing arbitration services when a dispute arises.

Advancements in information technology have challenged the need for both authoritative market operators and trusted intermediaries. In particular, blockchain technology is increasingly being applied to deploy digital marketplaces. Blockchain-based marketplaces facilitate trade directly between peers while reducing the dependency on both authoritative parties and trusted intermediaries. The role of blockchain in such marketplaces is to replace social trust with cryptographic primitives. This enables the *decentralization* and *disintermediation* of different components in digital marketplaces. In the context of this thesis, decentralization refers to the concept of delegating decision-making and activities away from a central authority. Disintermediation reduces or removes the involvement of trusted intermediaries when trading on a digital marketplace.

This thesis introduces innovative approaches to decentralize and disintermediate all aspects of blockchain-based marketplaces. We first identify the five aspects of blockchain-based marketplaces: information management, matchmaking, settlement, fraud management, and identity management. We then design, implement, evaluate, and deploy five decentralized mechanisms. Each introduced mechanism focusses on one or two aspects of blockchain-based marketplaces. For each mechanism, we consider feasibility and real-world deployment as crucial requirements for successful adoption.

In Chapter 1 we identify and describe the five aspects of blockchain-based marketplaces. We outline existing approaches that disintermediate and decentralize these aspects. We then formulate our research questions, describe our research and engineering methodology, and summarize the key contributions of this thesis.

In Chapter 2 we introduce a universal accounting mechanism, named *ConTrib*, to securely store information in decentralized applications. With ConTrib, each peer maintains a personal ledger containing tamper-evident records. A record describes an agreement between peers and links to other records. Fraud, the illegitimate modification of a record in one's personal ledger, is detected by continuously exchanging records and by verifying the consistency of incoming records against known ones. We experimentally show that ConTrib is highly scalable and that fraud can be detected quickly. To highlight the applicability of our work, we perform a two-year deployment trial of ConTrib to address free-riding behaviour in Tribler, our decentralized file-sharing application. We leverage the accounting capabilities of ConTrib for other mechanisms introduce in this thesis.

In Chapter 3 we introduce *MATCH*, a decentralized middleware for fair matchmaking in peer-to-peer markets. MATCH addresses manipulation concerns associated with mar-

ketplaces under central control, namely the ability by the market operator to prioritize, hide, or delay specific orders. By decoupling the dissemination of potential matches from the negotiation of trade agreements, MATCH empowers end users to make their own educated decisions and to engage in direct negotiations with trade partners. We evaluate MATCH with real-world ride-hailing and asset trading workloads. Our experiments demonstrate that MATCH is highly resilient against malicious matchmakers that deviate from a specific matching policy.

In Chapter 4 we introduce a universal and decentralized settlement mechanism named *XChange*. This mechanism enables the exchange of assets between permissioned blockchains without the requirement for a trusted intermediary, collateral deposits, or modifications to deployed blockchain applications. XChange records the progression of each trade within records on a distributed log. By inspecting these records, every participant can detect if a trader is refraining from fulfilling its obligations during an ongoing trade. To address counterparty risk, XChange bounds the economic gains of adversaries that have committed fraud during a trade by preventing them from engaging in other trades. Our evaluation shows that XChange is highly scalable and has low latency and resource overhead. With a real-world trading dataset, we show that our risk mitigation strategies reduce the fraud losses by more than 99%, even in extreme adversarial settings.

In Chapter 5 we introduce *Internet-of-Money*, a settlement mechanism for real-time and international money transfers between different banks. The key idea is to break up a slow inter-bank payment into multiple fast intra-bank payments. Each inter-bank payment with Internet-of-Money uses one or more volunteer-based services, named money routers. A money router possesses multiple bank accounts at different banks. This approach reduces the duration of inter-bank payments from days to mere seconds. To identity fraud, i.e., not forwarding incoming money as a router to the beneficiary, all transfer operations by users and money routers are recorded in a distributed log. To further reduce risks, we break up a single money transfer into multiple smaller ones and leverage multiple money routers in parallel. Our experiments show that Internet-of-Money enables fast inter-bank payments and that our risk mitigation strategies significantly reduce fraud gains by adversarial parties.

In Chapter 6 we introduce *dAppCoder*, a decentralized crowdsourcing marketplace for the development of decentralized applications. dAppCoder addresses fragmentation and lock-in effects associated with centralized marketplaces for crowdsourcing. A key part of dAppCoder is DevID, a blockchain-based identity solution for software developers. DevID unifies developer information within records on a distributed log. Developers can import data assets from third parties into a single DevID portfolio, add projects and skills, and receive endorsements. Clients can leverage the dAppCoder marketplace to create and manage projects, and to directly remunerate developers with cryptocurrencies while avoiding the need for trusted intermediaries. Our user trial demonstrates that both dAppCoder and DevID are efficient at storing and managing data.

Finally, in Chapter 7 we formulate the main conclusions of this thesis and present suggestions for further research directions.

Samenvatting

Markten faciliteren het verhandelen van diensten, goederen en informatie tussen individuen en bedrijven. Ze spelen een essentiële rol in onze economie. De standaard werkwijze om elektronische markten in te richten is door het gebruik van gecentraliseerde infrastructuur die volledig geopereerd en beheerd wordt door een marktexploitant. In dergelijke markten zijn er vaak vertrouwde tussenpartijen die verschillende diensten aanbieden aan gebruikers, zoals het beheren van marktinformatie, het verwerken van betalingen en het uitvoeren van arbitrage bij een geschil.

Innovaties in de informatietechnologie hebben de behoefte aan vertrouwde tussenpartijen in twijfel getrokken. Met name blockchaintechnologie wordt steeds vaker toegepast om elektronische markten te creëren. Blockchain-gebaseerde marktplaatsen faciliteren directe handel tussen gebruikers en verminderen de afhankelijkheid van zowel gezaghebbende partijen als vertrouwde tussenpartijen. De rol van blockchain op dergelijke marktplaatsen is om sociaal vertrouwen te vervangen door cryptografische algoritmes. Dit maakt de *decentralisatie* en *disintermediatie* van verschillende componenten in elektronische marktplaatsen mogelijk. In de context van dit werk betekent decentralisatie het verminderen van besluitvorming en activiteiten die worden uitgevoerd door een centrale autoriteit. Disintermediatie vermindert of verwijdert de betrokkenheid van vertrouwde tussenpartijen bij het handelen op een elektronische markt.

Dit proefschrift introduceert innovatieve mechanismes om alle aspecten van blockchain-gebaseerde markten te decentraliseren en te disintermediëren. We identificeren eerst de vijf aspecten van blockchain-gebaseerde markten: informatiebeheer, matchmaking, de afwikkeling van handel, het afhandelen van fraude en het beheren van identiteit. Vervolgens ontwerpen, implementeren en evalueren we vijf gedecentraliseerde mechanismen. Elk geïntroduceerd mechanisme richt zich op één of twee aspecten van blockchain-gebaseerde markten. Voor elk mechanisme beschouwen we een praktisch nut en een bijhorende implementatie als cruciale vereisten voor een succesvolle acceptatie.

In hoofdstuk 1 identificeren en beschrijven we de vijf aspecten van marktplaatsen die op blockchain gebasseerd zijn. We beschrijven bestaande oplossingen die deze aspecten disintermediëren en decentraliseren. Vervolgens formuleren we onze onderzoeksvragen, beschrijven we onze onderzoeks- en ontwikkelmethodologie en vatten we de belangrijkste bijdragen van dit proefschrift samen.

In hoofdstuk 2 introduceren we *ConTrib*, een universeel mechanisme voor het bijhouden van informatie in decentrale netwerken. Met ConTrib houdt elke gebruiker een persoonlijk grootboek met records bij. Een record bevat een contractuele overeenkomst tussen gebruikers en bevat ook verwijzingen naar andere records in het grootboek. Fraude, het onwettig wijzigen van een record in iemands persoonlijke grootboek, wordt gedetecteerd door continu records uit te wisselen en door de consistentie van inkomende records met reeds opgeslagen records te verifiëren. We laten met experimenten zien dat ConTrib zeer schaalbaar is en dat fraude snel kan worden gedetecteerd. Om de toepasbaarheid

van ons werk te evalueren voeren we een tweejarige pilot uit van ConTrib om meelifters (free-riders) aan te pakken in Tribler, onze gedecentraliseerde applicatie voor het delen van bestanden. We maken gebruik van de mogelijkheden van ConTrib voor andere mechanismen die in dit proefschrift worden geïntroduceerd.

In hoofdstuk 3 introduceren we *MATCH*, een gedecentraliseerde middleware voor eerlijke matchmaking in peer-to-peer markten. MATCH pakt manipulatieproblemen aan die verband houden met marktplaatsen onder centrale controle, namelijk de mogelijkheid om specifieke orders door de marktexploitant te prioriseren, te verbergen of te vertragen. Door de verspreiding van potentiële matches los te koppelen van de onderhandelingen over handelsovereenkomsten, stelt MATCH eindgebruikers in staat hun eigen weloverwogen beslissingen te nemen en directe onderhandelingen met handelspartners aan te gaan. We evalueren MATCH met zowel een ride-hailing als een token trading dataset. Uit onze experimenten blijkt dat MATCH zeer resistent is tegen kwaadwillende matchmakers die afwijken van een specifiek matching beleid.

In hoofdstuk 4 introduceren we een universeel en gedecentraliseerd mechanisme voor settlement, genaamd *XChange*. Ons mechanisme maakt de uitwisseling van tokens tussen blockchains met explicitiet geautoriseerde toegang mogelijk zonder een vertrouwde tussenpersoon, onderpanddeposito's of wijzigingen aan geïmplementeerde applicaties op de blockchain. XChange registreert de voortgang van elke transactie in een gedistribueerd grootboek. Door het gedistribueerde grootboek te inspecteren, kan elke gebruiker detecteren of een handelaar heeft afgezien van het nakomen van zijn of haar verplichtingen tijdens een lopende transactie. Om het tegenpartijrisico te verkleinen, beperkt XChange de economische winsten van kwaadwillige gebruikers die fraude hebben gepleegd door te voorkomen dat ze andere transacties aangaan. Onze resultaten tonen aan dat XChange zeer schaalbaar is en efficient functioneert. Met behulp van een realistische dataset laten we zien dat zelfs in situaties met zeer veel fraudeurs XChange de economische verliezen van gedupeerde gebruikers met meer dan 99% reduceert.

In hoofdstuk 5 introduceren we *Internet-of-Money*, een mechanisme voor realtime en internationale betalingen tussen verschillende banken. Het idee is om een langzame interbancaire betaling op te splitsen in meerdere snelle intrabancaire betalingen. Elke geldoverdracht maakt gebruik van één of meer op vrijwilligers gebaseerde diensten, geldrouters. Een geldrouter beheert bankrekeningen bij verschillende banken. Deze benadering reduceert de duur van interbancaire betalingen van dagen tot louter seconden. Om fraude, d.w.z. het niet doorsturen van inkomend geld als router naar de volgende gebruiker, worden alle betalingen tussen gebruikers en geldrouters geregistreerd in een gedistribueerd grootboek. Om de risico's verder te verkleinen, splitsen we een enkele betaling op in meerdere kleinere betalingen en gebruiken we voor een transactie meerdere geldrouters tegelijkertijd. Onze experimenten tonen aan dat deze aanpak de winsten voor kwaadwillige gebruikers aanzienlijk vermindert.

In hoofdstuk 6 introduceren we *dAppCoder*, een marktplaats voor het crowdsourcen van de ontwikkeling van gedecentraliseerde applicaties. dAppCoder pakt fragmentatie- en lock-in-effecten aan die verband houden met gecentraliseerde marktplaatsen voor crowdsourcing. Een belangrijk onderdeel van dAppCoder is DevID, een op blockchain gebaseerde identiteitsoplossing voor softwareontwikkelaars. DevID verenigt ontwikkelaarsinformatie en slaat deze informatie op in records in een gedistribueerd grootboek. Ontwikkelaars

kunnen gegevens van derde partijen importeren in een DevID portfolio, projecten en vaardigheden toevoegen en aanbevelingen voor vaardigheden ontvangen. Gebruikers kunnen dAppCoder benutten om projecten te maken en te beheren, en om ontwikkelaars te belonen voor hun werkzaamheden zonder tussenpartijen. Onze pilot toont aan dat zowel dAppCoder als DevID efficiënt zijn in het opslaan en beheren van gegevens.

Ten slotte formuleren we in hoofdstuk 7 de belangrijkste conclusies van dit proefschrift en doen we suggesties voor toekomstige onderzoeksrichtingen.

Acknowledgments

While it is often said that pursuing a PhD can be a solitary and tough process, I enjoyed most aspects of it. However, I would not have completed this journey were it not for the continuous support by colleagues, friends, and family!

First, I would like to thank Johan Pouwelse, my promotor and daily supervisor. Under your supervision, there has not been a single day when I ran out of ideas or was unmotivated to continue, even after yet another paper rejection. Our random and unorganized discussions, whether or not with other lab members, have been a great source of inspiration. Your leadership and advice have tremendously helped me to become an independent researcher and to get my ideas published in scientific venues, which was definitely not always easy. I highly value your relentless focus on practicality and usability ("no passing grade without running code"). Also, I am grateful for the opportunities you gave me to (co-)supervise BSc and MSc students, which is an activity I very much enjoy.

I would also like to express my gratitude to my promotor Dick Epema, who was always willing to bring my research work to the next level by identifying both fundamental flaws and minor presentation issues. Your knowledge, together with your thorough and critical feedback, has helped me to improve my writing skills, research papers, and this thesis. Your feedback has also been instrumental in getting various papers published.

I thank all my colleagues for making the office life very enjoyable. Quinten, Egbert, and Sandip: I fondly remember the times we worked on Tribler and the conversations we had in our old office, most of the time around the coffee machine (drinking coffee made up quite a part of my PhD life ☕). Bulat, thank you for the many in-depth discussions we had about our research, science, and many other topics. These conversations have definitely influenced and improved some of the mechanisms in this thesis. Can, Georgy, and Mitchell, thank you for your help as co-author on various papers! I additionally thank Georgy for providing new insights from a socio-technological perspective on the work we produce as a lab, which also helped me to think about the implications of my research beyond distributed systems. Ayman, thank you for inviting me to work together on an energy trading platform; I have learned many new things about your field! Leonard, thank you for occassionally providing feedback on my work from a financial regulatory perspective. I would also like to thank Elric, Vadim, Alexander, and Andrew for our fruitful collaborations on Tribler. Tamara, Kim, and Sophie: thank you for taking care of our administrative burdens! Jan, thank you for providing valuable feedback on the first versions of the MATCH mechanism. I also would like to thank the TU Delft graduate school for organizing a physical defence ceremony, despite the ongoing COVID-19 pandemic.

Besides the people directly involved in my research, I extend my gratitude to a large group of people with whom I spent many Wednesday evenings playing games, eating pizza and/or drinking beer in the /pub and the Doerak Café. These evenings were very helpful in unwinding from the (sometimes hectic) academic life. Otto, as the regular member of this group, thank you for the many conversions and coffee/beer talks that we had! I also

thank you for your technical support when we had issues with our Tribler infrastructure. I would also like to thank Stefan, Jesse, Hans, Ernst, Niels, Maria, Thomas, Liam, Gijs, Tim, Chris, Jetse, Alexander, Sacheendra, Taraneh, and others who occasionally joined for the Wednesday evening activities (there are just too many to list here!).

I am grateful for the many memories I made with friends. Laurens, thank you for the many projects we worked on, the games we played, the movies and tv shows we watched (most of which seem to be cancelled for some reason), and in general, for great memories since we both started our BSc in Delft! Laurens, Anton and Martijn: thank you for the many trips during the summer and winter holidays that we made within and outside Europe, and whether or not accompanied by others. I will never forget that feeling after we (finally) reached the summit of Mount Fuji (and the realization that we still had to go all the way down...)! 🗻

Last but definitely not least, I would like to thank my parents for their continuous support. I would also like to thank Walter for teaching me many new skills during the construction of our "emergency" apartment.

Martijn

1

Introduction

Marketplaces facilitate the exchange of services, goods, and information between individuals and businesses. They play an essential role in our economy, enabling the exchange of value on both a local and a global scale. A large part of all conducted trades proceeds on electronic marketplaces that leverage Internet technology to electronically buy or sell products and services. On electronic marketplaces, users routinely trade with other users with whom they never interacted before, unlike in many physical marketplaces. Amazon and eBay are well-known examples of large-scale electronic marketplaces that facilitate the exchange of goods between buyers and sellers. During the last decades, companies acting in the *sharing economy*, such as Uber and Airbnb, have further expanded the impact of e-commerce by offering global marketplaces for the sharing of personal resources, e.g., cars and houses, with strangers.

The standard approach to devise electronic marketplaces is by deploying centralized infrastructure, entirely operated and managed by an authoritative market operator. This market operator provides the required primitives for bringing buyers and sellers together, for the management of market information (e.g., product listings), and for transaction processing (e.g., by providing payment services). Also, the market operator often acts as a trusted intermediary between buyers and sellers, leveraging its intermediate position to address potential conflicts arising between traders. For example, the ride-hailing company Uber ensures that its drivers are sufficiently qualified to offer their services to passengers, mediates in case of a dispute, and processes all payments made by passengers. Market intermediaries usually charge users for the provided services through transaction fees.

Advancements in information technology, in particular blockchain technology, have challenged the need for both authoritative market operators and trusted intermediaries. The Bitcoin currency, powered by a tamper-proof distributed ledger, has demonstrated that it is possible to build a cash system that is not under the ownership of a financial institution [1]. Similarly, Ethereum enables developers to write legally-binding contractual logic without notaries [2]. As we will elaborate, the notion of *disintermediation*, reducing or removing the need for trusted intermediaries, is closely related to the process of *decentralization* where authority residing in a single entity is re-distributed over multiple entities. There is an increasing amount of research effort to disintermediate different as-

1

pects of electronic marketplaces and to replace centralized components with decentralized solutions, such as distributed ledgers.

This thesis introduces novel mechanisms for the decentralization and disintermediation in blockchain-based marketplaces. We design, implement, evaluate, and deploy five decentralized mechanisms that improve different aspects of blockchain-based marketplaces. These aspects are information management, matchmaking, settlement, fraud management, and identity management. Each introduced mechanism focusses on one or more of these aspects. In the remainder of this introduction, we define the concepts of decentralization and disintermediation in the context of this thesis, and elaborate on the five aspects of blockchain-based marketplaces.

1.1 Decentralization in Blockchain-based Markets

This thesis discusses five decentralized solutions for blockchain-based marketplaces. Therefore, it is important to understand what decentralization means in the context of this thesis and how blockchain-based marketplaces can achieve decentralization.

1.1.1 What is Decentralization?

The digital currency Bitcoin [1] and the anonymous communication protocol Tor [3] are prominent examples of decentralized Internet solutions that have seen successful adoption. At the same time, there is no established, standard definition of decentralization within the context of Internet-deployed systems. Decentralization is defined by Merriam-Webster as "the dispersion or distribution of functions and powers". It describes the process by which decision-making is delegated away from a central, authoritative entity, for example, shifting authority from a government to provinces or municipalities within a country. Decentralization is widely used as a term within different branches of science, including economics, social sciences, and computer science.

In computer science, the term decentralization is increasingly being used to indicate systems where decisions are not taken by a single entity and where the authority is spread over the participants in the system instead. Decentralization is usually accredited as a desirable property of a computer system, reducing censorship threats and raising the bar to manipulate and take down the entire system by adversarial actors [4]. To date, however, the vast majority of popular Internet applications are centralized systems, e.g., YouTube, Netflix and Facebook. During the last decade, these so-called "Big Tech" companies have accumulated an unprecedented amount of power and market share. A key advantage of centralized systems is that they are relatively easy to set up and maintain, in stark contrast to decentralized networks.

1.1.2 Blockchain Technology

Blockchain technology has profoundly shaped the notion of decentralization within the domain of distributed systems [5]. In 2008, Satoshi Nakamoto[1] introduced the Bitcoin cryptocurrency, a peer-to-peer cash system without banks [1]. Bitcoin challenged what has long been thought to be an impossible problem: reaching distributed consensus in open, large-scale networks without trusted intermediaries. At the core of Bitcoin is a

[1] A pseudonym. The real identity behind the pseudonym is unknown.

blockchain, a distributed ledger that is fully secured and maintained by participating users. The blockchain is a chain consisting of blocks, and each block contains one or more transactions. Each block, except for the first one, is equipped with a hash pointer that points to the previous block. This pointer makes the blockchain structure tamper-evident since the modification of historical transactions can efficiently be detected.

The Bitcoin network maintains a blockchain that includes all transactions ever made. In Bitcoin, users submit signed transactions and pay fees to get their transactions included in the blockchain. A particular group of users, also called *miners*, reach *consensus* on which transactions are deemed valid and enter the blockchain. This works as follows: miners periodically propose a new block with transactions to be appended to the current blockchain. Each miner competes with other miners by solving a computational puzzle derived from the proposed block, a process also called *block mining*. This puzzle involves finding a hash that satisfied a certain condition, for example, it has to start with a number of leading zeros. The first miner to present a block with a correct hash to the network can append its proposed block to the blockchain and gets rewarded with a fixed number of Bitcoin (this number decreases over time). Furthermore, the winning miner can claim all transaction fees of the transactions within the proposed block. The parameters in this consensus algorithm, also called *Proof-of-Work* (PoW) or *Nakamoto consensus*, are fixed such that a new block is created roughly every ten minutes in the Bitcoin network.

Despite significant hype surrounding the Bitcoin ecosystem and its tremendous market capitalization ($913 billion at the time of writing), we discuss three limitations of the Bitcoin cryptocurrency. First, the throughput of Bitcoin is theoretically limited to around seven transactions per second. This throughput is by far not sufficient to handle global financial traffic on its blockchain, which usually requires throughputs of tens of thousands transactions per second. For example, the VISA credit card company is processing around 1'700 transactions per second [6]. Second, despite popular belief, the Bitcoin blockchain is not tamper-proof and can be overwritten given enough computing power. In particular, a reorganization of the blockchain occasionally occurs when two blocks are mined roughly at the same time. Therefore, users have to wait for six additional blocks to be mined before their transaction is included with sufficient finality guarantees (which takes around one hour). This makes Bitcoin highly impractical for payments that require quick confirmation [7]. Third, PoW is a resource-intensive algorithm that consumes significant CPU power. There are increasing concerns around the environmental impact of Bitcoin as its block mining process is estimated to consume as much energy as Kansas City [8].

The limitations of Bitcoin and Bitcoin-derived cryptocurrencies have inspired much research into more scalable consensus mechanisms. On the one hand, much research effort concentrates on improving the throughput of PoW, mostly through parameter tuning [9, 10]. On the other hand, entirely new consensus families have been designed that are not based on burning resources. For instance, Proof-of-Stake (PoS) is an alternative consensus family where the creator of the next block can be chosen by combinations of random selection and the wealth (stake) or age of individuals in the network [11]. PoS is more scalable compared to PoW, however, a particular issue is that there is nothing at stake. This means that miners are free to vote for various, possibly conflicting blockchain histories without repercussions when their vote turns out to be incorrect. Delegated Proof-of-Stake (dPoS) is a semi-decentralized consensus algorithm where the members of

1

an elected committee produce blocks in a round-robin fashion [12]. Committee members that fail to produce a block within time can be voted out by other members.

1.1.3 Decentralized Blockchain Applications

The functionality of Bitcoin and early cryptocurrencies is limited to the minting and transfer of blockchain-based currencies to other users. Ethereum [2], a blockchain solution released in 2015, was the first platform that enables developers to write and deploy *smart contracts* on a blockchain ledger. Smart contracts, introduced by Nick Szabo in 1990, are self-enforcing computer programs that are automatically executed and reside on a blockchain [13]. A transaction in Ethereum can deploy a new smart contract on the blockchain or invoke a function of an existing smart contract. Users submitting transactions have to pay gas, the native currency of the Ethereum blockchain, to remunerate active miners. The amount of gas required for a transaction depends on the computations executed by the function invocation, e.g., expensive operations like encryption consume more gas.

Smart contracts enable developers to build decentralized blockchain-based applications, also called *DApps*. The most common DApp on the Ethereum blockchain is an ERC20 contract, which enables developers to issue and manage custom assets [14]. Besides ERC20 tokens, the Ethereum blockchain hosts almost 3'000 DApps at the time of writing, including lotteries, games, asset markets, voting, prediction markets, and decentralized lending solutions.[2] Despite the thriving ecosystem, it is non-trivial to revoke or disable a deployed smart contract when a software bug has been exploited. In 2016, the Ethereum network almost collapsed due to implementation errors in the smart contract that managed The Decentralized Autonomous Organization (The DAO) [15]. A hacker managed to compromise $50 million worth of Ethereum tokens. As a result, the Ethereum foundation decided to split (hard fork) the network in two where one network continues to operate the blockchain affected by the hack, and another network operates on an older version of the Ethereum blockchain that is unaffected by the hack. Since then, there has been much effort to build tools for developers to increase the security and correctness of smart contracts [16, 17].

1.1.4 Centralized Cryptocurrency Exchanges

At the time of writing, there are over 10'000 different cryptocurrencies across hundreds of blockchain platforms.[3] The proliferation of different digital assets has resulted in the deployment of centralized cryptocurrency exchanges, operated by a market authority. On these exchanges, users can trade their cryptocurrency for other cryptocurrencies or fiat currencies such as Dollars or Euros. When using the services of a centralized cryptocurrency exchange, users usually have to deposit their funds into a wallet owned by the market operator for their trade to complete. Users then create orders to buy or sell cryptocurrencies. The market operator matches the buy or sell order with existing orders and transfers the ownership of cryptocurrencies when a trading opportunity has been found. At the time of writing, the biggest cryptocurrency exchange is Binance with a 24-hour trading volume of $33 billion.[4]

[2] See `https://www.stateofthedapps.com`
[3] See `https://coinmarketcap.com/all/views/all/`
[4] See `https://coinmarketcap.com/rankings/exchanges/`

Centralized exchanges can facilitate trade between an extensive range of different blockchains, as long as the market operator maintains wallets on the involved blockchains and can issue transactions in these blockchain networks to transfer the assets. Furthermore, they usually provide a convenient interface to users, making it easy to enter the market and participate in trade. Yet, the idea of having an authoritative market operator responsible for asset exchange conflicts with the vision of blockchain technology, which is to provide open, decentralized ecosystems without trusted intermediaries. Users are required to trust that the market operator does not default and correctly executes a trade on behalf of the user. History has shown that cryptocurrency market operators sometimes lack the required knowledge to quickly scale up their infrastructure to meet increasing demand, leading to platform unavailability or even the inability to withdraw deposited funds from wallets. Furthermore, deposited cryptocurrencies are usually stored in a single location by the market operator, making it an attractive and valuable target for hackers. In 2014, hackers compromised assets worth around $450 million from Mt. Gox, the biggest cryptocurrency exchange at that time [18].

1.1.5 Decentralized Cryptocurrency Exchanges

Blockchain technology is increasingly being used to build decentralized exchanges, or *DEXes* [19]. DEXes enable direct peer-to-peer trading without a market operator. On DEXes, users can create their own assets, transfer owned assets to others, and trade assets with other users by publishing buy and sell orders on the blockchain. These orders are then automatically matched by miners during the validation of new transactions. Usually, a DEX only allows the trading of assets residing on the same blockchain. As we will further elaborate in Section 1.3.2, order matchmaking can also proceed outside the blockchain to increase efficiency. DEXes have become a fundamental component of Decentralized Finance (DeFi), which is an experimental form of finance conducted using blockchain applications [20]. One of the largest DEXes is Uniswap, processing trade worth over $1.1 billion on a daily basis, at the time of writing.[5]

We identify four advantages of DEXes over centralized cryptocurrency exchanges. First, DEXes enhance security during the trading process; a trade is usually an atomic operation, and there is minimal risk of losing funds as long as the underlying blockchain and consensus mechanism remain uncompromised. Second, users themselves remain in control of their funds when trading on a DEX, and they do not have to transfer ownership of their assets to the market operator. Third, the transaction fees associated with trading on a DEX are usually lower compared to a centralized exchange since there is no profit-driven intermediary. Fourth, DEXes allow users to remain anonymous, whereas centralized exchanges often require the validation of one's identity for participation.

We also point out three disadvantages of DEX-based trading. First, many DEXes currently suffer from low liquidity and trading volume, making them less attractive for long-term trading. Second, their peak transaction throughput depends on the consensus model used by the underlying blockchain, which might make particular DEXes unsuitable for bulk trading. Third, as DEXes and blockchains are a relatively new technology, design weaknesses can lead to the loss of funds as demonstrated by a number of recent attacks on DeFi applications [21, 22].

[5] See `https://coinmarketcap.com/rankings/exchanges/dex/`

1

1.2 Disintermediation in Blockchain-based Markets

In many physical and electronic marketplaces, middlemen play a key role in the matching of buyers and sellers, and in the facilitation of transactions between traders [23]. The popularity of electronic commerce and the rise of new business models has resulted in much interest to act as *trusted intermediary* to benefit from the interactions between traders [24]. A well-known example of a trusted intermediary is PayPal [25], a payment service provider for retailers. Besides providing payment services, PayPal can also act as arbitrator when a dispute between a buyer and seller arises. The ability to act as trusted intermediaries is at the core of electronic markets and their services help to ensure that a trade between a buyer and seller who might not necessarily trust each other proceeds without issues.

Despite their prominent role, trusted intermediaries increase the costs for traders since they are usually profit-driven and charge a fee for their services. As such, there is much interest in removing trusted intermediaries from the trading process, or *disintermediation*. Disintermediation is defined by Merriam-Webster as "the elimination of an intermediary in a transaction between two parties". Disintermediation is very much related to the concept of decentralization, specifically, disintermediation requires decentralization as its foundation [26]. The debate around disintermediation in electronic markets dates back to the rise of the Internet itself. The Internet provided infrastructure that offers users quick and convenient access to market information, therefore opening opportunities to remove traditional broker agents whose primary role was to aggregate this market information [27]. A clear example of disintermediation can be found in the book publishing market [28]. Information technology enables book buyers to quickly place their order and allows authors to only print their books when there is actual demand, therefore removing the retailer from the book supply chain.

Bitcoin and subsequent blockchain-related innovations have further challenged the need for trusted intermediaries. By leveraging cryptographic techniques, cryptocurrencies have demonstrated that a decentralized payment system without financial institutions is possible. Since the introduction of Bitcoin, there has been much effort by both industry and academia to critically assess the necessity of trusted intermediaries, and potentially replace them with another mechanism, e.g., using smart contracts on Ethereum [29].

In many domains, it has been proven to be possible to replace trusted intermediaries with cryptographic techniques. Yet, disintermediation is not always possible and sometimes not even allowed. In certain domains there is a need for traditional trusted intermediaries to safeguard business processes, in particular in the highly regulated financial sector. One might argue that electronic markets require at the very least some trusted intermediary to act as mediator between buyer and sellers if a trade is not an atomic operation. Furthermore, local regulations might require a trusted intermediary for certain market processes, e.g., when there is a need to verify the identity of business relations to prevent criminal activities (this process is also known as Know-your-Customer or KYC).

As also pointed out by other researchers, we argue that it is unlikely that electronic markets will be fully disintermediated by blockchain technology anytime soon [30]. Instead of complete disintermediation, it is a more likely scenario that the role of existing intermediaries will transform and that their involvement in market processes will be reduced. For this reason, numerous financial institutions are currently experimenting with distributed ledger technology to make existing settlement services more efficient and reli-

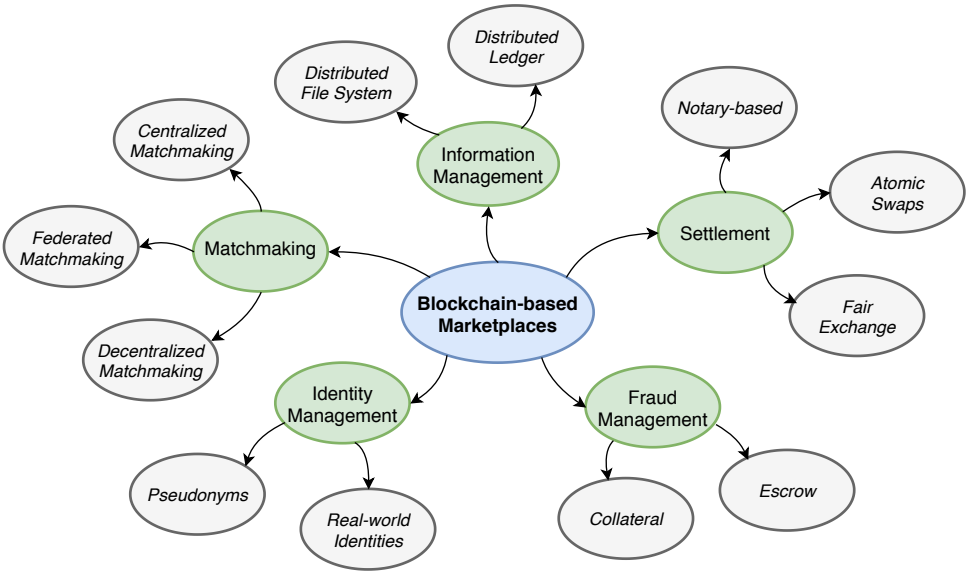


Figure 1.1: The five aspects of blockchain-based marketplaces (coloured in green). For each aspect, we identify existing mechanisms (coloured in grey).

able. Perhaps the most influential solution is Ripple [31], a credit network that is aimed to eventually replace the SWIFT payment infrastructure. Another example is Corda that is currently being developed by R3, a consortium consisting of the world's leading financial institutions [32].

1.3 Aspects of Blockchain-based Marketplaces

So far, we have outlined how blockchain technology is being applied to build decentralized marketplaces and how cryptographic techniques are capable of reducing the role of trusted intermediaries in existing electronic markets. We now shift our focus to the aspects of blockchain-based marketplaces. First, it is crucial to carefully define what a blockchain-based marketplace means in the context of this thesis. We observe that there is much ambiguity around the concept of blockchain-based marketplaces in academic work. This confusion is partially explained by the fact that electronic markets have different aspects, and blockchain technology can be applied to all or a subset of these aspects. For example, OpenBazaar is a decentralized marketplace that leverages blockchain-based cryptocurrencies for peer-to-peer payments between merchants and customers but uses a traditional peer-to-peer network to share product listings amongst participants [33]. In the context of this thesis, we define a blockchain-based marketplace as *a marketplace that leverages blockchain technology to carry out one or more of its critical operations.*

We first break up blockchain-based marketplaces in five different aspects. We then assess how the concepts of decentralization and disintermediation relate to each aspect. Figure 1.1 shows the five aspects of a blockchain-based marketplace, which are *information management*, *matchmaking*, *settlement*, *fraud management*, and *identity management*. This

figure is the result of our literature analysis in which we have studied scientific material on electronic marketplaces that leverage blockchain technology. We decompose each aspect into commonly used mechanisms. In the remainder of this section, we elaborate on each aspect and associated mechanisms.

1.3.1 Information Management

Electronic markets in general require a mechanism to manage and store all market information. This market information includes product listings, outstanding orders, and details on historical transactions (which is often used to estimate the trustworthiness of market participants). Traditional electronic marketplaces take a centralized approach to information management and maintain all market data on their servers. The advantage of this approach is that centralized servers are relatively straightforward to set up and maintain. Furthermore, they enable the market operator to optimize the access to stored information by participants. However, since the operator manages the market information, it is prone to manipulation, e.g., by tampering with or filtering search results.

The GEM system, introduced already in 1999, was one of the first electronic markets where information is stored on different servers, spanning multiple geographic locations [34]. With GEM, each server can be operated by different entities, resulting in a decentralized system architecture. Market autonomy is one of the design goals of GEM: it enables the integration of local markets that operate according to local rules. Except for a few DEXes (e.g., IDEX [35] and EtherDelta [36]), most blockchain-based marketplaces take a decentralized approach to information management and refrain from storing market information on centralized servers. Specifically, we identify two conventional approaches to data storage and dissemination of market information, which are outlined in the remainder of this subsection.

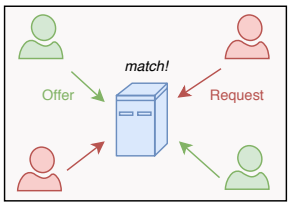
Distributed Ledger

Many blockchain-based marketplaces persist their market information on a distributed ledger, e.g., a blockchain. With this approach, the full market state is stored within transactions on a tamper-proof distributed ledger, secured by a consensus mechanism. Since blockchain is an append-only data structure, no information is ever removed from the distributed ledger. Some blockchains have relatively high storage requirements, for example, the entire Bitcoin blockchain requires around 346 GB of storage at the time of writing.[6] Therefore, some blockchain-based marketplaces deploy one or more *full nodes* that remain synchronized with the network and can be queried by market participants. If a user wishes to avoid dependency on a full node, they are required to download the entire distributed ledger from the network to access the latest market state.

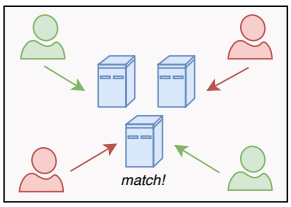
Distributed Filesystem

The high costs associated with storing data on a blockchain has motivated some blockchain-based marketplaces to leverage another storage mechanism besides a distributed ledger. Distributed file systems have proven to be a robust solution for the storage of binary data across a network. There is a wide range of research on using Distributed Hash Tables

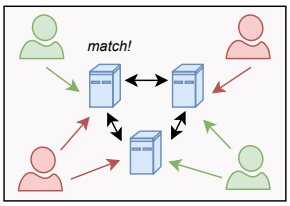
[6] See `https://www.blockchain.com/charts/blocks-size`



(a) *Centralized matchmaking*: a new order is always sent to a single matchmaker.



(b) *Federated matchmaking*: a new order is sent to one of the available matchmakers.



(c) *Decentralized matchmaking* a new order is sent to multiple matchmakers.

Figure 1.2: Three approaches for matchmaking. Traders create offers and requests (coloured green and red respectively), which are matched by matchmakers (depicted in blue).

(DHTs) for the structured storage of key-value pairs [37]. PeerMart, a decentralized marketplace for the trading of Internet resources, leverages a DHT to store pricing information on offered resources [38].

The InterPlanetary File System (IPFS) is a decentralized peer-to-peer network for the storage of and access to files, websites, applications, and data [39]. IPFS builds on top of libp2p [40], a networking framework created for decentralized protocols. OpenBazaar, one of the most popular decentralized marketplaces, builds on IPFS to store and share market information [33]. Filecoin, a decentralized market for file storage, also leverages IPFS to store a subset of all information [41].

1.3.2 Matchmaking

Matchmaking between buyers and sellers is a prerequisite for online trade and therefore essential for any marketplace. It is defined as the process of mediating supply and demand in markets, based on profile information [42].[7] Matchmaking depends on the individual constraints and preferences of market participants. Notable examples are the matching of idle agents to incoming jobs or the matching of suppliers of specific assets to buyers with interest in these assets. Inefficient matchmaking between participants decreases overall market efficiency and customer satisfaction [43]. For example, prolonged suboptimal matching in a ride-hailing market like Uber increases the waiting time for passengers and forces drivers to traverse a greater distance to pick up their customers.

In many blockchain-based markets, a trader can create an *order* to signal their intention to buy or sell assets, resources, or services [42]. This order is then sent to one or more matchmakers. In general, the economic literature distinguishes between two types of orders: *offers*, created by traders offering a specific asset, service, or resource, and *requests*, created by interested buyers. The main objective of a matchmaker is a quick and effective mediation between incoming offers and requests, based on the constraints and preferences included in each order. Matchmakers match incoming offers and requests with other requests and offers, respectively, according to a *matching policy*.

In Figure 1.2 we show three approaches for matchmaking: *centralized matchmaking*,

[7] In multi-agent systems, a matchmaker is considered as an entity that only aggregates offers. Brokers aggregate both offers and requests. We will use the term matchmaker in this thesis since we found it to be more common in related work.

1

federated matchmaking, and *decentralized matchmaking*. Each matchmaker (depicted in blue) is operated by a different user. We now elaborate on each approach.

Centralized Matchmaking
Centralized matchmaking (Figure 1.2a) is the most common solution to match market orders. Traders send new offers and requests to a dedicated matchmaker, usually a centralized system under the control of a single authority. This model is widely adopted by commercialized marketplaces such as stock exchanges (e.g., NYSE or NASDAQ) and resource-sharing markets (e.g., Uber or Airbnb).

Centralized matchmaking with a single server is relatively straightforward to implement since all network communication follows the client-server model, i.e., there is no synchronization required between peers.[8] Also, since all orders are stored and matched by a single matchmaker, orders can be processed based on full market knowledge and therefore matched optimally with existing orders. With centralized matchmaking, the identity behind each order is only disclosed to the market operator, therefore protecting the privacy of individual traders.

The emergence of electronic trading gave rise to fairness, transparency, and manipulation issues during the matchmaking process [44]. For example, with centralized matchmaking, the matchmaker is capable of censoring or delaying specific orders. Information asymmetry between market operators and traders allows matchmakers to exploit their information advantage, e.g., by front-running on specific orders. From a systems perspective, centralized matchmaking has lower scalability compared to decentralized solutions since the matchmaker becomes a bottleneck when more orders are being submitted within the same period [45]. Finally, centralized matchmaking exhibits low fault tolerance: if the single matchmaker becomes unavailable, e.g., due to infrastructure failures, incoming orders cannot be matched and all market activity stalls.

Within the context of blockchain-based marketplaces we find that centralized matchmaking is widely adopted by centralized cryptocurrency exchanges. For DEXes, this model is uncommon since matchmaking can proceed as part of the blockchain logic. Notable exceptions are the Ethereum-based exchanges EtherDelta [36] and IDEX [35] that deploy one or more servers to store and match market orders.

Federated Matchmaking
Federated matchmaking (Figure 1.2b) is an alternative approach where instead of relying on a central matchmaker, multiple (independent) matchmakers individually maintain an order book. The set of matchmakers can either be static, e.g., elected by a committee or some voting mechanism, or dynamic, e.g., each peer can opt-in to become a matchmaker for others. A new order is submitted to one of the available matchmakers, selected by the order creator. The reliability or trustworthiness of individual matchmakers might impact the choice for the preferred matchmaker. When a matchmaker is suspected of mistreating incoming orders, or when the matchmaker provides poor services, traders can entrust their orders to another matchmaker instead. This approach increases robustness against

[8] We acknowledge that centralized matchmaking can be achieved with a distributed system architecture to improve fault tolerance and availability. This is more challenging to implement since it requires coordination between servers. We classify this approach as centralized if the involved servers are under the operation and control of a single authority.

failure of individual matchmakers since a trader can send its order to another available matchmaker in this situation. However, the market orders are now fragmented across different matchmakers, potentially leading to sub-optimal market efficiency compared to centralized matchmaking.

The 0x [46] and Swap [47] trading protocols orient around the trade of Ethereum tokens and have adopted the federated matchmaking model. Both protocols allow any user to act as matchmaker and therefore build an off-chain matchmaking network for orders. We observe, however, that the most used matchmaker is often the one provided by the protocol developers. This makes the added benefit of this approach, compared to centralized matchmaking, questionable.

Decentralized Matchmaking

The main idea of *decentralized matchmaking* (Figure 1.2c) is that a single order is sent to multiple matchmakers simultaneously. In addition, matchmakers are able to synchronize known orders with other matchmakers. This approach is exclusively used in the context of blockchain-based marketplaces, to the best knowledge of the authors. We further distinguish between on-chain and off-chain decentralized matchmaking.

On-Chain. Most DEXes that operate on a blockchain use *on-chain* decentralized matchmaking. This process either relies on a smart contract to match known orders or executes the matchmaking logic as part of the transaction validation. The market orders are embedded in transactions and sent to miners for inclusion on the blockchain. For example, Stellar maintains an exchange on its distributed ledger and allows users to issue buy and sell orders for any asset that is native to the Stellar blockchain [48]. In the same way, the BitShares DEX offers specialized transactions to create new or to cancel existing orders [49].

The main advantage of on-chain decentralized matchmaking is tight integration with the blockchain logic; no additional components are required to process and match orders. However, since users need to pay fees when issuing the transactions to manage their orders, order management can become costly, particularly when done in bulk. Furthermore, matchmaking on a blockchain can be orders of magnitude slower compared to centralized matchmaking due to the need to reach agreement on issued transactions. Finally, on-chain matching protocols do not explicitly store all established matches. Therefore, to reconstruct the order book at a specific block height, one might need to replay all transactions up to that block in the blockchain.

Off-Chain. To lower the costs of order management, some blockchain-based marketplaces maintain the order book *off-chain*. Loopring, for example, is an order sharing protocol where new orders are sent to one or more relays in an off-chain mesh network [50]. Relayers claim the margin between two matched orders, or can alternatively charge a fixed fee for their services. The Republic Protocol builds a decentralized network of nodes that match orders without revealing any information about individual orders [51]. The protocol uses Shamir secret sharing [52] to break down an order into multiple order fragments which are distributed through the network, thus hiding the identity of the order creator and the specifications of created orders.

We identify two advantages of decentralized off-chain matchmaking compared to centralized and federated matchmaking. First, by sharing orders between matchmakers, one

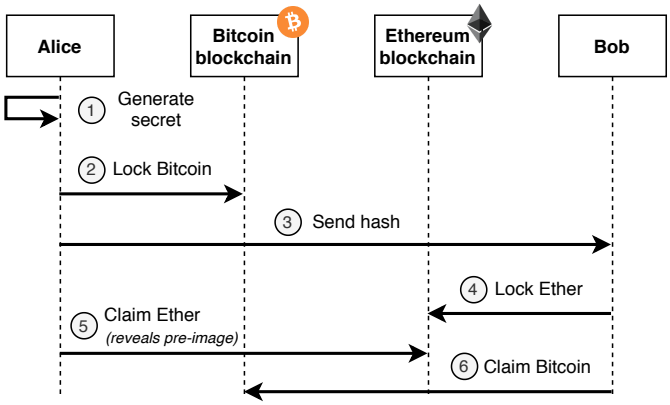


Figure 1.3: Sequence diagram of a successful HTLC-based atomic swap between Alice and Bob.

can achieve similar matching effectiveness compared to centralized matchmaking, depending on how quickly orders are synchronized amongst matchmakers. Second, decentralized matchmaking exhibits high tolerance against the failure of individual matchmakers and can withstand the failure or partition of a subset of all matchmakers. However, this model increases bandwidth usage since orders are replicated over multiple matchmakers. It also might take longer before a new order is fulfilled in the case that it is sent to matchmakers that are unable to match this order immediately.

1.3.3 Settlement

Settlement is the process of fulfilling the obligations by trading parties. In traditional marketplaces and many cryptocurrency exchanges, it is common practice to have a trusted intermediary settle a trade. An asset exchange using a trusted intermediary completes as follows: two parties that agree on a trade first transfer the assets they offer to one of the wallets owned by the trusted intermediary. When this intermediary has received both assets, it finishes the exchange by transferring the appropriate assets to the other party. In this approach, the trusted intermediary holds (temporary) ownership of the assets to be traded. Relying on a trusted intermediary removes the risks when trading directly between the parties, but it requires both parties to have faith that the intermediary does not default or steal their assets.

Blockchain-based marketplaces often refrain from settlement through a trusted intermediary. Instead, they either use cryptographic techniques to ensure trade atomicity or rely on a group of semi-trusted peers to settle a trade. We now outline three settlement techniques commonly found in blockchain-based marketplaces.

Atomic Swaps

The *atomic swap* is a coordination protocol that is commonly used to exchange assets between different blockchains, without need for a trusted intermediary [53]. Atomic swaps enable two parties to exchange blockchain-based assets in an atomic manner. This means that the exchange either completes for both parties and have their assets traded, or it

fails. When the exchange fails, both parties do not suffer an economic loss and retain ownership of the assets involved in the exchange. We remark that the atomicity property of the atomic swap protocol critically depends on the characteristics of the underlying blockchains. If one of the blockchains is compromised by adversaries, or if a chain reorganization occurs, atomicity during asset exchange cannot be guaranteed and one of the parties can lose its funds to the counterparty.

Atomic swaps eliminate the risk of losing assets to an adversarial trader during the exchange. The main idea is that trading users lock their assets in a specialized transaction on the blockchain in such a way that no single party can claim both locked assets. This is achieved with *Hash-Timelock Contracts* (HTLCs), a special transaction that leverages hash locks and time locks. A hash lock is a restriction that prevents the transfer of assets until the pre-image of a provided hash is revealed. A time lock is a primitive that prevents the transfer of assets until a specific time. The latter primitive prevents assets from being locked up indefinitely during an atomic swap. This time lock should be well above the block confirmation time of the underlying blockchain to prevent the loss of assets during a blockchain reorganization. In practice, this value is often fixed to several hours.

We further explain the atomic swap by considering a trade with Bitcoin and Ether (the native token of the Ethereum blockchain). Figure 1.3 visualizes an atomic swap between two parties, Alice and Bob, where Alice sells her Bitcoin in return for Ether. The basic atomic swap, described by Tier Nolan [54], consists of the following six steps:

Step 1. Alice generates a secret value s and computes $H(s)$, where $H(\cdot)$ is a secure hash function.

Step 2. Alice submits a hash-timelock transaction T_1 to the Bitcoin blockchain, locking her Bitcoin and using $H(s)$ for the hash lock. A party can claim the Bitcoin held by T_1 with another transaction that provides s, within a specific time duration.

Step 3. Alice sends $H(s)$ to Bob using any communication medium.

Step 4. Bob submits a hash-timelock transaction T_2 to the Ethereum blockchain, locking his Ether and also using $H(s)$ for the hash lock.

Step 5. Alice claims Bobs' Ether locked in T_2 by submitting a transaction, T_3, to the Ethereum blockchain, containing s. T_3 unlocks the hash-lock in T_2. This reveals pre-image s to Bob.

Step 6. Bob now claims Alice's Bitcoin locked in T_1 by submitting a transaction, T_4, to the Bitcoin blockchain, containing s. The asset exchange is now complete.

The above protocol requires a total of four transactions, two for each involved party. Note how Alice is not able to claim Bobs' assets without providing the opportunity for Bob to claim her assets.

Fair Exchange

Fair exchange is a well-studied technique in computer science and is leveraged by a few blockchain-based marketplaces as settlement mechanism [55]. An exchange is considered fair if both of the parties receive the items they expect, or none of them do. Therefore, the atomic swap can be considered as a fair exchange protocol. The FairSwap protocol ensures a fair exchange of digital goods by leveraging smart contracts and zero-knowledge proofs [56]. The protocol, however, is designed around the exchange of digital commodities and is therefore not usable for generic asset exchange across different blockchains.

1

Optimistic fair exchange algorithms address counterparty risk by relying on the arbitration by a trusted third party when one of the involved traders attempts to cheat [57]. Optimistic fair exchange has been initially applied for the exchange of digital signatures but has recently been leveraged to ensure the execution of a cryptocurrency payment in exchange for a receipt [58]. Such optimistic algorithms, however, require the participation of a trusted third party to resolve disputes.

Notary-based
Notary-based schemes are another settlement approach where the approval by a group of credible nodes (often called notaries) is required to perform some operation. Notary schemes aim to partially alleviate the trust issues arising when relying on a single trusted intermediary through the approval by a group of semi-trusted notaries instead. These notaries reach consensus on the occurrence of particular events, e.g., on the inclusion of a transaction on a distributed ledger. Compared to an asset exchange coordinated by a trusted intermediary, notary schemes assume a weaker trust model. Specifically, they can usually withstand adversarial behaviour of a fraction of all notaries such as collusion.

The Interledger project, pioneered by Ripple, is the most advanced approach in this direction [59]. Interledger proposes a notary-based protocol to conduct payments across different ledgers. In atomic mode, these payments are realized through atomic swaps and are coordinated by a different group of notaries for every involved blockchain. Interledger uses payment paths where additional intermediate platforms and their notaries are used to exchange assets between ledgers that do not have a direct connection. Interledger also supports bidirectional asset exchange but is vulnerable to a fraction of notaries colluding with one of the trading parties.

1.3.4 Fraud Management

The management of fraud is a crucial requirement for any marketplace and is closely related to the settlement process. The risk of fraud typically occurs when a buyer and seller have never interacted before and therefore do not have a prior trust relation. A common type of fraud is *counterparty fraud*, where a party does not fulfil its obligation towards the counterparty during the settlement of a trade, e.g., by not delivering the promised assets or goods. In centralized marketplaces, this kind of fraud is often resolved by the market operator, acting as arbitrator during the dispute resolution process. For example, the market operator can communicate with both parties and take appropriate measures when enough evidence has been collected, e.g., suspending the account of a fraudulent user. Within DEXes, however, counterparty fraud is prevented since the execution of a single blockchain transaction that transfers assets is atomic: either both trading parties receive their assets, or nothing happens. Fraud management, however, becomes instrumental when the settlement process is not atomic and requires both involved traders to move value to the counterparty manually. We identify two conventional approaches to manage fraud arising during a non-atomic trade in blockchain-based marketplaces: using escrow services and collateral.

Escrows
Some blockchain-based marketplaces are using a third-party escrow service when a dispute arises. This escrow may be a single entity, e.g., another user in the marketplace, or

a group of users with some authority to resolve the dispute. In the Bisq decentralized exchange, for example, users can make a call on mediators or arbitrators to resolve fraud [60]. Mediators attempt to resolve the dispute but do not have authority over the funds being traded. Arbitrators, however, can redistribute traded assets through the usage of multi-signature techniques.

Collateral

Some blockchain-based marketplaces require users to deposit collateral before trading. This collateral is slashed when its depositor does not adhere to an agreement or deviates from the protocol. The XClaim protocol, for example, relies on collateral deposits to enable asset trading between distinct blockchain ledgers and to incentivize users to behave in line with the system rules [61]. When a participant misbehaves, the collateral is slashed and wronged actors are reimbursed.

1.3.5 Identity Management

The final aspect of blockchain-based marketplaces is how to manage the digital identities of participants. Traders enter a blockchain-based marketplace under a digital identity and subsequently use this identity to participate in the market. We discuss two approaches to identity management in blockchain-based marketplaces: using pseudonyms and using real-world identities.

Pseudonyms

A key property of blockchain technology is the ability to join the network under a pseudonym, a disguised identity usually in the form of a cryptographic keypair. This keypair is generated by users themselves. Users are then identified by their public key, and ensure authenticity of their transactions by digitally signing the transactions with their private key. Since trading on a specific DEXes is constrained to a single market environment, DEXes do not require identity verification and allow traders to participate under a pseudonym.

Real-world Identities

In traditional electronic marketplaces, the digital identity under which a user operates is usually linked to a real-world identity [62]. Identity validation in electronic marketplaces has several purposes. First, it ensures accountability of one's actions within the market in case of a dispute between a buyer and seller. Second, it prevents the situation where a user can easily re-enter the market under a different identity after having committed fraud. Third, identity verification is often part of the regulatory compliance of market operators, as often required by anti-money laundering policies imposed by governments or supervisory authorities. For example, eBay requires its users to go through an identity verification process before they can buy or sell goods on the platform. Similarly, some blockchain-based markets such as Bisq [60] support the trade of fiat currency for cryptocurrencies and are therefore required to conduct additional security checks. In addition, many centralized cryptocurrency exchanges require user verification since these exchanges often allow payments with fiat money, which is more strictly regulated.

1

1.4 Research Questions

In this thesis we focus on decentralization and disintermediation of the five identified aspects of blockchain-based marketplaces. The overarching research question of this thesis is as follows:

How can all aspects of blockchain-based markets be decentralized and disintermediated?

To answer our overarching research question, we formulate and address the following five research questions:

[RQ1] How can a scalable and decentralized mechanism for the storage and dissemination of market information be built? Adequate management of information is essential for electronic marketplaces. Traditional marketplaces store all market orders and product listings on a (centralized) server, which is prone to manipulation by the market operator. Blockchain-powered decentralized exchanges persist all market information on a distributed ledger but this approach suffers from scalability limitations. Our goal is to build a scalable and decentralized storage mechanism in which participants themselves manage all market information.

[RQ2] How can market orders efficiently and fairly be matched without a centralized matchmaker? Order matchmaking in electronic markets is predominantly performed by a centralized server, owned by the market operator. This approach, however, enables the operator to delay, hide, or prioritize incoming orders, resulting in an unfair system. Leveraging blockchain technology to perform order matchmaking has the potential to address these fairness issues but is not scalable enough for usage by many marketplaces. We aim for an efficient matchmaking mechanism with fairness guarantees while avoiding centralized coordination by a market operator.

[RQ3] How can assets securely be exchanged between any permissioned blockchain without a trusted intermediary? Permissioned blockchains are gaining popularity to manage real-world asset within industrial domains such as supply chain management. While the number of permissioned blockchains is proliferating, there is no universal mechanism to quickly exchange assets between different ecosystems without using a trusted intermediary. We aim for a universal settlement mechanism that is capable of securely exchanging assets between any permissioned blockchain.

[RQ4] How can the settlement durations of (international) bank payments be reduced? Secure trade settlement is a key requirement for electronic marketplaces. Trade often involves real-world currencies that are managed by a bank. A major problem of current banking systems is that the settlement duration of an international payment between two different banks is significant and can take days to complete. Furthermore, these payments often require disproportional transaction fees to cover back-office costs. Our goal is to reduce the settlement duration of international payments between banks.

[RQ5] How can a decentralized crowdsourcing platform for the development of dApps be built? The engineering of decentralized applications, or dApps, is a challenging task and requires engineers with appropriate qualifications. Crowdsourcing is getting increasingly popular for software development. At the same time, the credentials of developers are fragmented across many crowdsourcing platforms and vendor-locked. This makes it challenging to get a representative impression of one's skills and introduces

RQ	Mechanism	Evaluation	Deployment
1	ConTrib	DAS5 + simulations	integration in Tribler
2	MATCH	DAS5	integration in Tribler
3	XChange	DAS5 + simulations	integration in Tribler
4	Internet-of-Money	DAS5	in-house Android app
5	dAppCoder + DevID	user trial	in-house desktop application

Table 1.1: An overview of evaluation and deployment methods for each mechanism introduced in this thesis.

search frictions. Our goal is to address this problem by unifying developer credentials and by building a decentralized crowdsourcing platform without fragmentation and vendor-locking effects.

1.5 Research and Engineering Methodology

Blockchain is a relatively young technology of which research is conducted in economics, computer science, and social sciences. We identify two main research approaches in this field. On the one hand, published blockchain research in systems-oriented conferences adopts experimental methods where the performance of proposed mechanisms is evaluated using a comprehensive set of experiments and benchmarks. On the other hand, there is a vast body of theoretical research on the security aspects of distributed ledgers and their consensus algorithms. This body of research tends to follow a theoretical approach through bound analysis, formal proofs, or protocol simulations.

In this thesis we adopt an experimental approach where we answer each research question by designing, implementing, and evaluating a decentralized mechanism that targets different aspects of blockchain-based marketplaces. For all of our proposed mechanisms, our primary objective is to build a solution that is ready for deployment and usable by end users. We have implemented each mechanism in the Python programming language, usually within a few thousand lines of code. We utilize an existing library for networking primitives and decentralized overlay engineering, fully developed and maintained by our lab [63]. Table 1.1 lists for each mechanism how we have evaluated and deployed it. Except for dAppCoder (and by extension, DevID), we evaluate each mechanism with an appropriate set of experiments using our nation-wide compute cluster (the DAS5 [64]). To set up and execute these experiments we make use of the Gumby framework, developed and maintained by our lab [65]. We further evaluate the ConTrib and XChange mechanisms using discrete-event simulations. These simulations enable us to evaluate a mechanism with many users and allows us to quickly replay longitudinal real-world traces. For example, we evaluate fraud gains of adversaries in the XChange mechanism by replaying a week of buy and sell orders with our simulator.

Driven by our focus on practicality, we have deployed each mechanism proposed in this thesis, either internally within our department or with an integration in the Tribler application [66]. Tribler is our open-source, academic research vehicle and offers decentralized, anonymous file-sharing capabilities. Tribler has been downloaded by over 1.7 million users and enables us to test our ideas and algorithms in a geo-distributed, real-world environment and on consumer-grade hardware. For example, we have integrated the ConTrib mechanism in Tribler to prevent free-riding behaviour in our anonymous

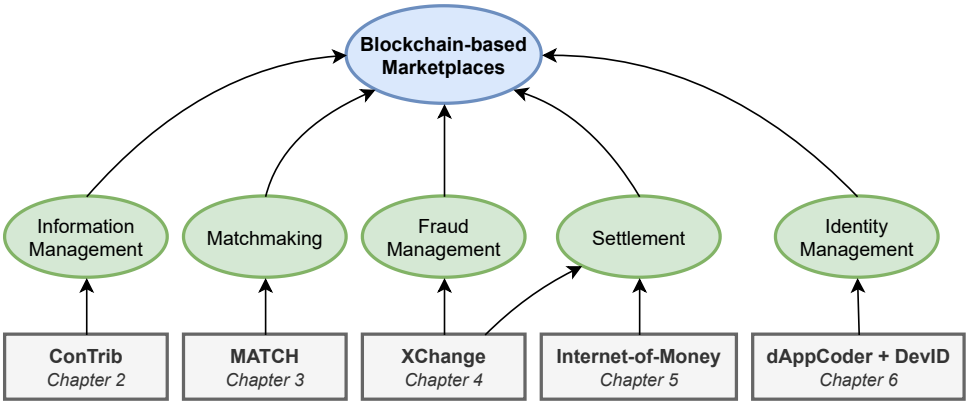


Figure 1.4: The five decentralized mechanisms presented in this thesis, in the context of blockchain-based marketplaces and the identified aspects.

peer-to-peer overlay. This two-year deployment period of ConTrib has allowed us to refine our ideas and incrementally improve our accounting mechanism. We attempt to make our research reproducible by publishing the implementation of each mechanism, including documentation and unit tests, on GitHub.[9] Due to legal considerations, we are unable to provide an open-source implementation of the four reverse-engineered banking algorithms included in our Internet-of-Money mechanism.

Dealing with the complexities of real-world distributed systems often posed a challenge when deploying our mechanisms. For example, shortly after the initial deployment of the MATCH mechanism in Tribler, students forged a specific network packet that would crash a Tribler instance that received it. Tribler would, however, forward the packet to other connected peers before crashing. Therefore, this bug affected a significant part of the Tribler user base, and we were forced to quickly deployed a fix for this behaviour by releasing a new version. Despite the large efforts required to get our systems deployed and operational, this process has been tremendously helpful in discovering both critical design flaws and minor implementation errors.

1.6 Thesis Outline and Contributions

In Chapter 2-6, we address the research questions stated in Section 1.4. Figure 1.4 shows the mechanisms introduced in this thesis and visualizes the aspect(s) that each mechanism relates to. We do note that fraud management in Figure 1.4 refers to the management of counterparty fraud during a trade. Other mechanisms introduced in this thesis also address fraud but these types of fraud are not related to trade settlement and are therefore not visualized in Figure 1.4. The content and contributions in each chapter are as follows:

[Chapter 2] ConTrib: Maintaining Fairness in Decentralized Big Tech Alternatives by Accounting Work. In this chapter we address RQ1 and build a universal mechanism for the accounting of information in decentralized applications. We present

[9]Links to the implementation are provided in the technical chapters of this thesis.

ConTrib, a scalable mechanism for the accounting of interactions within a decentralized network. Each individual in ConTrib maintains a personal ledger with tamper-evident records. Records can point to other ones and can be agreed on by other users, resulting in a global DAG structure. Fraud, the illegitimate modification of a record, is effectively detected since users continuously share and validate records. We devise a system architecture with flexible validation and fraud policies. Our evaluation reveals that ConTrib is highly scalable, tolerates packet loss, and exhibits relatively low fraud detection times. To highlight the potential of ConTrib, we leverage our solution for bandwidth accounting in the Tribler application and successfully address free-riding behaviour. Our two-year deployment trial has resulted in over 160 million records, created by more than 94'000 Internet volunteers. We make use of the accounting capabilities of ConTrib for other mechanisms introduced in this thesis, namely XChange, Internet-of-Money, and dAppCoder. This chapter is based on the following two publications:

Martijn de Vos and Johan Pouwelse, "ConTrib: Universal and Decentralized Accounting in Shared-Resource Systems", *Distributed Infrastructure for Common Good (DICG), 2020.*

Martijn de Vos and Johan Pouwelse, "ConTrib: Maintaining Fairness in Decentralized Big Tech Alternatives by Accounting Work", *Computer Networks*, 2021, Elsevier.

We have presented an earlier version of the ConTrib mechanism (named TrustChain) in the following article:

Pim Otte, Martijn de Vos and Johan Pouwelse, "TrustChain: A Sybil-resistant Scalable Blockchain", *Future Generation Computer Systems (FGCS), Special Issue on Cryptocurrency and Blockchain Technology*, 2020, Elsevier.

In comparison to our earlier article on TrustChain, the articles presenting the ConTrib mechanism contain additional details on the data structure, include a full description of our fraud detection algorithm, and include the results of our two-year deployment trial within Tribler. The XChange, Internet-of-Money, and dAppCoder mechanisms use the TrustChain data structure as presented by Otte et al.

[Chapter 3] MATCH: A Decentralized Middleware for Fair Matchmaking in Peer-to-peer Markets. In this chapter we address RQ2 and focus on matchmaking, the process of bringing market participants together based on individual preferences. Matchmaking is a core enabling element in peer-to-peer markets. To date, matchmaking is predominantly performed by proprietary algorithms, fully controlled by market operators. This raises fairness concerns as market operators effectively can hide, prioritize, or delay the orders of specific users. Blockchain technology has been proposed as an alternative for fair matchmaking without a trusted operator but is still vulnerable to specific fairness attacks like front-running. We present *MATCH*, a decentralized middleware for fair matchmaking in peer-to-peer markets. By decoupling the dissemination of potential matches from the negotiation of trade agreements, MATCH empowers end users to make their own educated decisions and to engage in direct negotiations with trade partners. This approach makes MATCH highly resilient against malicious matchmakers that deviate from

1

a specific matching policy. We implement MATCH and show the resilience of our middleware using real-world ride-hailing and asset trading workloads. The author of this thesis has collaborated with Georgy Ishmaev, who framed the research problem that MATCH addresses within the broader scope of socio-ethical implications of matching platforms for peer-to-peer markets. This chapter is based on the following publication:

Martijn de Vos, Georgy Ishmaev and Johan Pouwelse, "MATCH: a Decentralized Middleware for Fair Matchmaking in Peer-to-peer Markets", *Middleware, 2020.*

[Chapter 4] XChange: A Universal Mechanism for Asset Exchange between Permissioned Blockchains. In this chapter we address RQ3 and present a universal mechanism for asset exchange between permissioned blockchains. Permissioned blockchains are increasingly being used as a solution to record transactions between companies. Several use cases that leverage permissioned blockchains focus on the representation and management of real-world assets. Since the number of incompatible blockchains is quickly growing, there is an increasing need for a universal mechanism to exchange, or trade, digital assets between these isolated platforms. There currently is no universal mechanism for inter-blockchain asset exchange without a requirement for trusted authorities that coordinate the trade. We address this shortcoming and present the *XChange* mechanism. To achieve universality and to avoid trusted authorities that coordinate a trade, XChange does not provide atomic guarantees but leverages risk mitigation strategies to reduce value at stake. Our mechanism records the specifications and progression of each trade within records in a distributed log. XChange reduces the economic gains of adversaries by bounding the total amount of fraud they can commit at any time. After having committed fraud, an adversary is forced to finish its ongoing trades before it can engage in new trades. We first present a four-phased protocol that coordinates an asset exchange between two traders. We then outline how trade records can be stored on the distributed TrustChain ledger. We implement XChange and conduct experiments. Our experiments demonstrate that XChange is capable of reducing the economic gains of adversaries by more than 99.9% when replaying a real-world trading dataset. A deployment on low-resource devices reveals that the additional trade latency induced by XChange is only 493 milliseconds. Finally, our scalability evaluation shows that XChange achieves over 1'000 trades per second and that its throughput, in terms of trades per second, scales linearly with the system load. This chapter is based on the following publication:

Martijn de Vos, Can Umut Ileri and Johan Pouwelse, "XChange: A Universal Mechanism for Asset Exchange between Permissioned Blockchains", *World Wide Web Journal, Springer, 2021.*

We have presented an early version of the XChange mechanism in the following publication:

Martijn de Vos and Johan Pouwelse, "XChange: A Decentralized, Blockchain-based Mechanism for Generic Trade at Scale", *ninth Erasmus Liquidity Conference*, 2019 (no proceedings).

The above publication presents a full architecture for a decentralized marketplace, in-

cluding an early version of the MATCH mechanism.

[Chapter 5] Internet-of-Money: Real-time Money Routing by Trusting Strangers with your Funds. In this chapter we answer RQ4 and reduce the settlement duration of inter-bank payments. The key idea is to break up a particular inter-bank payment into a series of intra-bank payments between strangers which are quick to complete. Specifically, we address the challenging problem of giving money to others and relying on them to forward it. To identify fraud, we record money transfers between interacting strangers on a scalable, distributed ledger. This work represents a small step towards a generic infrastructure for trust, moving beyond proven, single-vendor platforms like eBay, Uber, and Airbnb. Expanding upon trust relations, we design, implement, and evaluate a decentralized overlay network: *Internet-of-Money*. Internet-of-Money is capable of real-time money transfers to different banks by routing funds through money routers. A money router manages bank accounts at different banks. This removes the need for central banks to handle a payment. Our network reduces traditional payment durations from a day or even a few days in weekends, to mere seconds. With real-world experimentations, we prove that Internet-of-Money enables fast money forwarding. We also show that our overlay network is capable of discovering a majority of available money routers well within a minute, ensuring quick availability for end users. Finally, we demonstrate how the profit of cheating routers is limited and that misbehaviour is punished. This chapter is based on the following publication:

Martijn de Vos and Johan Pouwelse, "Real-time Money Routing by Trusting Strangers with your Funds", *IFIP Networking, 2018*.

[Chapter 6] dAppCoder: A Decentralized Marketplace for dApp Crowdsourcing. In this chapter we answer RQ5 and build a crowdsourcing platform for the development of decentralized applications. Decentralized applications, also known as dApps, are the new paradigm for writing business-critical software that runs on a blockchain. Recruiting developers with appropriate qualifications and skills for this activity is key, yet challenging. The main problem is that the portfolio of developers is usually scattered across centralized platforms like GitHub and LinkedIn, and vendor locked-in. This can result in an incomplete impression of their capabilities, introducing search frictions. We address this problem and first introduce *DevID*, a blockchain-based portfolio for developers. This portfolio enables developers to build up a trustworthy collection of records over time that showcase their capabilities and expertise. They can import data assets from third parties into a unified DevID portfolio, add projects and skills, and receive endorsements from other users. All portfolio records are managed by developers themselves and stored on an existing scalable distributed ledger named TrustChain. The essential idea of TrustChain is to exploit the tamper-proof property of the blockchain while avoiding the need to reach a resource-intensive agreement on all transactions. We then build a decentralized crowdsourcing platform, named *dAppCoder*, for the development of dApps. On dAppCoder clients are able to submit their ideas and developers can find work. dAppCoder utilizes DevID portfolios to match these clients and developers. We fully implement our ideas and conduct a deployment trial. Our trial demonstrates that DevID is efficient at storing portfolio records. The author of this thesis has collaborated with Mitchell Olsthoorn,

who helped with developing the conceptual idea, writing the article, and conducting the user trial. This chapter is based on the following publication:

Martijn de Vos, Mitchell Olsthoorn and Johan Pouwelse, "DevID: Blockchain-based Portfolios for Software Developers", *IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON'19)*.

Chapter 6 contains various improvements compared to the above publication. Specifically, we have reorganized the storyline to focus more on the dAppCoder platform, and we have included more technical details on TrustChain transactions and their validation. We have also removed the dependency on the trusted notary service for payments. Clients now issue direct payouts to developer using cryptocurrencies while ensuring that misbehaviour (i.e., not compensating a particular developer for their work) can be detected.

[Chapter 7] Conclusions. We end this thesis with the conclusions, a summary of the lessons learned, and suggestions for further work.

2

ConTrib: Maintaining Fairness in Decentralized Big Tech Alternatives by Accounting Work

"Big Tech" companies provide digital services used by billions of people. Recent developments, however, have shown that these companies often abuse their unprecedented market dominance for selfish interests. Meanwhile, decentralized applications without central authority are gaining traction. Decentralized applications critically depend on its users working together. Ensuring that users do not consume too many resources without reciprocating is a crucial requirement for the sustainability of such applications.

In this chapter we present ConTrib, a universal mechanism to maintain fairness in decentralized applications by accounting the work performed by peers. In ConTrib, participants maintain a personal ledger with tamper-evident records. A record describes some work performed by a peer and links to other records. Fraud in ConTrib occurs when a peer illegitimately modifies one of the records in its personal ledger. This is detected through the continuous exchange of random records between peers and by verifying the consistency of incoming records against known ones. Our simple fraud detection algorithm is highly scalable, tolerates significant packet loss, and exhibits relatively low fraud detection times. We experimentally show that fraud is detected within seconds and with low bandwidth requirements. To demonstrate the applicability of our work, we deploy ConTrib in the Tribler file-sharing application and successfully address free-riding behaviour. This two-year trial has resulted in over 160 million records, created by more than 94'000 users.

2.1 Introduction

Over the last decades, "Big Tech" companies have obtained an unprecedented market dominance in the industry for information technology [67]. Companies such as Google, Amazon, Facebook, and Apple are omnipresent in our current society and even have the means of acting as small states, inhabited by billions of users worldwide. By continuously broadening their activities, these companies seek to expand their virtual territory and seek to obtain monopolistic control over the enabling elements for digital services, such as access to the Internet [68].

The societal impact of "Big Tech" companies is a double-edged sword. On the one hand, these companies are facilitating new modes of digital interaction between users and enable new business models. The sharing economy is a prime example of these phenomena. It is made up by digital markets for the trustworthy exchange of personal assets (e.g., houses and cars) between strangers [69]. Sharing personal assets is a concept that has long been confined to trusted individuals, such as family and friends [70]. Likewise, media platforms such as YouTube provide the required infrastructure for new forms of user engagement through video weblogging or "vlogging".

On the other hand, it has become apparent that "Big Tech" companies tend to exploit their established market position and are increasingly involved in regulatory or political battles. This behaviour sometimes goes undetected for years. For example, researchers have only recently demonstrated that Uber actively manipulates the matchmaking process between passengers and drivers for commercial interests, therefore decreasing platform fairness and income equality of drivers [71]. Similarly, Apple is currently under antitrust investigation by the European Commission that is assessing whether Apples' rules for developers on the distribution of apps via the App Store violate competition rules [72].

These concerning developments have contributed to an increase in the deployment of *decentralized* applications. Decentralized applications avoid centralized ownership and delegate the decision-making away from a single authority. A decentralized application mainly operates through the direct cooperation and information exchange between users, which we call *peers*. Arguably, Bitcoin is the most influential solution in this direction and provides a decentralized cash system without the supervision by an authoritative bank [1]. The underlying data structure of Bitcoin, a blockchain, is at the core of numerous decentralized applications [73]. At the time of writing, there are thousands of decentralized applications deployed on the Ethereum blockchain alone [2]. These decentralized applications include marketplaces, auctions, voting systems, lotteries, and games.

In contrast to the applications deployed by "Big Tech" companies, decentralized applications are fully maintained by peers, without coordination by a third party. Decentralized applications require peers to pool their computer resources to provide the desired services to participants. Specifically, peers have to communicate with other peers, have to dedicate computational power to process incoming network messages, and frequently have to store data generated by other peers. Some decentralized applications critically depend on the voluntary contribution of computer resources by peers. Bitcoin, for example, prevents the uncontrolled minting of digital coins through a resource-based consensus mechanism executed by miners [1]. These miners continuously attempt to solve a computational puzzle, a resource-intensive task that decides who can append transactions to the blockchain ledger. Another volunteer-based application is Tor, providing anonymity by routing In-

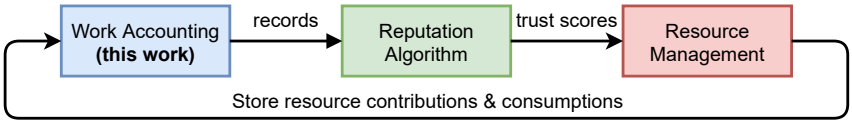


Figure 2.1: Addressing fairness issues in decentralized networks through work accounting, reputation and resource allocation. This work introduces a lightweight mechanism for secure work accounting.

ternet traffic through user-operated relay and exit nodes [74].

Unfortunately, long-term cooperation between peers in decentralized applications is non-trivial to achieve. Not rewarding peers for performing work can result in an unfair situation where peers enjoy the services provided by others, without contributing computer resources in return. This detrimental behaviour, also called *free-riding*, can degrade network health in the long term, as dedicated peers will ultimately leave [75]. Measurements have shown that free-riding often prevails in cooperative applications such as BitTorrent and Tor [76]. Since the cooperation between peers is at the heart of decentralized technology, we argue that this form of fairness is a crucial requirement for *any* decentralized application to ensure long-term sustainability [77]. With the renewed interest in decentralized alternatives for "Big Tech", ensuring fairness in decentralized applications is a significant challenge.

A promising approach to address these fairness issues is by deploying a decentralized reputation mechanism, and allocate resource based on trust scores of individuals. This process is visualized in Figure 2.1. First, users account all performed and consumed work in the network within records. A reputation mechanism then computes trustworthiness scores of users, based on created records. A user decides who to help based on a resource management algorithm. In general, users with low reputation scores should be refused services whereas trusted users enjoy preferential treatment from others. There currently is no accounting mechanism that is specifically built to account work performed and consumed by peers in decentralized networks, to the best of our knowledge.

Our Solution. We specifically focus on the accounting of work performed by peers, which is crucial to ensure fairness within decentralized applications. In this work we design, implement, and evaluate a universal data store, named ConTrib. ConTrib is capable of accounting work within different decentralized applications. Examples of work include storing files on behalf of other peers, performing computations, or relaying network packets. With ConTrib, each peer maintains a *personal ledger* with tamper-evident *records*. The ConTrib records can then be used by an application to determine the trustworthiness of individuals, e.g., with a reputation algorithm. Consequently, users have a natural incentive to increase their social standing by modifying or removing records. This misbehaviour is a key threat to the integrity of the ConTrib data structure. We refer to the illegitimate modification of a record as fraud. To detect fraud, peers continuously request random records from other peers and disseminate newly created records in the network. Peers verify the consistency of incoming records with the ones stored in their database.

ConTrib enables connected applications to select which work should be accounted. Figure 2.2 shows how a decentralized application can leverage ConTrib to account work. By inspecting the records in personal ledgers, an application can gather evidence of free-

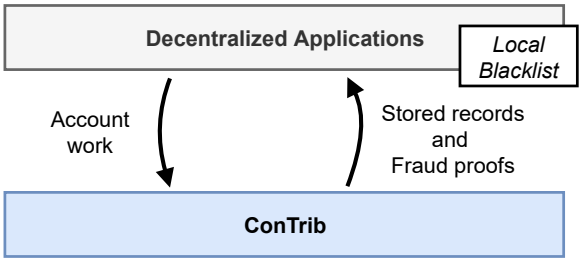


Figure 2.2: Decentralized applications can use ConTrib to account the work performed by peers within tamper-evident *records*. These records are used by connected applications to detect free-riders and fraudsters, which are added to a local blacklist. Applications can then choose to refuse services to the peers on the blacklist.

riding behaviour. Each application maintains a local blacklist with both free-riders and peers that have committed fraud. Peers refrain from performing work for peers on the blacklist. ConTrib can be deployed to alleviate fairness concerns in a wide range of decentralized applications, including peer-assisted video distribution, anonymous communication networks, and distributed learning environments.

We implement ConTrib and evaluate how different parameters impact the efficiency of fraud detection and the network usage. We find that fraud can be detected within seconds on average, even in larger networks with 10'000 interacting peers where every peer commits fraud, and under a conservative strategy for record exchange. We also show that ConTrib is highly resilient against packet loss.

To show the effectiveness of ConTrib in a realistic environment, we employ our accounting mechanism to address free-riding behaviour in Tribler. Tribler is a decentralized application downloaded by over 1.7 million users [78]. We specifically use ConTrib to account bandwidth exchanges in Triblers' Tor-like overlay and use the accounted work to refuse services to free-riders. Our two-year measurements have resulted in over 160 million records, created by more than 94'000 users. This large-scale deployment trial is a key milestone in our ongoing research effort to solve the tragedy-of-the-commons within Internet communities [79].

The main contribution of this work is four-fold:

1. ConTrib, a *universal mechanism* that maintains fairness in decentralized applications by accounting work (Section 2.3).

2. An efficient *fraud detection mechanism* to detect the illegitimate tampering of created records in ConTrib (Section 2.4).

3. An *implementation* and *evaluation* of ConTrib with up to 10'000 peers, demonstrating the scalability of our mechanism and showing that fraud can be detected within seconds on average (Section 2.5 and Section 2.6).

4. A *two-year deployment trial* of ConTrib in Tribler, involving 94'000 Internet-recruited volunteers. This trial successfully addresses free-riding behaviour in Tribler (Section 2.7).

2.2 Background and Problem Description

This work addresses fairness issues in decentralized applications, in particularly free-riding behaviour. Many decentralized applications integrate a mechanism to reward peers for performing work [80]. We first outline two incentive mechanisms that address free-riding by peers, namely trade-based and trust-based incentives [81].

2.2.1 Trade-based Incentives

With trade-based incentives, performed work by peers is remunerated using a credit or payment system. Peers that use the services of other peers are required to pay for that service. Remuneration either occurs immediately after the work is performed or when a certain number of payments is outstanding. The accrued credits can either be converted to real-world money or are merely useful to show the dedication of a particular peer. BOINC is a well-known volunteer computing project that rewards users with virtual credits for processing scientific workloads [82].

Blockchain technology also relies on financial remuneration to keep the system secure [83]. Miners, dedicated peers that maintain the blockchain ledger, are often financially rewarded for their efforts. Specifically, users pay a small fee for each issued transaction and miners then collect these fees when including their transactions in the blockchain. Other decentralized applications have adopted cryptocurrencies as a payment system to reward the performed work. Filecoin is a decentralized system where users pay with a blockchain-based token to have their data stored by other peers [41]. Likewise, TorCoin proposes a mechanism where the relay and exit nodes "mine" a Bitcoin-derived cryptocurrency by relaying Internet traffic [84].

Even though trade-based incentives are frequently used to incentivize work, remuneration is not an adequate solution for any decentralized applications, for the following three reasons [85]. First, they require the integration of a secure payment infrastructure which complicates the system design and potentially enables new forms of attack, such as coin forgery and double-spending. Using a central authority to keep track of each peer's balance introduces a central component and poses a single-point-of-failure. Second, remuneration requires peers to determine the price of a digital service, which can be hard to estimate. Third, remuneration can result in new forms of unfairness where a few affluent peers exclusively enjoy the services of a decentralized application. This situation could, for example, arise when operating peer-to-peer auctions for the allocation of services.

2.2.2 Trust-based Incentives

Applications implementing trust-based incentives indirectly reward community members for their work. For example, the system can reward dedicated peers with preferential treatment or provide them access to exclusive services. This approach often requires peers to keep track of the long-term contributions of other peers using accounting infrastructure [86]. The specifications of accounted work can then be used by the application to detect how a particular peer has contributed to the system. For instance, the accounted work can be used by a reputation algorithm that outputs a ranking of peers [87]. If the ranking of a specific peer is below a threshold, the application can decide to refuse to perform work for this peer until its ranking has improved. We outline related work that uses

work accounting and trust-based incentives. For an overview of (decentralized) reputation mechanisms and trust models, we refer the interested reader to existing work [88, 89].

Perhaps the most popular decentralized application is BitTorrent, a peer-to-peer file exchange protocol [90]. In BitTorrent, each peer has a limited number of slots to allocate to other peers. The system uses tit-for-tat, a cooperation strategy where a counterparty loses its slot when it stops to reciprocate. This simple strategy leads to higher network utilization since long-term free-riders will not be allocated slots. BitTorrent does not persist all contributions and consumptions of other peers, but tracks the performance of connected peers for each download.

The InterPlanetary File System (IPFS) is a decentralized system for file storage and exchange [39]. IPFS breaks up files into blocks, which are identifiable by a content identifier. The original IPFS whitepaper describes BitSwap, a set of tools to exchange blocks while addressing free-riding behaviour through block bartering. It ensures that peers are incentivized to seed blocks by pair-wise tracking of outstanding "balances". Peers that do not sufficiently share blocks will be ignored by others.

Wallach et al. present different mechanisms for the fair sharing of resources in decentralized applications [91]. These mechanisms ensure that each peer maintains a log with actions and includes random auditing of logs. The applicability of their work is exclusive to storage-based application and is not reusable for other decentralized applications. Osipkov et al. describe an accounting mechanism for file-sharing applications [92]. Specifically, each peer maintains a set of witnesses that monitors all transactions of that peer.

LiFTinG and AcTinG are protocols for tracking free-riding behaviour in gossip-based applications [93, 94]. The LiFTinG protocol exploits the message dynamics between peers and verifies that the content received by a peer is further propagated according to the protocol. The design depends on a statistical approach and cross-checking of logs to detect free-riders but is not reusable for applications beyond gossip. AcTinG is a gossip-based dissemination protocol that is resistant against colluding rational peers.

Other approaches maintain a distributed ledger that store information in decentralized applications. Seuken and Parkes introduce a Sybil-resistant accounting mechanism based on transitive trust [95]. PeerReview is an accountability mechanism to record message exchange between peers [96]. Peers store all network messages in a local log. Dedicated witnesses continuously audit peers and detect whether a peer has deviated from the protocol. The FullReview protocol extends PeerReview by addressing selfish behaviour with a game-theoretical model [97]. Otte et al. present TrustChain, a Sybil-resistant reputation mechanism with an accompanying accounting mechanism [98]. The authors apply their mechanism to address free-riding behaviour in a file-sharing network. We find that peers in TrustChain cannot engage in the recording of multiple interactions simultaneously, significantly limiting the achievable throughput. Crosby et al. present a data structure for tamper-evident logging [99]. This data structure orients around the efficient logging of unilateral system events on a server. Peermint is an accounting mechanism designed for market-based management of decentralized applications [100].

2.2.3 Problem Description

There currently is no *universal* accounting mechanism that can be used to address fairness issues in decentralized applications, to the best of our knowledge. We address this

shortcoming and describe three challenges when designing such a mechanism.

Challenge I: Universality. The trust-based solutions that we have identified so far are designed for usage within a single application domain and are infeasible to re-use. We believe that universality is an important property to address fairness concerns in novel decentralized applications.

Challenge II: Full Decentralization without Central Authority. To keep our system reusable and universal, we avoid *any* decision-making by entities with leveraged authorities and central servers. The lack of a central authority makes our mechanism *fully decentralized* and easier to deploy. In general, decentralized mechanisms are less vulnerable to large-scale attacks, tend to scale better, and are more resilient to failure. They also are an excellent architectural fit with existing decentralized applications that avoid central authorities.

Challenge III: Fraud Detection. Peers have a natural incentive to misrepresent the magnitude of their efforts to inflate their social standing or to hide information unfavourable to their standing [86]. Our accounting mechanism must address the completeness and correctness of the stored information. We must *detect the manipulation or hiding of accounted information* and punish adversarial peers accordingly.

2.3 Accounting Work with ConTrib

The design of our universal accounting mechanism, named *ConTrib*, is inspired by the tamper-evident properties of blockchain but does not require peers to reach consensus on a coherent history of records. Instead, ConTrib optimistically detects the illegitimate modification of records while keeping the computational overhead and bandwidth requirements low. Decentralized applications can account the work performed by peers within tamper-evident *records*. A record describes some work performed by one peer for another peer. Each peer organizes its records in a *personal ledger*. Records point to prior records in the same personal ledger and also point to records in the personal ledger of others. The latter pointer captures an agreement between two peers. Peers continuously exchange records with other random peers and request records in the personal ledgers of others. By validating the consistency of incoming records against known ones, a peer can irrefutably prove fraud attempts to other peers.

We further elaborate on the design of ConTrib. We first outline the network and threat model. We then describe the ConTrib data structure and show how ConTrib accounts the work in decentralized applications.

2.3.1 Network Model

The ConTrib mechanism is built on a peer-to-peer network. We assume an unstructured network structure. Unstructured networks are relatively straightforward to maintain and are highly resilient against churn. We assume that the used networking library handles network bootstrapping and peer discovery. We also assume that the communication channels between peers are unreliable and unordered (e.g., by using the UDP communication model). The arrival time of messages is not upper-bounded, and outbound messages can fail to arrive at their intended destination. Each peer has a cryptographic key pair, consisting of a public and private key. The public key acts as a unique identifier of the peer in

the network, whereas the private key is used to sign records and outgoing network messages. We consider attacks targeted at the network layer, e.g., the Eclipse Attack, outside the scope of this work.

A significant threat in Internet-deployed applications is the Sybil Attack, where an adversary operates multiple identities to subvert the network [101]. The Sybil Attack frequently occurs in open Internet communities where the cost of creating a new digital identity is often negligible. Although the ConTrib mechanism does not include defences against Sybil identities, we argue that this threat can be mitigated with well-established techniques that can complement ConTrib in a deployment setting. A basic defence mechanism is to have peers solve a computational puzzle when they wish to join the network [102]. In addition, using a Sybil-resistant reputation mechanism that processes ConTrib records can effectively mitigate the effect of Sybil identities on computed trust scores [103, 104]. We also consider self-sovereign identities as a promising solution that can bolster decentralized networks with long-term identities [105].

We leave defences against misreporting, the accounting of work that has not actually occurred in the application, to other layers in the application stack. This attack is closely related to the Sybil Attack since Sybil identities are likely to create fake interactions amongst them [106]. Misreporting is challenging to address in a generic manner since there is not always a straightforward method to assess if some accounted work is legitimate. Some protocols use cryptographic techniques to prove the accuracy of performed work, for example, Proof-of-Storage and Proof-of-Bandwidth [41, 84]. These techniques, however, are not generic and cannot easily be used within many application domains.

2.3.2 Threat Model

Our threat model orients around malicious peers that attack the integrity of the ConTrib data structure. This attack proceeds through the strategic modification of ConTrib records. For example, a peer can inflate the amount of work it has performed by modifying one of the records in its personal ledger. We refer to the illegitimate modification of ConTrib records as *fraud*. Even though this definition may seem limited, we argue that this kind of fraud is a fundamental threat to the ConTrib data structure. In particular, our definition of fraud also entails a more advanced form of record manipulations where peers collude to erase a particular interaction from history.[1] In a reputation system, for example, this would happen when a well-trusted peer temporarily boosts the reputation of another peer by accounting some work and then attempts to hide the existence of this interaction later. Reverting this interaction requires both counterparties to either override or remove the associated records, which we consider as fraud. We note that a particular fraud instance in our system involves at most two guilty peers. As we discussed in Section 2.2.3, we require that fraud is detected. We assume that the computing power of adversaries is bounded and that cryptographic primitives are secure.

[1]Peers might refrain from overriding or erasing their records during or after a collusion attempt. We do not consider this as fraud since adversaries do not exploit the ConTrib data structure. Since all records associated with the collusion attempt are accounted, decentralized applications might employ additional logic to analyse created records and attempt to detect possible collusion attempts, e.g., with correlation analysis [107].

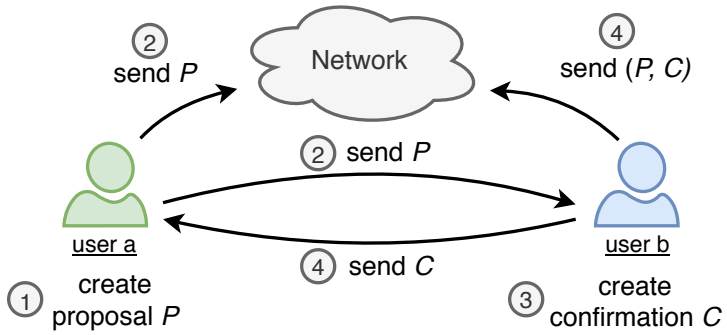


Figure 2.3: The process of recording work between peers a and b within two records: a proposal P and a confirmation C.

2.3.3 Recording Interactions

Some work that involves peers a and b is recorded using two records: one *proposal* created by a and one *confirmation* created by b. W.l.o.g., we assume that the accounted work is performed by peer a for peer b. The process of accounting this work is visualized in Figure 2.3. First, a creates a proposal record, which we refer to as P (step ①). Proposal P, created by peer a, is a tuple with the following four attributes:

$$P = (\texttt{pubKey}, \texttt{pubKeyOther}, \texttt{payload}, \texttt{sig})$$

Proposal P contains the public key of peers a and b (`pubKey` and `pubKeyOther`, respectively), an application-specific payload (`payload`), and a digital signature (`sig`) created by a of the record in binary form. The `payload` attribute is an arbitrary blob of data and is provided by the connected application. The payload could include an identifier that uniquely identifies the performed work. To increase the resilience against manipulation, we extend records with additional fields in the next section. After peer a has included all described attributes in the proposal, it persists the record to its database, sends the proposal to b, and disseminates the proposal to f random peers in the network (step ②). We refer to f as the *fanout* value.

When peer b receives the proposal P, b verifies its validity. It is during this step that fraud is detected. The validation logic of incoming records is elaborately discussed in Section 2.4. If the incoming proposal P is deemed valid, the connected application determines if the payload in P truthfully describes the performed work. If P is considered invalid, b ignores the incoming proposal and takes no further action. Otherwise, b creates a confirming record that confirms P (step ③). This confirmation, denoted by C, contains the same attributes as the proposal P and also includes the hash of P. Confirmation C, created by peer b, is a tuple with the following five attributes:

$$C = (\texttt{pubKey}, \texttt{pubKeyOther}, \texttt{payload}, \texttt{proposalHash}, \texttt{sig})$$

The value of `proposalHash` is computed by $H(P)$, where $H(\cdot)$ is a secure hash function. We call the `proposalHash` attribute in C the *confirmation pointer*. After the creation of C, peer b persists the confirmation to its database, sends it to peer a, and disseminates both

P and C to f random peers (step ④). Upon the reception of C, peer a validates C and persists the confirmation if it is valid. Both parties are now in possession of proposal P and confirmation C that together prove an agreement on work between these parties. The process of accounting work is lightweight since it requires minimal computational steps and data exchange. Additionally, peers can engage in the recording of multiple interactions simultaneously.

A potential risk is that b refuses to confirm P, even though the incoming proposal is valid and contains the correct work details. This could, for example, occur when confirming P negatively impacts b's social standing. This leaves a with an unconfirmed proposal, which alone is not sufficient evidence to convince other peers of the performed work by a for b. When b refuses to sign an incoming proposal, a will add b to the local blacklist managed by applications, refusing to perform work for b until b has confirmed P. The losses for a depend on the magnitude of the (unconfirmed) work performed for b. To minimize these losses, we suggest that decentralized applications record small units of works using ConTrib. For example, a file-sharing application can choose to account unconfirmed work when it reaches a threshold, e.g., 10 MB of traffic exchanged. Depending on the granularity of accounting, this approach can significantly reduce the impact of peers refusing to acknowledge the contributions of their counterparties.

2.3.4 Improving Resilience by Linking Records

To prevent the modification of created records, we enforce each peer in ConTrib to link their records together in a *personal ledger*, incrementally ordered by creation time. Linking records will also make it harder for malicious peers to hide specific records. We make the following four modifications to records:

1. First, we include a sequence number $s \in \mathbb{Z}$ in each record that is incremented by one when a record is added to one's personal ledger. The sequence number of the first record in the personal ledger is 1.

2. Second, each record now includes the hash of the prior record in the personal ledger of the creator. This modification makes the ConTrib data structure comparable to a hash chain, e.g., as used by blockchain applications. The modification of a particular record now changes the hash of subsequent records, a feature that enables us to detect illegitimate changes to stored records (also see Section 2.4). The previous hash of the first record in a personal ledger is empty and referred to as $\perp$.

3. Third, we extend the confirmation pointer with the sequence number of the proposal record that it confirms.

4. Forth, we include at most b additional hashes in each record of distinct, prior records in the same personal ledger. We refer to the set with these hashes as S and call these hashes *back-pointers*. As we will further show in Section 2.4, the inclusion of these back-pointers significantly speeds up the detection of fraud. The required back-pointers in some record R are deterministically given by a pseudo-random function σ that takes the public key of the record creator and the sequence number of R as input. σ returns a set with at most b prior records which hashes should be included

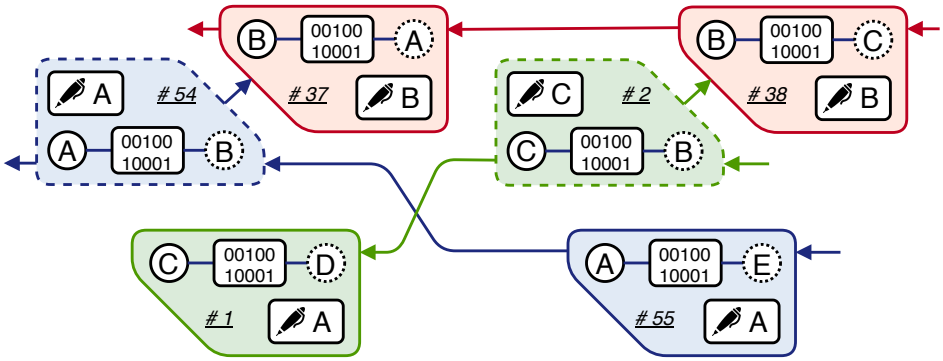


Figure 2.4: A part of the ConTrib DAG, involving five peers and six records: four proposals (solid borders) and two confirmations (dashed borders). The proposal created by user a with sequence number 55 is unconfirmed.

in R. All peers must use the same version of σ, which we achieve by bundling its implementation in the ConTrib software.

The above modifications change the attributes of proposal and confirmation records. We re-define a proposal P, created by peer a and with counterparty b, as follows:

$$P = (\texttt{pubKey}, \texttt{pubKeyOther}, \texttt{payload}, \texttt{sig}, \texttt{seqNum}, \texttt{prevHash}, \texttt{backPointers})$$

The variables coloured green are new compared to our previous definition of P. `seqNum` refers to the sequence number of P, `prevHash` indicates the hash of the previous record, and `backPointers` is the set with back-pointers (where $|S| \leq b$). We re-define a confirmation C, created by peer b, as follows:

$$C = (\texttt{pubKey}, \texttt{pubKeyOther}, \texttt{payload}, \texttt{linkInfo}, \texttt{sig}, \texttt{seqNum}, \texttt{prevHash}, \texttt{backPointers})$$

We extend confirmations with the same attributes as a proposal but replace the `proposalHash` attribute with `linkInfo`. This is in accordance with our third modification. `linkInfo` is now defined as a tuple with the hash and sequence number of the referred proposal record:

$$C.\texttt{linkInfo} = (\texttt{hash}, \texttt{seqNum})$$

Creating records yields the graph structure shown in Figure 2.4. Figure 2.4 shows a part of the ConTrib graph with six records, created by three distinct peers (a, b and c). Same-coloured records are part of a single personal ledger, and arrows represent hash pointers to other records. Proposals have a solid border whereas confirmations have a dashed border. Note how in a's personal ledger the record with sequence number 55 is unconfirmed. For presentation clarity, we only show the pointer to the prior record in one's personal ledger and omit additional back-pointers from the figure.

ConTrib publicly accounts work in interlinked personal ledgers. Since all performed work is publicly stored and accessible, other users might acquire and analyse ConTrib records to reveal potentially sensitive information, e.g., the time at which a particular user

is online or the interaction patterns between users. To reduce this threat, we outline two techniques that applications can use to enhance privacy. First, applications can account performed and consumed work in *batches*. For example, an application can record all outstanding contributions and consumptions every hour, therefore hiding granular work statistics. Second, an application can add some noise to the amount of work being accounted. This technique effectively reduces linkability, e.g., when accounting traffic that is being relayed through multiple hops. We believe that the combined power of these two techniques already provides sufficient privacy guarantees for most decentralized applications. A more advanced approach, used by the Monero cryptocurrency, leverages ring signatures and zero-knowledge proofs to hide the amounts of work performed [108, 109]. This approach would require fundamental changes to ConTrib, and we therefore leave this enhancement for further work.

2.4 Detecting Fraud

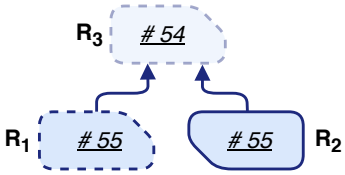
We require that ConTrib detects illegitimate tampering of the records in a personal ledger. ConTrib is built around fraud *detection* instead of *prevention*. We argue this is a reasonable assumption for two reasons. First, decentralized applications often do not require the prevention of fraud [110]. We argue that fraud prevention is disproportional in the context of work accounting since this work usually holds no or low monetary value. Second, fraud prevention is often a resource-intensive process that requires peers to reach a consensus on all created records, e.g., by using classical BFT algorithms or Proof-of-Work [111]. The requirement to reach consensus would dramatically reduce the scalability of ConTrib.

Fraud in ConTrib occurs when a peer illegitimately modifies one of the records in their personal ledger. This fraud, for example, happens when an adversary attempts to hide a specific record in the personal ledger by replacing it with another one. This modification would result in pairs of records with the same sequence number and the same creator, but with a different hash, and violates the integrity of the ConTrib data structure. A key objective of ConTrib is to detect such conflicting records quickly.

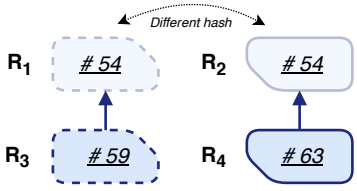
2.4.1 Detecting Forks

Fraud in ConTrib is detected by sharing newly created records with other peers, and by requesting random records in the personal ledgers of others. Each peer assesses the consistency of incoming records with the ones in its local database. This simple approach allows for quick detection of fraud through the collective effort of peers. In Figure 2.5 we visualize four identified scenarios in which we can either expose an adversarial peer (scenario I and II) or detect an inconsistency without assigning blame (scenario III and IV). Each scenario shows the situation from a single peer's perspective and highlights records that a peer has in its local database, or does not have. Records not in the possession by a peer are faded. Records with the same colour are created by the same peer. We discuss each scenario and elaborate on how they either lead to fraud exposure or the detection of an inconsistency.

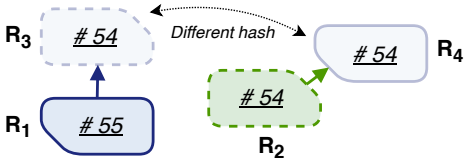
- **Scenario I.** The first scenario, visualized in Figure 2.5a, describes a situation where a peer can directly expose a fork in the personal ledger of peer a. The personal ledger of peer a has been forked since records R_1 and R_2 have the same sequence number



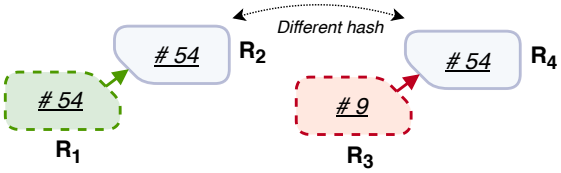
(a) *Scenario I*: Records R_1 and R_2 have the same sequence number but a different hash. The pair (R_1, R_2) is irrefutable proof that peer a has forked its personal ledger.



(b) *Scenario II*: Records R_3 and R_4 both contain a back-pointer to the record with sequence number 54 but their hashes differ. The pair (R_3, R_4) is irrefutable proof that peer a has forked its personal ledger.



(c) *Scenario III*: Records R_1 and R_2 contain a differing hash of a's record with sequence number 54. This reveals an inconsistency.



(d) *Scenario IV*: Records R_1 and R_3 both confirm a's record with sequence number 54, but they contain a different hash. This reveals an inconsistency.

Figure 2.5: Four scenarios that allows a peer to either expose fraud (forking of a personal ledger), or to detect an inconsistency (without assigning blame). The colour of each record indicates the identity of its creator (blue for a, green for b and red for c). Solid and dashed records indicate proposals, respectively confirmations. Opaque records are not in possession by the peer.

but a different hash. As soon as another peer, say b, receives R_1 while already having R_2, or receives R_2 while already having R_1, the pair (R_1, R_2) is sufficient evidence to expose the fraud by a. The digital signature by a in the records prove that a deliberately created both records. Note that b does not need to have R_3 to detect nor to prove this fraud. We call the pair (R_1, R_2) a *fraud proof*. Fraud proofs are by default shared with other peers in the network through a `FraudProof` message.

- **Scenario II.** The second scenario describes the situation where one can prove fraud by detecting inconsistencies in the included back-pointers of records. Figure 2.5b shows four records created by peer a. Records R_3 and R_4 contain the hash of the record with sequence number 54 in the back-pointer set; however, these back-pointers

describe the same record with a different hash. The pair (R_3, R_4) is irrefutable proof that peer a has committed fraud and this pair can be used to construct a fraud proof.

- **Scenario III.** Figure 2.5c shows the third scenario where a peer receives proposal R_1 and already has confirmation R_2, or receives confirmation R_2 while already having proposal R_1. The peer does not have records R_3 and R_4. The hash of record R_3 in R_1 differs from the hash in the confirmation pointer in R_2. This situation reveals an inconsistency that is either introduced by peer a forking its personal ledger at height 54, or by b having included a wrong hash in R_2. To assign blame, the peer that is validating the incoming record requires either R_3 or R_4. A peer that encounters this situation sends the pair (R_1, R_2) within an `Inconsistency` message to other random peers, hoping that others will be able to expose the malicious peer.

- **Scenario IV.** Figure 2.5d highlights the fourth scenario where a peer either receives confirmation R_1 while already having confirmation R_3, or vice versa. Both confirmations point to a record with the same public key and sequence number, but the hash of this record differs. This situation either indicates a fork of the personal ledger of a, or it can be the result of an invalid pointer in one of the confirmations. Similar to scenario III, the validating peer sends the pair (R_1, R_3) within an `Inconsistency` message to other, random peers.

2.4.2 Record Validation Logic

Based on the four identified scenarios, we design and describe the validation logic of an incoming record R. Each peer keeps track of known hashes in a dictionary named `knownHashes`. This dictionary is indexed with a tuple, containing the public key and sequence number of a record. The value of dictionary entries is the hash of the record being queried. The validation logic of incoming records consists of the following five steps:

Step 1. We first verify the validity of the fields in incoming record R. This step is performed by the validateFields procedure, which returns a boolean value indicating whether the fields in the record are valid or not. A pseudocode description of this procedure is given in Algorithm 1. This step validates the sequence number (line 3), the included public keys (line 9 and 15), and the digital signature (line 12). If the incoming record is a confirmation, it also verifies that the sequence number in the `linkInfo` attribute is within a valid range (line 6). It also checks whether the hash of the prior record is sane when the record is the first in ones personal ledger (line 18). We remark that this step does not compare the validity of R in the light of other records. Any error in the included fields of R is computationally efficient to detect and likely originates from a software bug.

Step 2. Next, we query the database for a record with the same public key and sequence number as the incoming record R. If such a record R' is in the database, we check the equality of R and R' by performing a comparison between their included fields. If $R \neq R'$, we have detected a fork in the personal ledger of the creator behind R. We then share the fraud proof (R, R') with other peers in the network. During this step, we detect the fraud described by scenario I in Section 2.4.1.

Step 3. Then, we verify if the hash pointers in the incoming record are consistent with known ones. This is performed by the validateHashes procedure which pseudocode

Algorithm 1 The validation of the fields in record R.

```
1:  procedure validateFields(R)                                   ▷ Step 1
2:      valid ← true
3:      if R.seqNum < 1 then
4:          valid ← false
5:      end if
6:      if isConfirmation(R) and R.linkInfo.seqNum < 1 then
7:          valid ← false
8:      end if
9:      if not publicKeyIsValid(R.pubKey) then
10:         valid ← false
11:     end if
12:     if not signatureIsValid(R.pubKey, R.sig) then
13:         valid ← false
14:     end if
15:     if not publicKeyIsValid(R.pubKeyOther) then
16:         valid ← false
17:     end if
18:     if R.seqNum = 1 and R.prevHash ≠ ⊥ then
19:         valid ← false
20:     end if
21:     return valid
22: end procedure
```

description is given in Algorithm 2. This procedure first checks whether the `prevHash` attribute in R is consistent with the information in the `knownHashes` dictionary (line 3). We then iterate over all included back-pointers and verify the consistency of these hashes with the entries in the `knownHashes` dictionary (line 7-12). During this step, we detect the fraud described by scenario II.

Step 4. Next, we compare incoming record R with a link record, if such a record is available in the database. When R is a proposal, we get the corresponding confirmation from the database, and if R is a confirmation, we get the corresponding proposal. This step is performed by the validateLink procedure which pseudocode description is given in Algorithm 3. We first get the linked record from the database (line 2) and only continue with this validation step if we have this record in the database. If so, we check whether the public keys included in the proposal and confirmation are consistent (line 10), and verify the consistency of the `linkInfo` attributes in the confirmation (line 13-18). We detect the inconsistency described by scenario III (line 16-18) and scenario IV (line 19-22) during this step.

Step 5. Finally, we verify the validity of the included payload, which is an application-dependent validation procedure. As we will further outline in Section 2.5, decentralized applications using ConTrib should implement a *validation* policy that denotes whether the payload of an incoming record is valid in the context of the connected application.

If any of the above steps fail, the record is considered invalid and not further processed.

Algorithm 2 The consistency validation of hashes in an incoming record against known ones.

```
1:  procedure validateHashes(R)                                  ▷ Step 4
2:      hash ← knownHashes[(R.pubKey, R.seqNum − 1)]
3:      if hash ≠ ⊥ and hash ≠ R.prevHash then
4:          return false
5:      end if
6:
7:      for seqNum, hash in R.backPointers do
8:          known ← knownHashes[(R.pubKey, seqNum)]
9:          if known ≠ ⊥ and hash ≠ known then
10:             return false
11:         end if
12:     end for
13:     return true
14: end procedure
```

2.4.3 Exchanging Records with Other Peers

The detection of fraud in ConTrib depends on peers exchanging records with each other. A peer is motivated to share collected records with others since they might eventually reveal fraud conducted by one of their former counterparties. So far, we have not discussed how records are disseminated. Record dissemination is an essential process that affects the speed at which fraud can be detected. For example, a slow record exchange strategy is likely to increase fraud detection times compared to more aggressive record dissemination. We consider both *push-based* and *pull-based* exchange of records, which is explained next.

Pull-based Record Exchange. Each peer by default requests (pulls) records from other random peers at a fixed rate by sending out `Request` messages. Applications can choose to send out `Request` messages to specific peers to build profile information about that peer, e.g., to detect free-riders. A `Request` message contains a list of sequence numbers and the recipient is expected to send back the records with these sequence numbers in their personal ledger. When responding to a `Request` message, the recipient also includes linked proposal or confirmation records in the response. When a peer a does not respond with records within a reasonable time, the requesting peer adds a to their local blacklist.

Push-based Record Exchange. ConTrib also supports push-based record exchange in which case the creator of a record disseminates it to f random other peers (as also discussed in Section 2.3.3). This push-based exchange allows for quick detection of fraud since the probability of no user receiving two conflicting records goes to zero quickly, even when the network size increases [112]. Even if the malicious peer refrains from broadcasting a conflicting record, the counterparty is very likely to do so, assuming there is no collusion between interacting peers. Immediate dissemination of created records in the network also increases record availability when the sending peer goes offline.

We envision that ConTrib is used by multiple applications simultaneously. It is likely that particular applications require different dissemination rates that deviate from a system-wide strategy. Even though our experiments consider a single application that leverages

Algorithm 3 The validation of an incoming record against a linked record.

```
1:  procedure validateLink(R)                                   ▷ Step 3
2:      linked ← db.getLinked(R)
3:      if linked = ⊥ then
4:          return true
5:      end if
6:
7:      proposal ← linked if isConfirmation(R) else R
8:      confirmation ← linked if isConfirmation(linked) else R
9:
10:     if confirmation.pubKeyOther ≠ proposal.pubKey then
11:         return false
12:     end if
13:     if confirmation.linkInfo.seqNum ≠ proposal.seqNum then
14:         return false
15:     end if
16:     if confirmation.linkInfo.hash ≠ proposal.hash then
17:         return false
18:     end if
19:     linkLinked ← db.getLinked(linked)
20:     if linkLinked ≠ ⊥ or link_linked ≠ R then
21:         return false
22:     end if
23:     return true
24: end procedure
```

ConTrib, we briefly discuss how we can optimize ConTrib to serve different applications. One approach is to build *virtual ledgers* where the personal ledger of each peer consists of multiple sub-ledgers. This can be achieved by adding additional hash pointers between the records associated with a particular application. These records can then be disseminated with custom dissemination strategies to other peers that participate in a particular application. A single application can also create multiple virtual ledgers, e.g., when the application requires accounting of distinct record types with differing creation rates. This would, however, require additional membership logic where we track which peers are involved in what application. Note that exchanging application-specific records separately strengthens the integrity of ConTrib in general since these records also bear pointers to records associated with other applications.

2.4.4 Limitations

Even though our simple algorithm can detect the modification of records, the probabilistic nature of our algorithm can render ConTrib unsuitable for deployment in specific application domains. Since fraud detection is a probabilistic approach, some fraud instances can take relatively long to be uncovered (e.g., several minutes). We also encountered this behaviour during our experiments (see Section 2.6). At the same time, we argue that this

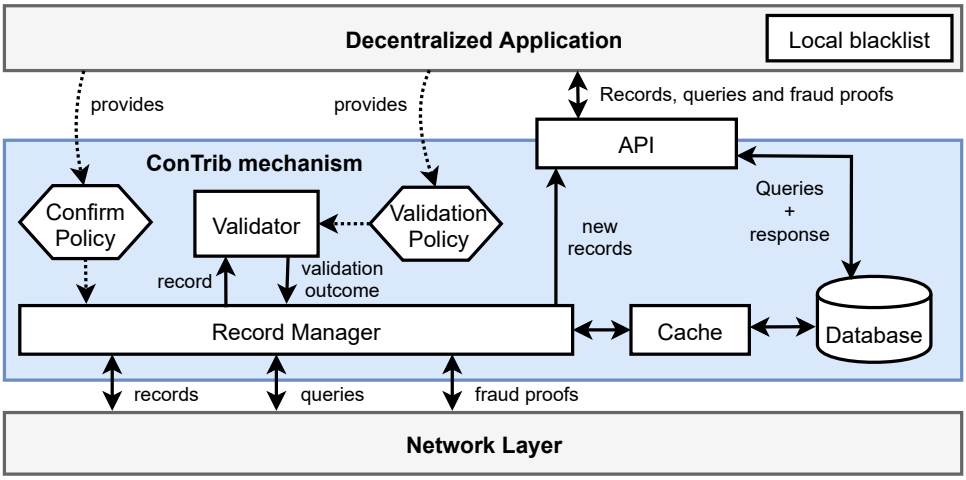


Figure 2.6: The system architecture of ConTrib.

is not an insurmountable problem in decentralized applications where performed work holds no or low monetary value.

Since our algorithm is based on fraud detection, ConTrib is not suitable for applications that require a high level of security, as is the case with decentralized financial applications. We believe that blockchain technology provides more appropriate security guarantees for such application domains, at the cost of increased resource usage and lower scalability.

Finally, we note that ConTrib is mainly built for the lightweight accounting of work in decentralized applications. In its current state, ConTrib cannot capture more complicated operations, e.g., executing arbitrary logic like smart contracts. However, as demonstrated by recent research, a lightweight accounting mechanism can be an enabling component to devise novel and scalable types of decentralized applications with dynamic risk guarantees [113–115].

2.5 System Architecture

We devise a system architecture of our ConTrib mechanism, see Figure 2.6. The network layer is the lowest layer in our architecture and provides the primitives for decentralized communication and peer discovery. This layer can be realized using existing frameworks to build peer-to-peer overlay networks, for example, libp2p.[2]

Record Manager. The record manager interacts with the network layer to disseminate records and processes incoming ones. It queues incoming records for validation and persists incoming fraud proofs to the database and connected application. It also manages the confirmation of incoming and valid proposals targeted at that peer. Applications using ConTrib should provide a *confirmation policy* that predicates whether an incoming proposal should be confirmed.

[2]See `https://libp2p.io`

Validator. The validator determines the validity of incoming records according to the algorithms described in Section 2.4.2. Connected applications can provide a custom *validation policy*. If provided, this validation policy is invoked during step 5, when the application-specific payload in a record is validated. The flexibility to provide custom validation and confirmation policies for incoming records makes ConTrib universal and reusable across different application domains.

Persistence. Records and fraud proofs are persisted in a database. The ConTrib system architecture provides an interface for the queries made to the database and supports different database architectures. Our system architecture includes a record *cache*, which is an intermediary component that stores all records in the personal ledger of the operating peer in memory. This cache allows ConTrib to quickly respond to incoming record queries in the personal ledger of the operating peer. This cache forwards queries to the database for the retrieval and storage of records and fraud proofs.

To contain the growth of the database and to keep the storage overhead manageable, an application can choose to periodically prune the ConTrib database when a storage threshold is reached. In our implementation, by default, we start pruning when at least one million records have been stored. Applications may increase or decrease this number, depending on the storage capacities of participating peers and the deployment environment. The default pruning strategy of ConTrib continuously removes the record with the lowest database insertion timestamps until the database size has reached its storage threshold again. The pruning of older records might cause some forks to go undetected since records could be removed before a fraud proof can be constructed. As we will show in Section 2.6.2, most forks in ConTrib are quickly detected, and there should be ample time to detect inconsistencies before relevant records are pruned.

Fraud Management. When the validation algorithm exposes fraud, or when ConTrib receives an incoming fraud proof, the connected applications are notified of the fraud and can punish the misbehaving peer accordingly. For example, a fraud policy in a bandwidth sharing application could decide to not serve the fraudster for some time. The connected applications store the digital identities of fraudsters in a local blacklist.

Interactions between ConTrib and Applications. Decentralized applications interact with ConTrib through an API. This API allows connected applications to query the content of the database. Furthermore, connected applications can subscribe to incoming records. The record manager forwards new records to the API, which passes these records to subscribed applications.

Parameter	Default Value
Peers (n)	1'000
Workload	1 proposal per second per peer
Record exchange strategy	PULL+RAND+PUSH
Record fanout (f)	5
Record request batch size	2
Record request interval	0.5 seconds
Packet Loss Rate	0%
Individual forking probability	10%
Back-pointers (b)	10

Table 2.1: The default parameters used during our evaluation.

2.6 Implementation and Evaluation

In this section we systematically explore how ConTrib behaves when modifying system parameters. We implement ConTrib in the Python 3 programming language. We leverage the network library implemented by our research group, and use the UDP protocol for network communication between peers [63]. Our implementation uses the `asyncio` framework for asynchronous event handling. This implementation features both an in-memory storage and a persistent (sqlite) database which can be used to persist records over different sessions. The full implementation of ConTrib, including unit tests and documentation, is published on GitHub.[3]

2.6.1 Experiment Setup

We evaluate the impact of different parameters on the efficiency of fraud detection. We do so by measuring the time between committing fraud and its initial detection. We substitute our networking layer with the SimPy discrete event simulator [116]. Each peer in the ConTrib network knows the network address of 100 random other peers, resulting in an unstructured overlay topology. This topology remains fixed during our experiment. Table 2.1 lists the default parameters used during our evaluation. To encourage reproducibility, we have open-sourced the ConTrib simulator and all experiment scripts.[4]

Workload and Attack Model. During our experiments, peers create records with other random peers. Our default workload has each peer initiate one proposal per second with another random peer. Note that the rate at which new records are created grows with the network size, which should capture the dynamics of real-world applications (when there are more peers, there is usually more work performed in the application). We use a uniform transaction load to analyse the characteristics of ConTrib under a predictable load. Even though this transaction load is predictable, it resembles an application where work is periodically accounted. We experiment with network sizes ranging from 1'000 to 10'000 online peers. Though some deployed networks have many more peers (e.g., BitTorrent and Tor), our experimental results suggest that ConTrib has no issues scaling beyond 10'000

[3]See `https://github.com/tribler/py-ipv8/tree/master/ipv8/attestation/trustchain`
[4]See `https://github.com/tribler/trustchain-simulator-pysim`

peers. In Section 2.6.6, we subject ConTrib to a realistic workload, extracted from the interactions in a decentralized file-sharing application.

Each peer forks its personal ledger with a probability of 10% by removing the last record in its personal ledger and re-using its sequence number to create a record. Each peer commits this fraud once. A peer committing fraud will not broadcast the duplicate record when push-based record exchange is enabled. In each experiment run, all peers start with an empty personal ledger, and interaction partners always confirm incoming proposals. A peer that has exposed the fraud of another peer will refuse to confirm the proposals by that peer. Each experiment run terminates either when all fraud attempts have been discovered or after ten minutes have elapsed. We are interested in the detection time of fraud instances, which is the time period between committing the fraud and its first detection by a peer in the network.

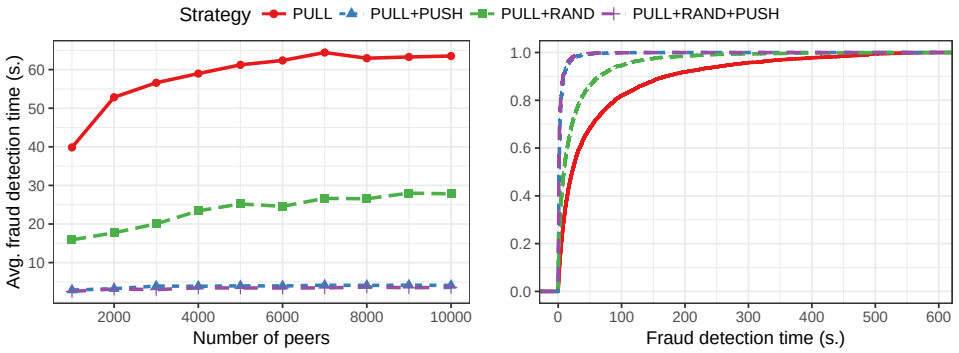
Record Exchange Strategies. We consider the following four strategies for exchanging records. With the PULL strategy, each peer requests two contiguous records at a random height in the personal ledger of another random peer every half a second (the *record request batch size* and *record request interval* parameters in Table 2.1). Under the PULL+RAND strategy, a peer also returns five random records in their database upon receiving a query. Including random records in query responses enables the detection of fraud of offline peers. The PULL+PUSH strategy also pushes new records to f random users upon creation, in addition to the PULL strategy. Finally, we consider the PULL+RAND+PUSH record exchange strategy, which is a combination of the above techniques.

2.6.2 Scalability

Our first experiment quantifies the scalability of ConTrib when increasing the number of peers in the network, see Figure 2.7. Figure 2.7a shows the effect of increasing the number of peers on the average time until fraud detection, for the four discussed record exchange strategies. For all record exchange strategies, the average fraud detection time seems to increase when the network size grows. For $n = 10'000$, the PULL strategy shows an average fraud detection time of 63.5 seconds, whereas this number decreases to 3.6 seconds under the PULL+RAND+PUSH strategy. We notice that the PULL+PUSH and PULL+RAND+PUSH strategies show detection times under five seconds on average. These low detection times demonstrates that disseminating a record just after its creation is a highly successful strategy. Including random records in crawl responses also seems to decrease the average fraud detection times. For $n = 10'000$, the average fraud detection time decreases from 63.5 seconds for the PULL strategy to 27.8 seconds for the PULL+PUSH strategy.

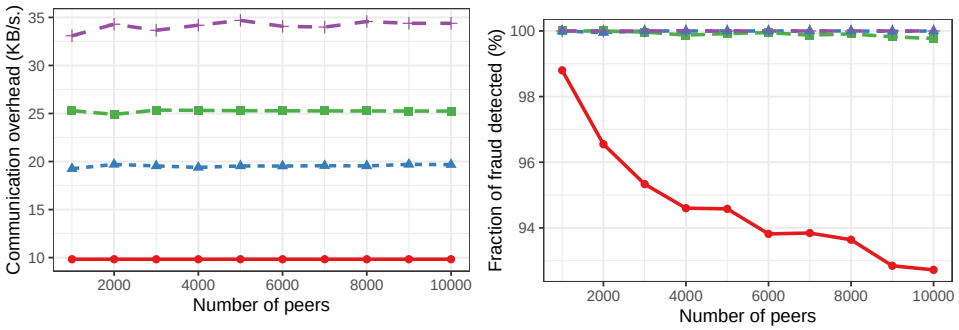
In Figure 2.7b, we show the Empirical Cumulative Distribution Function (ECDF) of fraud detection times for $n = 5'000$. We observe that it can take several minutes for some fraud attempts to be discovered, in particular for the PULL and PULL+RAND strategies. This is not unexpected since fraud detection in ConTrib is a highly probabilistic approach. For the PULL strategy, we can detect 90% of fraud attempts within 160 seconds and 50% of the fraud attempts within 30 seconds. We also observe that the vast majority of fraud is detected within a few seconds when pushing random records after creation. 82.9% of all fraud attempts are detected within five seconds for the PULL+RAND+PUSH strategy.

Figure 2.7c shows the average network usage per peer as the network size increases, for different record exchange strategies. The PULL strategy requires less than 10 KB per



(a) Average fraud detection times as the number of peers increases.

(b) The Empirical Cumulative Distribution Function (ECDF) for 5'000 peers.



(c) Average network usage per peer as the number of peers increases.

(d) Fraction of fraud exposed after our experiment ends.

Figure 2.7: The results of our scalability experiments, with up to 10'000 peers. We evaluate four record exchange strategies.

second of network usage. Compared to the PULL strategy, the PULL+RAND+PUSH strategy requires more than three times as much bandwidth, around 35 KB per second. Note how the network usage remains roughly the same for all considered record exchange strategies as we add more peers to the experiment. Figure 2.7c also shows that it is feasible to deploy ConTrib in consumer-grade network environments. System designers can decrease the network usage of ConTrib even more by lowering the crawl interval or crawl batch size, at the cost of increased fraud detection times.

Not all fraud has been detected after our experiment has ended. Figure 2.7d shows the percentage of fraud attempts that has been detected after the experiment has ended, for an increasing number of peers and different record exchange strategies. It becomes less likely that fraud is detected during our experiment when increasing the network size under the PULL strategy. For $n = 10'000$, 7.28% of fraud attempts remain undetected. These attempts would likely be discovered when prolonging the experiment duration.

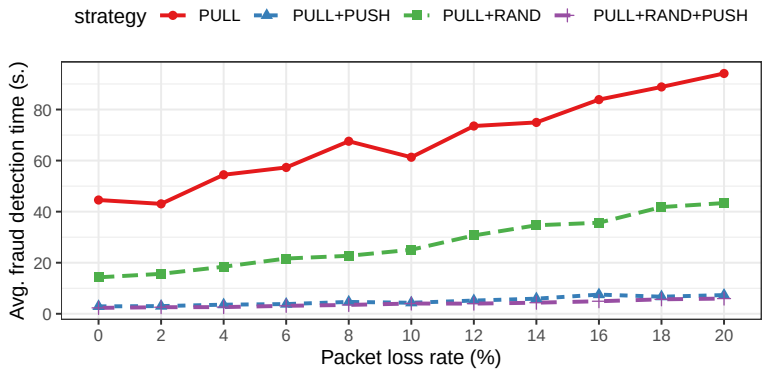


Figure 2.8: The effect of packet loss on the average fraud detection times, for different record exchange strategies.

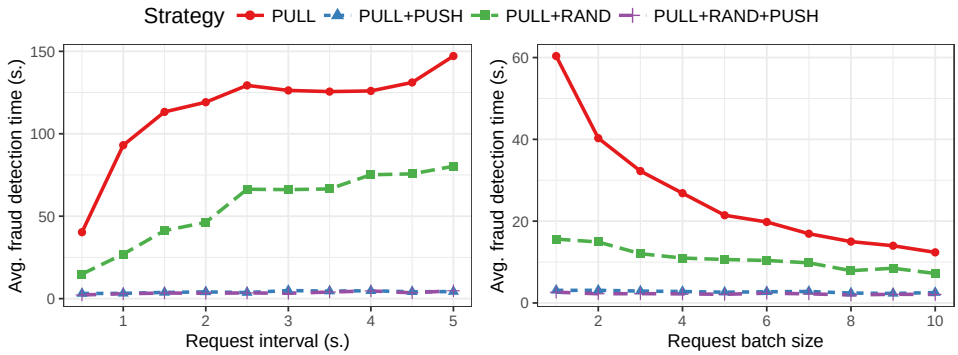
2.6.3 Packet Loss

We reveal the robustness of ConTrib by quantifying the effect of packet loss on the efficiency of fraud detection. To this end, we increase the packet loss rate, up to 20%, and run our simulations under our four record exchange strategies. Even though a packet loss rate of 20% is unlikely for any environment in which ConTrib is to be deployed, we want to analyse how robust ConTrib is, even in such extreme circumstances. We expect fraud detection times to increase when network stability is lower since losing packets makes it more challenging to detect inconsistencies.

Figure 2.8 shows the average fraud detection times when increasing the packet loss rate for our four record exchange strategies. We observe that fraud detection times increase under the PULL and PULL+RAND strategies, whereas this effect is less for the PULL+PUSH and PULL+RAND+PUSH strategies. Average fraud detection times under the PULL strategy increase from 44.6 seconds with no packet loss to 94.1 seconds with 20% packet loss. For the PULL+RAND+PUSH strategy, this same increase is from 2.3 seconds to 6.0 seconds. We notice that with a packet loss of 20% and the PULL strategy, 23.8% of all fraud attempts remains uncovered after the experiment has ended. This number reduces to just 0.2% when no packets are lost. Under the PULL+RAND+PUSH strategy, we note that all fraud attempts are discovered during our experiment, for all evaluated packet loss rates.

2.6.4 Request Interval and Batch Size

We modify the record request interval and batch size, and analyse the effect on the average fraud detection times, see Figure 2.9. Figure 2.9a visualizes the impact of the record request interval on the average fraud detection times for different record exchange strategies. We increase the record request interval, ranging from 0.5 seconds to 5 seconds, in steps of 0.5 seconds. First, we observe that the average fraud detection times for the PULL+PUSH and PULL+RAND+PUSH strategies remains roughly constant. This trend indicates that pushing records to random peers very effectively exposes fraud instances. We also note that average fraud detection times for the PULL and PULL+RAND strategies are increasing when the record request interval becomes larger. For the PULL strategy, we notice that when



(a) Average fraud detection times as the request interval increases.

(b) Average fraud detection times as the request batch size increases.

Figure 2.9: The effect of changing the request interval batch size on the average fraud detection times.

the request interval is over 3 seconds, most fraud instances remain undetected after our experiment finishes. These fraud instances become increasingly harder to detect as the modified record in a personal ledger gets "buried" under additional records. As such, the average fraud detection time is likely higher when running the experiment until all fraud instances have been detected. Unfortunately, we are unable to significantly prolong the experiment duration as our simulations are already at peak memory usage, even after various optimization efforts.

Figure 2.9b shows the average fraud detection times when increasing the number of records queried in each request, the request batch size, from 1 to 10, for different record exchange strategies. Again, the PULL+PUSH and PULL+RAND+PUSH strategies are indifferent to this increase. The average fraud detection time for the PULL strategy decreases from 60.4 seconds to 12.4 seconds when increasing the request batch size from 1 record to 10 records, respectively. This decrease indicates that increasing the request batch size is particularly effective when using the PULL strategy, at the expanse of increased bandwidth usage.

2.6.5 Back-Pointers

We vary the number of back-pointers (b) in each record and analyse the effect on average fraud detection times. We suspect that adding more back-pointers increases the probability of detecting fraud since individual records now bear more hashes of records in ones personal ledger. This advantage comes, however, at a cost of additional network usage and computational overhead to analyse and verify the back-pointers. Each back-pointer adds 32 bytes to the serialized record size.

Figure 2.10 shows the average time until fraud is detected while varying the number of back-pointers and for different record exchange strategies. Adding additional back-pointers indeed decreases fraud detection times. Under the PULL record exchange strategy, it takes 111.2 seconds to detect fraud when no back-pointers are included ($b = 0$). In comparison, this number decreases to 41.6 seconds when adding up to ten back-pointers to each record, a decrease of 58.4%. This decrease is much more for the PULL+PUSH strategy, namely 97%. Note that the effect of adding more back-pointers diminishes for $b > 4$. This

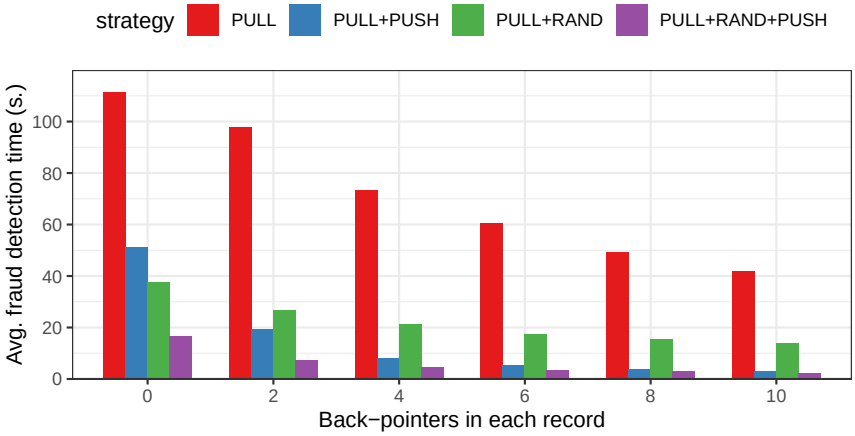


Figure 2.10: The effect of adding more back-pointers to each record on the average fraud detection times, for different record exchange strategies.

effect can likely be attributed to the fact that all peers start with an empty personal ledger in our simulations, and that different records with lower sequence numbers in the same personal ledger are more likely to include identical hashes in their back-pointers. However, we believe that the effect of additional back-pointers becomes more apparent when personal ledgers grow to considerable sizes, since different records are then more likely to include more unique hashes.

2.6.6 Fraud Detection Times under a Realistic Workload

Our experiments conducted so far are using a synthetic workload. We now evaluate the effectiveness of fraud detection in ConTrib of our four considered record exchange strategies using a realistic workload. This workload is derived from deployment data of ConTrib in Tribler, our decentralized file-sharing application. An interaction describes network communication between two peers in a Tor-like overlay. We provide further details on this dataset in Section 2.7. The following experiment replays interactions made during the busiest 24 hours of our deployment period: November 28, 2020. On this particular day, a total of 440'130 records have been created, involving 2'027 digital identities. In the following experiment, we simulate a peer for each digital identity. In line with our prior experiments, each peer commits fraud with a probability of 10% when creating a record. To match the ConTrib settings in our deployed environment (see Section 2.7), we increase the record request interval to 10 seconds.

We noticed that all fraud instances have been detected after our experiment ends. The average fraud detection time for the `PULL` strategy is just 29.4 seconds whereas this number decreases to 18.5 seconds for the `PULL+RAND+PUSH` strategy. 2.5% of all fraud instances take longer than three minutes to detect, with the highest detection time being 1'276 seconds (just over 21 minutes). Figure 2.11 shows the Empirical Cumulative Distribution Function (ECDF) of fraud detection times, for the evaluated strategies. For presentation clarity, we

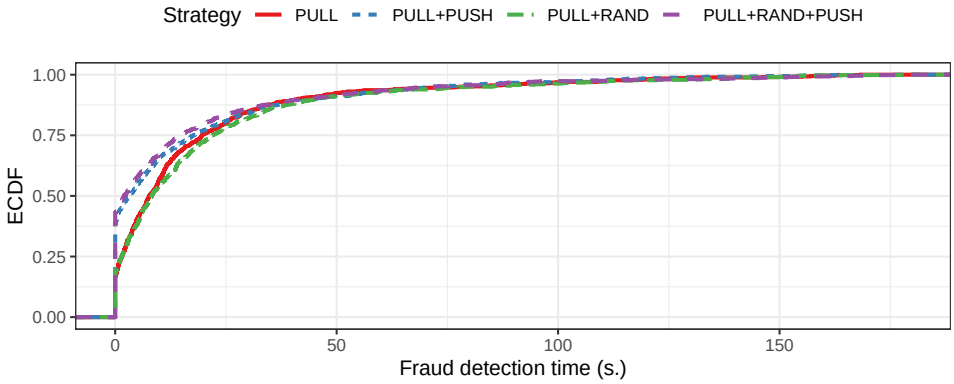


Figure 2.11: Fraud detection times lower than 180 seconds for different record exchange strategies when replaying a day of interactions made by Tribler users.

only show the detection times of fraud instances lower than three minutes. We observe that over 25% of all fraud for the PULL+PUSH strategy is detected immediately. During this experiment, we also see that the network usage per peer is relatively low. For the PULL strategy, each peer, on average, consumes merely 1.7 MB of hourly network traffic. This number increases to 5.1 MB under the PULL+RAND+PUSH strategy.

This experiment shows that ConTrib, under a realistic workload of Tribler interactions, exhibit relatively low fraud detection times and has low network usage. As we have also measured during our deployment trial (see Section 2.7), the resource overhead of ConTrib is minimal. For Tribler, the fraud detection times shown in Figure 2.11 are acceptable. However, as we have also shown in our other experiments, these detection times can further be decreased with more frequent record crawling.

2.6.7 Discussion

In summary, our experiments demonstrate that ConTrib exhibits low fraud detection times, scales when the network grows, has reasonable bandwidth overhead, and is robust against packet loss. We have also demonstrated the effect of the number of back-pointers in each record on the efficiency of fraud detection. Finally, we have shown that ConTrib remains effective at detecting fraud when using a realistic workload. Even though we have not evaluated the effect of all parameters in Table 2.1, we believe that this set of experiments provides a good starting point for system designers to adopt and configure ConTrib. With our open-source simulator, system designers can quickly analyse the effect of different parameters, guided by a synthetic or realistic workload that resembles the behaviour in their application.

We have demonstrated that there is a trade-off between the average fraud detection times and network usage. The acceptable network usage depends on the application. For example, bandwidth is less likely to be a bottleneck when considering a video streaming application compared to an Internet-of-Things environment with low-resource devices. By reducing the record request intervals, fanout value, and the maximum number of back-

pointers, one can reduce the bandwidth footprint of ConTrib, at the cost of increased fraud detection times. Figure 2.7c shows that the active record exchange strategy has a notable impact on network usage. In a dynamic network where the sessions of peers are short-lived, we recommend the PULL+RAND or PULL+RAND+PUSH strategies, under which peers share random records in their database with others. This strategy allows the detection of fraud attempts of offline peers. We recommend the PULL+RAND+PUSH strategy when fraud must be detected quickly. We recommend the PULL strategy when bandwidth is a limiting factor.

2.7 Applying ConTrib to Address Free-riding at Scale

To show the effectiveness of ConTrib with a real-world application, we conduct a large-scale deployment trial of ConTrib with Tribler. Tribler is our decentralized file-sharing application and is downloaded by over 1.8 million users [78]. This application features an onion-routing overlay that tunnels BitTorrent traffic through *relay and exit nodes* to provide anonymity. Unfortunately, the Tribler network suffers from an undersupply of exit nodes, leading to frequent network congestions and overall degradation of download speeds for all users. We employ ConTrib to account the performed work by relay and exit nodes, and the consumed work by downloaders. We then punish free-riding behaviour by offering users with higher net contributions preferential treatment during periods of congestion. In the remainder of this section, we elaborate on the integration of ConTrib in Tribler, on the collected data, and on the effectiveness of free-riding detection.

2.7.1 Accounting Bandwidth Transfers

We enable peers to earn *bandwidth tokens* by providing services as a relay or exit node in the Tribler network. The mutations in bandwidth token balances of each peer is accounted in ConTrib records. For example, a payout corresponding to a data exchange of a 50 MB file between peers a and b deduces 50 MB of a's balance and increments b's balance by 50 MB (MB is the unit of this bandwidth token). Peers can have a negative balance of bandwidth tokens, in which case they have enjoyed more services from others than they contributed back to the network.

When a peer downloads content using Tribler, the Tribler software establishes a *circuit*, containing exactly one exit node and optionally some relay nodes. This is comparable to circuits in the Tor protocol. All traffic is securely routed through relay and exit nodes. Figure 2.12 shows how bandwidth tokens are paid out after a peer has downloaded a 50 MB file over a three-hop circuit (with two relay nodes and one exit node). The downloader accounts a transfer of 250 MB to the first relay using ConTrib. Specifically, the downloader creates a proposal record, containing the pair-wise byte counter with the first relay and the magnitude of the current payout. In the latest version of Tribler, newly created records are by default disseminated to 20 random peers. Each peer also requests a random record from another known peer every 10 seconds. During our deployment period, we have periodically revised the record exchange strategy in response to the observed network behaviour and growth.

After the downloader has finished the payout to the first relay, the first relay then transfers 150 MB to the next relay, resulting in a net positive token balance of 100 MB for

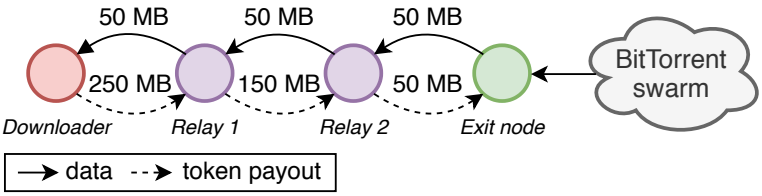


Figure 2.12: Accounting specifications of an anonymous 50 MB BitTorrent download over a three-hop onion-routing circuit.

the first relay. The rationale behind our payout scheme is that we reward relay and exit nodes for performing the cryptographic operations on the forwarded data. Relays that do not forward the payout to the next hop will be added to the local blacklist by the previous hop, therefore lowering their opportunity to earn bandwidth tokens in the future.

Since this use-case involves anonymous downloading, there is an important trade-off between accountability and anonymity. We plan to address privacy concerns around the accounting of bandwidth transfers by having each peer aggregate and delay payouts. This privacy-enhancing technique is introduced in the work of Palmieri et al. [117]. Still, work accounting with ConTrib does not leak the identity of a downloader to other peers in the network, nor reveals any data being transferred over circuits. To address the uncontrolled minting of bandwidth tokens by accounting fake work, we are currently looking into the design and deployment of a Sybil-resistant reputation mechanism [98].

2.7.2 Circuit Assignment

We use the bandwidth token balances included in ConTrib records to grant preferential treatment to dedicated peers during periods of congestion. Specifically, we modify the Tribler protocol such that each relay and exit node maintains a fixed number of *slots*. A circuit that includes a relay or exit nodes consumes one slot at their side. We distinguish between *random* and *competitive* slots. Random slots are filled on a first-come-first-serve basis whereas the assignment of competitive slot is based on the bandwidth token balance of a circuit initiator. The intuition behind this approach is to still give peers with lower trust scores an opportunity to acquire a random slot but at the same time, give preferential treatment to well-behaving peers with competitive slots. The total number of such slots is flexible and depends on the hardware capabilities of the node operator since our Tor-like routing protocol is heavy on CPU usage.

A pseudocode description of the slot assignment logic is given in Algorithm 4. When a circuit initiation request arrives, the `onCircuitRequest` method is invoked and Tribler first determines if there is a random slot available (line 5-10). If so, we assign the new circuit to the random slot (line 7). If no random slot is available, Tribler queries the bandwidth token balance of the circuit initiator i by requesting the records in the personal ledger of i (line 11). When receiving these records, Tribler determines the current bandwidth token balance and checks eligibility for a competitive slot (line 14-30). If there is an unoccupied competitive slot, Tribler assigns the new circuit to it (line 19). If all competitive slots are filled, the circuit of the initiator with the lowest amount of bandwidth tokens, say p, is destroyed if the token balance of i is higher than the token balance of p (line 28). This

Algorithm 4 The assignment logic of slots to circuits. *numRand* and *numComp* represent the maximum number of random and competitive slots, respectively.

```
1:  randomSlots ⟵ [⊥]* numRand
2:  competitiveSlots ⟵ [(−∞, ⊥)]* numComp
3:
4:  function onCircuitRequest(circuit)
5:      for i = 0 to numRand do
6:          if randomSlots[i] = ⊥ then
7:              randomSlots[i] ⟵ circuit
8:              return
9:          end if
10:     end for
11:     Query the balance of the initiator of the circuit
12: end function
13:
14: function onBalance(circuit, balance)
15:     lowestBalance ⟵ ∞
16:     lowestIndex ⟵ ∞
17:     for i = 0 to numComp do
18:         if competitiveSlots[i] = (−∞, ⊥) then
19:             competitiveSlots[i] ⟵ (circuit, balance)
20:             return
21:         end if
22:         if competitiveSlots[i][0] < lowestBalance then
23:             lowestBalance ⟵ competitiveSlots[i][0]
24:             lowestIndex ⟵ i
25:         end if
26:     end for
27:     if balance > lowestBalance then
28:         destroyCircuit(competitiveSlots[lowestIndex][1])
29:         competitiveSlots[lowestIndex] ⟵ (circuit, balance)
30:     end if
31: end function
```

pre-emptive approach frees up the competitive slot for the circuit of *i*. As a result, users with a higher token balance have more chance to claim a competitive slot during periods of congestion, compared to free-riders, and experience higher and more stable download speeds. We consider the analysis of different slot allocation policies, e.g., using a packet-granular scheduler [118], as further work.

2.7.3 Data Collection

We integrate both ConTrib and the slot assignment logic in Tribler and release a new version of our software. We also deploy a crawler that continuously requests ConTrib records from random peers in the Tribler network. This crawler selects a random peer in the Con-

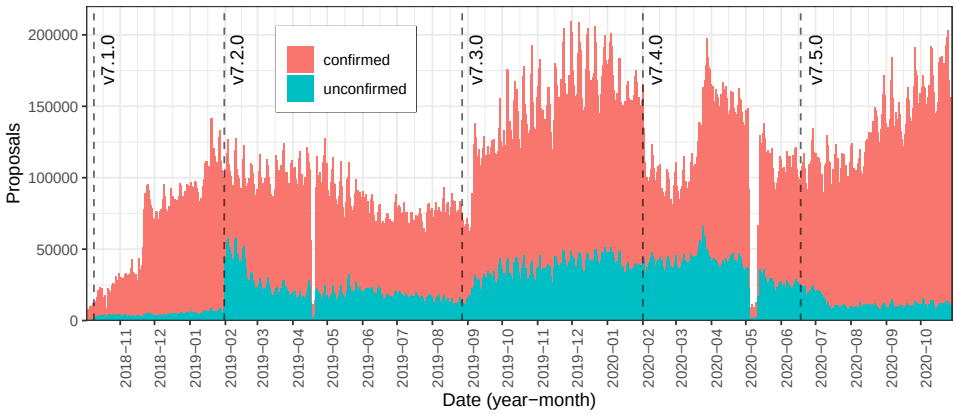


Figure 2.13: Daily creation statistics of proposals, during our two-year deployment trial. We show the amount of confirmed and unconfirmed proposals. We annotate the major releases of Tribler.

Trib network every two seconds and requests missing records in their personal ledger. The crawler statistics are published on a public website.[5] A deployment period of two years has resulted in more than 160 million records, created by over 94'000 peers. The crawler stores collected records in a sqlite database that is enhanced with additional indices to speed up insertion and analysis queries. The file size of the database with all collected records is around 120 GB, and we plan on releasing the full data set later. Our crawler discovered 127'135 proofs of fraud, either detected by the crawler itself or by other peers. We also find that 11.4% of all collected proposals in the deployed ConTrib network is unconfirmed. This relatively high fraction of unconfirmed proposals is either because of software bugs, because the proposal counterparty has not created a confirmation, or because our crawler has not picked up the confirmation record. At the same time, the amount of work included in each record is relatively small, reducing the impact of unconfirmed records.

Deploying a crawler and monitoring the records created by ConTrib allows us to detect anomalies caused by software bugs or unexpected user behaviour. It also provides us with the means to monitor the growth of users within Tribler by tracking the number of unique peers in the ConTrib network. We have also included a creation timestamp in the payload of each record created with Tribler, allowing us to perform a time-based analysis. We note that this timestamp might not accurately reflect the creation time of the record since users could have misconfigured their time-zone settings.

Figure 2.13 shows the daily number of created proposal records. We annotate the dates on which we released a major version of Tribler. Figure 2.13 reveals that more users run Tribler during the weekend and create more proposals on a Saturday and a Sunday. We also observed two large-scale outages of exit nodes, in April 2019 and May 2020, likely due to infrastructure failures. Despite these outages, users would still perform payouts when downloading directly from other Tribler users without anonymity (since this does not involve exit nodes).

[5]See `http://explorer.tribler.org`

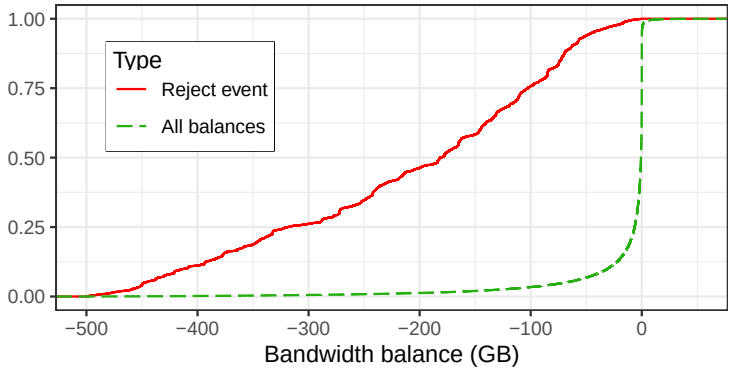


Figure 2.14: Empirical Cumulative Distribution Function (ECDF) of the bandwidth token balances of peers and individual rejects events at exit nodes.

2.7.4 Free-rider Identification and Service Refusal

To show the effect of ConTrib and our slot assignment mechanism on free-riders, we deploy 48 exit nodes in the Tribler network, running on the same machine. Each exit node is configured with a total of 10 random slots and 20 competitive slots, resulting in a total of 1'440 slots. We determined this number of random and competitive slots based on the hardware capacity of our machine. We are specifically interested in the situation when a peer is unable to claim a slot, due to their bandwidth token balance being insufficient. We refer to this situation as a *reject event*. For each reject event, we log the bandwidth token balance of the rejected peer. In total, we logged over 1.4 million reject events over three weeks.

Figure 2.14 shows an Empirical Cumulative Distribution Function (ECDF) with the bandwidth token balances of all peers (dotted green line) and the balances associated with rejected circuit requests (solid red line). For presentation clarity, we filter out all peers and reject events with balances higher than 50 GB or lower than -500 GB. Many Tribler users have a negative bandwidth token balance. The median token balance of all users is -713 MB. This number suggests that there is not much opportunity to earn bandwidth tokens by contributing to the network. By default, the Tribler software downloads content over a 1-hop circuit, only involving an exit node. Changing the default behaviour to use 2-hop downloads could alleviate this issue, at the cost of decreased download speeds. Figure 2.14 also shows that users with a relatively low token balance (e.g., < 50 GB) are frequently rejected a slot. The median token balance associated with reject events is -181.4 GB, demonstrating that our mechanism effectively targets peers with lower bandwidth token balances. The competitive slots claimed by free-riders will likely go to dedicated peers when the network is congested. This deployment trial shows that ConTrib is effective at detecting and addressing free-riding behaviour in Tribler. Based on our observations, we believe that the integration of ConTrib has increases network performance and helps to maintain fairness amongst downloading users.

2.8 Conclusions

We have presented ConTrib, a universal accounting mechanism to maintain fairness in decentralized applications by accounting work. The ConTrib data structure uses records to capture the work performed by peers. Each peer maintains a tamper-evident personal ledger with interlinked records. Fraud, the illegitimate modification of a record in ones personal ledger, is optimistically detected through the random exchange of records and thorough validation of incoming ones. We have devised a system architecture of ConTrib and implemented it. Our evaluation has demonstrated that ConTrib is capable of detecting fraud within seconds, even when the network grows to 10'000 peers and when scaling the system load. Through a two-year deployment trial of ConTrib in Tribler, involving more than 94'000 users, we have successfully addressed free-riding behaviour.

We envision and encourage the usage of ConTrib beyond work accounting in decentralized applications. Currently, ConTrib is being evaluated in different scenarios that require accountability, including decentralized trading, software developer portfolios, decentralized crowdsourcing, and self-sovereign identity [105, 113, 114]. Since these scenarios leverage the same data structure, these applications all benefit from the accounting capabilities and integrity guarantees that ConTrib offers.

3

MATCH: A Decentralized Middleware for Fair Matchmaking In Peer-to-Peer Markets

Order matchmaking is a core enabling element for electronic markets and online economy. A common approach to order matchmaking is the deployment of proprietary solutions, controlled by the market operators. This approach raises fairness concerns as market operators effectively have the capability to discriminate specific users when matching their orders. Blockchain technology has been proposed to enable transparent, open matchmaking solutions without a trusted operator. In practice, however, blockchain technology does not provide the required performance, in terms of transaction throughput, for fast order matching across many domains.

In this chapter we present MATCH, a decentralized middleware for fair order matchmaking. By decoupling the dissemination of potential matches from the negotiation of trade agreements, MATCH empowers end users to make their own educated decisions and to engage in direct negotiations with trade partners. This approach makes MATCH resilient against matchmakers that pursue selfish interests, a severe issue with centralized matchmaking. We implement MATCH and evaluate our middleware using real-world ride-hailing and asset trading workloads. It is demonstrated that MATCH maintains high matching quality, even in the presence of malicious matchmakers. Further, we show that the bandwidth, memory usage, and order fulfil latency of MATCH is orders of magnitude lower compared to matchmaking on an Ethereum blockchain.

3.1 Introduction

The deployment of peer-to-peer markets by companies operating in the sharing economy has been hailed to boost the global economy [119]. Beyond the promises of increased economic welfare, the broader appeal of the sharing economy also lies with the development of new modes for the sharing of unused or underutilized assets, such as cars and houses. Estimations on the impact of the sharing economy suggest an increase in global revenue from $14 billion in 2014 to $335 billion by 2025, partially enabled by major platforms such as Uber (ride-hailing) and Airbnb (house-sharing) [120].

The effect of these platforms on peer-to-peer markets, however, is not unequivocal. It has been argued that market operators exploit their prominent position and charge high transaction fees for their role as intermediary [69]. Market operators gain unprecedented power through the control of all the key enabling elements for electronic marketplaces, including settlement, arbitrage, and matchmaking [121–123]. The latter element is of particular interest as at the dawn of e-commerce matchmaking solutions were envisioned to create open, fair, and competitive markets on the Internet [124].

Matchmaking in electronic markets can be considered as the process of mediating between supply and demand [125]. Currently, centralized matchmaking is the approach taken by most commercial market operators [121, 123]. With centralized matchmaking, market operators deploy proprietary servers that are optimized to efficiently match new buy and sell orders with existing ones within a specific domain. A key advantage of centralized matchmaking is that the market operator can match incoming orders with the (current) best compatible order since they maintain all market information.

Unfortunately, this also enables market operators to exploit the marketplace through the practices of unfair matchmaking to increase intermediary revenues [126]. Manipulation in the matchmaking process was exposed through analysis of different e-commerce platforms and financial exchanges [44, 121]. An emblematic example of this issue is the practices of Uber, implementing price discrimination and phantom matches to manipulate the behaviour of users [127]. It has recently been demonstrated through experiments that the matchmaking algorithm of Uber undermines revenues of drivers to the advantages of the platform operator [71]. As researchers point out, unfair matchmaking is a complex, multilayered issue that can not be mitigated only with algorithmic adjustments [71]. We suggest that this problem requires a next step towards a different approach to matchmaking in peer-to-peer markets.

It is technologically feasible to have market participants carry out the matchmaking process themselves, without trusted operator. In particular, blockchain technology provides the means to record and match market orders on a distributed ledger [62]. Smart contracts, self-executing programs stored on a blockchain, are capable of executing the matchmaking logic [128]. Even though it seems like an appealing solution to fairness issues, the consensus algorithm managing the blockchain ledger is vulnerable to various attacks against fairness, specifically, *transaction re-ordering* and *front-running* [129–131]. These attacks effectively allow consensus participants to influence how specific orders are matched. In addition to these threats, scalability issues inherent to all the blockchain protocols based on a global consensus carry significant limitations on the speed of matchmaking, as we will experimentally show in Section 3.6.3 [132].

The ineffectiveness of matchmaking on a blockchain is also identified by various de-

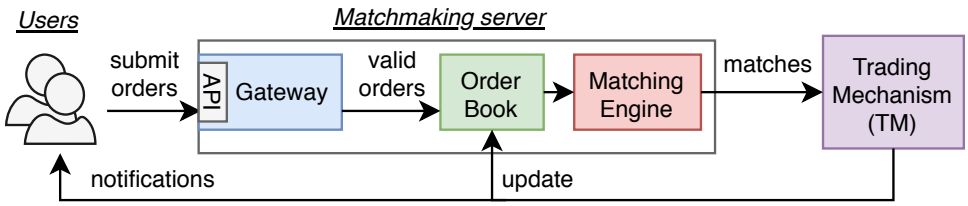


Figure 3.1: A generic model for centralized matchmaking in peer-to-peer markets, using a single matchmaking server.

centralized exchanges that are operated by a blockchain, also called DEXes [35, 46, 129]. In response, these DEXes opt for a federated approach where any participant can host a matchmaking server and can act as a matchmaker. In practice, most orders in these DEXes are managed by servers that are under the control of the exchange operator and therefore still carry limitations on the achievable level of fairness desirable on such markets.

In summary, there is a dilemma of choice between two desirable properties of matchmaking mechanisms. *Efficiency* of matchmaking achievable through the concentration of orders by a trusted central party, versus provable *fairness* of matchmaking achievable with the transparency of decentralized on-chain matchmaking. We argue, however, that this dilemma does not present an insurmountable obstacle for the implementation of efficient and fair matchmaking solutions.

Contributions. We present MATCH, a decentralized middleware for fair matchmaking in peer-to-peer markets. Our solution is based on the principle of strictly decoupling the matching process from the negotiation of trade agreements. Our first contribution is the MATCH protocol (Section 3.4) where any user can act as a matchmaker for others. Matchmakers store open orders created by users, match incoming orders with existing ones, and inform order creators about potential matches. Users then engage in trade negotiation with prospective counterparties. This approach makes MATCH highly resistant against matchmakers which deviate from a standard matching algorithm. The second contribution is the generic MATCH middleware architecture (Section 3.5). MATCH does not rely on the specifications of orders and is therefore re-usable across different trading domains.

We devise the first decentralized and fair alternative to the Uber ride-hailing market (Section 3.6.1), to the best knowledge of the authors. Even when 75% of all drivers prioritize their own ride services during matchmaking, negotiated matches in our market maintain a high quality. Furthermore, with a real-world asset trading workload (Section 3.6.2) we show that MATCH is asset-agnostic, enabling the deployment of open and universal matchmaking infrastructures. Finally, we show that MATCH has bandwidth usage and order fulfil latencies that are several orders of magnitude lower compared to matchmaking on an Ethereum blockchain (Section 3.6.3).

3.2 Towards Decentralized Matchmaking

In our approach, users in a peer-to-peer market conduct the matchmaking process themselves. To elaborate on the components used in our solution, we first devise a generic, centralized model for matchmaking. We then identify technical concerns that arise when decentralizing this model.

3.2.1 Centralized Matchmaking

We devise a generic model for centralized matchmaking in peer-to-peer markets, see Figure 3.1. This model is the starting point for our fair matchmaking solution and is inspired by existing architectures that have been widely used by financial markets [42, 44]. Users create *orders*, which they then submit to the matchmaking server. An order specifies interest to buy or sell assets, resources, or services (orders are further discussed in Section 3.4.1). Many markets deploy one or more gateways that filter out invalid orders and mitigate targeted attacks on the matchmaking server, such as a DDoS attack [44]. Valid orders are inserted in the *order book*, a local data structure that bundles all open and valid orders.

Upon the insertion of a new order in the order book, it is forwarded to the *matching engine*. This component searches for existing orders in the order book that match with the newly submitted order. In particular, an incoming order should be matched with the current best compatible order(s). Whether two orders match is predicated by a *matching policy*. For example, the price-time strategy is a predominant matching policy in financial markets where orders are first matched based on their price and then on their creation time (prioritizing older orders) [44]. The matching engine can establish multiple matches for a single order, e.g., a buy order for many assets can be matched with multiple (smaller) sell orders. Established matches should be "executed", which is an application-dependent operation. In a financial exchange, for example, the specified assets in the matched orders should be exchanged between the order creators. In a ride-hailing market, however, the driver and passenger should be put in contact with each other. We model the component that executes established matches as the *trading mechanism* (TM), which we consider external to the model in Figure 3.1. After established matches have been executed by the TM, the affected orders in the order book are updated (or removed if they are completed), and the order creators are notified of the match execution.

3.2.2 Decentralized Matchmaking

The model in Figure 3.1 allows the server operator to conduct unfair matchmaking by manipulating the matching engine (or policy) to hide, prioritize, or delay specific orders. To address this situation, we propose a solution where users (order creators) themselves carry out matchmaking while ensuring that no single user can authoritatively decide on how the orders of other users are executed. We first consider a basic, decentralized architecture where every user operates a single matchmaking server. A user then submits a new order to all matchmaking servers, and all servers forward established matches to the same TM. The TM executes incoming matches in a FCFS manner. This approach, however, raises the following technical concerns:

1) How does a decentralized matchmaking architecture process matches of the same order found by distinct matchmaking servers? Deploying a single matchmaking server prevents the situation where distinct matchmaking servers submit the same or different (valid)

matches for the same order to the TM. Assuming a FCFS execution of incoming matches by the TM, having multiple matchmaking servers sending matches to the TM likely results in the situation where matches of the same order are executed multiple times, resulting in an incorrect order state. To ensure correctness, our replicated matchmaking architecture requires additional coordination.

One solution involves the periodic execution of a Byzantine fault tolerant consensus protocol by matchmaking servers to decide which matches are sent to the TM. Unfortunately, reaching consensus is a resource-intensive process and existing protocols, e.g., PBFT [133] or Proof-of-Work [9], do not scale when the number of matchmaking servers or orders increases [111]. We avoid the need for consensus by having users *negotiate* trade agreements with potential counterparties (further described in Section 3.4.4). Upon a positive negotiation outcome, these trade agreements, digitally signed by both parties, are sent to the TM and executed. Matchmakers only *notify* users about potential matches for their orders. Since it is in the best interest of users to correctly manage their orders, rational users will not sign trade agreements that would result in an incorrect order state.

2) Is it required to disseminate a new order to all matchmaking servers? In the architecture described above, a user sends a new order to all matchmaking servers. This results in full replication of the order book, under the assumption that all matchmaking servers eventually receive every new order. The problem is that a flooding-based dissemination strategy leads to severe performance degradation when the number of matchmaking servers increases, as illustrated by deployed peer-to-peer applications like Gnutella [134]. We address this concern by sending a new order to a small, random subset of all matchmaking servers such that it is still likely that at least one honest matchmaking server receives compatible orders (further described in Section 3.4.2).

3.3 System and Threat Model

We address the aforementioned concerns and present a decentralized middleware for fair matchmaking, named *MATCH*. We now outline the system and threat model of MATCH.

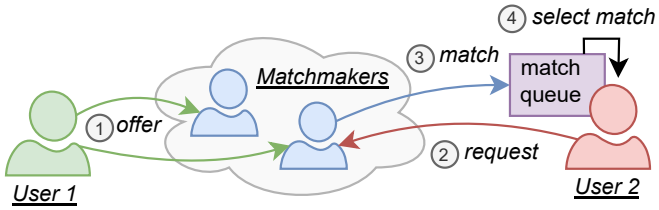
Market and Order Model. We adopt a continuous market model where orders are matched in a FCFS manner. This model is commonly used by peer-to-peer markets (e.g., by Uber). To represent a two-sided market with supply and demand, we introduce two order types: *offers* and *requests*. Offers, respectively requests, indicate interest to sell, respectively buy specific assets, services, or resources. To ensure re-usability across different markets, matchmakers in MATCH can host multiple order books and manage orders with differing specifications. Each order has a quantity q, which is an integer value indicating the number of assets, services, or resources being offered or requested. The state of an order can be either *open* (when the order has a positive quantity, $q > 0$), *completed* (when all quantity in the order has been traded, $q = 0$), *cancelled* (when the order has been cancelled by its creator) or *expired* (when the timeout of the order has expired). The structure and content of an order are further elaborated in Section 3.4.1.

Actors. We refer to an entity in the MATCH network as a *node*. A node can act as a *user*, *matchmaker*, or take on both roles. Users disseminate offers and requests to matchmakers. MATCH requires the active participation of users to negotiate with other users and thus requires users to be online for their order to be completed (also see Section 3.4.4).

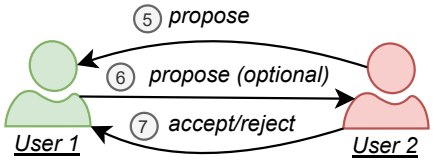
Network. We aim for MATCH to be deployed in a WAN environment. We consider an unstructured peer-to-peer network where each node knows the network addresses of active matchmakers. This can be achieved by maintaining a list of matchmakers, e.g., on a website or through a decentralized publishing network like the Kademlia DHT [37]. New matchmakers enrol themselves on this list while matchmakers leaving the network un-enrol from this list. As we show in Section 3.4.2, MATCH is able to deal with offline matchmakers that are still enrolled on the list. Users periodically download the latest version of this list to keep up with the set of active matchmakers in MATCH.

Each node possesses a public and private key. The public key is used to identify the node in the network whereas the private key is used to sign all outgoing network messages. We assume that the digital identity of each node in MATCH is tied to a real-world identity, preventing uncontrolled identity creation (also known as the Sybil Attack) [101]. This is not an unrealistic requirement since many electronic marketplaces already impose identity verification in order to participate [135]. Identity verification can, for example, be achieved by using the services of a centralized registration authority. We note, however, that a centralized dependency might be undesirable in marketplaces with a decentralized structure. In such marketplaces, we encourage the use of (semi-)decentralized solutions for identity management, like self-sovereign identities [136, 137].

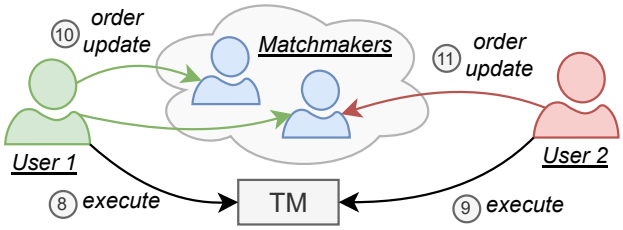
Threat Model. In this work our threat model orients around *malicious matchmakers* that deviate from a standard matching policy and match incoming orders according to a custom policy. For example, a matchmaker can deliberately match an incoming offer with the second-best request in their order book and match one of their own offers with the best request instead. This threat model also captures collusion, the situation where a subset of matchmakers agrees to match orders according to a custom policy to gain economic benefit as a group. Malicious matchmakers are often driven by economic incentives, e.g., when a matchmaker wants to prioritize its own orders or when a group of matchmakers collectively attempt to drive competitors out of business by ignoring their orders. Malicious matchmakers directly affect market fairness since they treat incoming orders unequally. We assume that cryptographic protocols are secure and that the computational power of adversaries is bounded.



(a) Order creation and dissemination.



(b) Order negotiation.



(c) Match execution and order updates.

Figure 3.2: High-level overview of the MATCH protocol and the message exchange between users and matchmakers.

3.4 The MATCH Protocol

We visualize the MATCH protocol and the message exchange between users and matchmakers in Figure 3.2. The key idea behind our protocol is that matchmakers only inform users about matches and that users negotiate a trade directly with counterparties. First, users send new offers and requests to one or more matchmakers (step ① + ②). Matchmakers match incoming orders with existing orders in their order books and notify users about potential matches (step ③). Users aggregate potential matches of a specific order in a match queue. Some period after receiving the first match for a specific order, a user starts to process matches in the associated match queue (step ④), starting with the best match, and negotiates with the user behind the matched order (step ⑤ - ⑦). When the negotiating parties reach a trade agreement (in other words, intend to fulfil their orders with each other), they execute the negotiated trade agreement by sending it to the TM (step ⑧ + ⑨). The negotiating parties then inform the matchmakers about the executed trade, so they can update the state of the affected orders accordingly (step ⑩ + ⑪). The matchmakers are also informed about a negative outcome during the negotiation process.

If an order is still open, a user selects the next best item from the associated match queue, if it is non-empty, and initiates the next negotiation process. This repeats until the match queue is empty or the order is completed. The steps in the MATCH protocol are now further explained.

3.4.1 Order Creation

In MATCH, users create new orders to indicate their willingness to buy or sell resources, services, and assets. Listing 3.1 exemplifies the structure of an order in MATCH that specifies a transportation request in a ride-hailing market. This order, in JSON format, includes the waiting location of the order creator in the `data` field. The content of the `data` field is flexible and depends on the context in which the order is created. It can contain many attributes and constraints that affect how the order is matched. Each order has a `type` field which is a string value indicating the type of the order. The order type is used by matchmakers to apply the right policies for validation and matching, and to store the order in the appropriate order book. In Listing 3.1, the `RIDE` type indicates an order in a ride-hailing market. Similarly, an order with type `EUR/USD` can indicate an order trading Euro for Dollar. The `is_offer` field is a flag that indicates whether the order is an offer or a request. The identifier in an order is an integer value that indicates the position of the order in the sequence of all orders created by that user. Together with the public key of the order creator, it uniquely identifies an order in the network. The digital signature in an order allows matchmakers to verify its authenticity. Inclusion of the `timeout` value prevents orders from being included in order books for an indefinite amount of time. Finally, each order has a quantity, which is an integer value that specifies the amount of assets, services or resources being offered or requested. After creation, a user serializes the order in an `order` message and sends it to matchmakers.

Users can cancel an open order, say O, at any time by sending a `cancel` message with the identifier of O and its public key to matchmakers. A `cancel` message for O should be sent to the same matchmakers as the `order` message that contained the description of O. Therefore, users keep track of the matchmakers to which they have sent an `order` message. Upon reception of a `cancel` message, matchmakers remove the cancelled order from their

Example 3.1: An order in a ride-hailing market (in JSON format).

```
{       "timestamp": "2020-02-24T09:09:19+0000",
        "type": "RIDE",
        "timeout": 3600,
        "is_offer": false, // request for transportation
        "public_key": "82ae2f8f0c473cbdf63b920...",
        "signature": "d54af87c8f8e6d917729d14...",
        "identifier": 5,
        "quantity": 1,
        "data": {
                "latitude": "40.712776",
                "longitude": "-74.005974"
        }
}
```

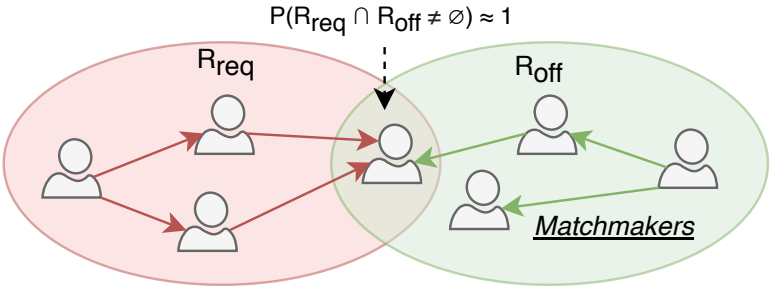


Figure 3.3: The intuition behind order dissemination in MATCH.

order book.

3.4.2 Order Dissemination

In the replicated matchmaking model proposed in Section 3.2.2, a new order is disseminated to all matchmakers. We now address concern 2 from Section 3.2.2 and show how to considerably decrease the fanout of `order` messages (i.e., the number of matchmakers that a specific `order` message reaches) while still ensuring a high probability that a new order reaches a matchmaker with the current best matching order in their order book. Specifically, we send a new order to a random subset of all matchmakers. Let R_{req} and R_{off} indicate the sets of matchmakers that receive a specific request and offer, respectively. The probability that at least one matchmaker will receive both a specific offer and request quickly goes to one as the order fanout increases, even when the order fanout is low compared to the number of matchmakers. This phenomenon is also known as the "birthday paradox" and is in practice exploited when computing hash collisions or when detecting a double spend attack in the Bitcoin network [138]. Figure 3.3 shows the intuition behind this idea. The figure shows in red the set of matchmakers that receive a particular request and in green the set of matchmakers that receive a particular offer. One matchmaker receives both the request and offer and is able to match these orders.

Determining to how many matchmakers a new order is sent, is key. In particular, we are interested in computing the probability that at least one matchmaker receives a matching offer and request. If we consider a network with 1'000 matchmakers where new orders are disseminated to 50 matchmakers, this probability is given by $\frac{1000-50}{1000} \cdot \frac{1000-50-1}{1000} \cdots \frac{1000-50-49}{1000}$. The probability that there is at least one matchmaker amongst all m matchmakers receiving both an offer and a request, with order fanout f, is equal to:

$$P(R_{req} \cap R_{off} \neq \emptyset) = 1 - \prod_{i=0}^{f-1} \left(\frac{m-f-i}{m}\right) \tag{3.1}$$

The value of $P(R_{req} \cap R_{off} \neq \emptyset)$ quickly increases when f increases. Even if $m = 100'000$ and $f = 500$ (i.e., orders are sent to only 0.5% of all matchmakers), the probability that at least one matchmaker receives both a matching offer and a request, is already 97.7%.

In this setting, it reduces the required fanout of an `order` message from 200'000 (when disseminating a new order to all matchmakers) to merely 1'000, thus significantly reducing the network traffic required for order dissemination. Note that the value of m is known to users in MATCH since they possess a list of all matchmakers. We envision that the MATCH software uses a default target value for $P(R_{req} \cap R_{off} \neq \emptyset)$ (fixed to 0.95 in our experiments). Depending on the application domain and the number of incoming match messages, users can increase or decrease the fanout as they see fit.

Malicious Matchmakers. Equation 3.1 assumes that all matchmakers in the set $R_{req} \cap R_{off}$ follow the protocol and actually inform order creators when receiving matching orders. This assumption violates our threat model since malicious matchmakers can respond with sub-optimal matches, or not respond with matches at all. Therefore, we modify Equation 3.1 to account for the situation where a fraction r of all matchmakers is malicious. Intuitively, this situation would require a higher value of f in order to reach at least one honest matchmaker. Given a fraction of malicious matchmakers, denoted as r, $P(R_{req} \cap R_{off} \neq \emptyset)$ is now equal to:

$$P(R_{req} \cap R_{off} \neq \emptyset) = 1 - \prod_{i=0}^{f-1} \left(\frac{(m - \lfloor (1-r)f \rfloor - i}{m} \right) \tag{3.2}$$

We show in the next subsection that this is an appropriate modelling of malicious matchmakers. The rationale behind this model is as follows. The quality of matches from malicious matchmakers is likely to be lower compared to those received from honest matchmakers. Therefore, there is a high probability that the effect of malicious matchmakers is negated upon receiving matches from an honest matchmaker, since a user will process the matches of honest matchmakers first. Reaching honest matchmakers in the presence of malicious matchmakers requires a higher fanout value.

When a matchmaker receives an `order` message describing an order O, it matches O with existing orders in its order book with the same type, according to a matching policy (see Section 3.5). For each order matched against O, the matchmaker constructs a `match` message and sends it to the user that created O (step ③ in Figure 3.2). A `match` message contains the full specifications of the matched orders and the network address of the user behind the matched order. This network address is used by the receiver of the `match` message to initiate the order negotiation process with the user behind the matched order.

3.4.3 Match Queue

Upon receiving a `match` message from a matchmaker, a user contacts the creator of the matched order and initiates a negotiation process (further discussed in Section 3.4.4). A potential strategy is that the user immediately contacts another user upon the arrival of a `match` message. This possibly minimizes the time for an order to be completed. However, this strategy leaves the user vulnerable to an attack where a malicious matchmaker is the first to send a specific `match` message to a user, which immediately triggers the negotiation process. The quality of the received match described by the `match` message might be poor and the user might have received a match with a higher quality if it had waited for additional matches from honest matchmakers.

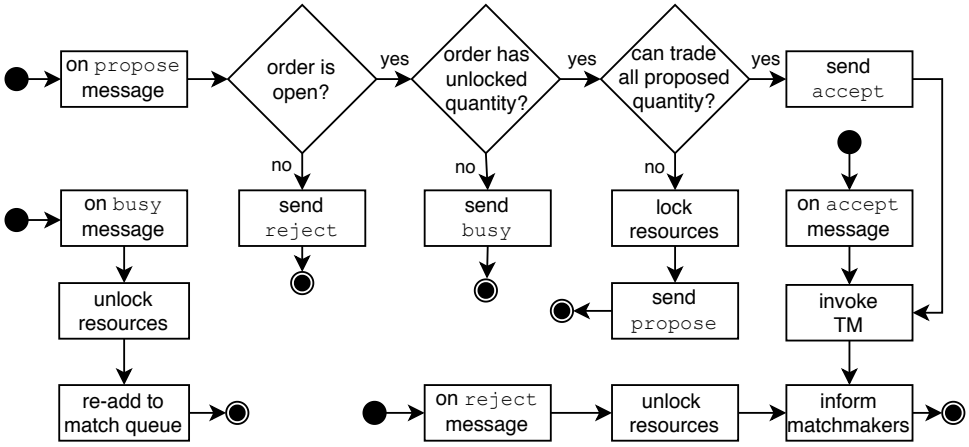


Figure 3.4: The flow diagram (in UML format) when receiving a message during the negotiation process for a specific order.

To address this issue, incoming `match` messages for a specific order are first stored in a *match queue*. When a user receives the first `match` message for one of its offers (or requests), say O, it creates a new match queue for O. Each entry in the match queue of offer (or request) O is a tuple (a, R) where R is a request (or offer) that matches with offer (or request) O. a indicates the number of failed negotiation attempts for R. The value of a is locally tracked by each user. Removing items from the match queue is first based on a (item with a lower value of a have higher priority) and is then based on the quality of the match (the user prioritizes the negotiation of matches with higher quality). The quality of a match is an application-specific metric that can be considered as the "distance" between an offer and a request, and is computed by the matching policy. An incoming `match` message can be inserted in multiple match queues, e.g., in the match queues for offers or requests with the same type and similar specifications.

Before a user starts to select items from a match queue, it waits for some duration W_{match}, which we call the *match window*. The value of W_{match} should be carefully considered: a higher value of W_{match} increases the probability of receiving more and better matches but adds to the order completion time since a user has to wait longer before starting order negotiations. Decreasing W_{match}, however, might result in missing better matches. W_{match} also depends on the link latency of the peer-to-peer network where the value of W_{match} should be increased in unreliable networks. Furthermore, W_{match} is influenced by the trading domain, e.g., passengers in a ride-hailing market can usually tolerate an additional wait time of a few seconds, whereas this increase might be unacceptable when a user quickly wants to buy some assets in response to price fluctuations in an asset market. When the match window expires, a user removes the entry (a, R) with the best quality from the match queue and initiates the order negotiation process with the user that created R.

3.4.4 Order Negotiation

In MATCH, users negotiate about their orders with other users. This negotiation approach increases resilience against malicious matchmakers since rational users choose to negotiate about the best incoming matches in order to get the best deal. When two negotiating users reach a trade agreement, both users send the agreement to the TM, upon which the trade is executed. This approach addresses question 1 in Section 3.2.2 and avoids the need for network-wide consensus.

We now elaborate on the negotiation procedure between two users. Figure 3.4 shows the control flow in UML format when a user receives specific messages during the order negotiation process. We elaborate the negotiation process between users U_1 and U_2 according to Figure 3.4. In the following, we assume that user U_1 created offer O, user U_2 created request R, and these two orders match. Now, U_2 has received a `match` message from a matchmaker, informing U_2 about matching offer O. This puts entry (a, O) in the match queue associated with R. Order negotiation, based on match queue entry (a, O), starts by U_2 locking quantity in request R. How much quantity is locked depends on the available quantities in both O and R. Specifically, U_2 proposes to trade as much available quantity as possible, given the specifications of O and R. Explicitly locking quantity in an order prevents a user from engaging in parallel negotiations for the same resources. MATCH does not enforce the locking of quantity since we assume that rational users will correctly manage their orders. After locking the quantities in R, U_2 sends a `propose` message to U_1, containing the full specifications of R, the identifier of O, and the proposed quantity to trade.

When U_1 receives a `propose` message from U_2, it first checks if its offer O, which identifier is contained in the `propose` message, is still open. If O is expired, has been cancelled, or has been completed already, U_2 immediately responds with a `reject` message, containing the reject reason. When U_2 receives the `reject` message from U_1, it unlocks the locked quantity for that negotiation and selects the next entry from the match queue of its request R. Since matchmakers might have outdated information about O (e.g., when O has been completed but the matchmaker has not been notified about this event), U_2 forwards the `reject` message to the matchmakers that informed U_2 about the match with O. Matchmakers then update the state of O accordingly when receiving a `reject` message.

If offer O is open, U_1 first determines if the incoming proposal is acceptable. This step depends on the trading domain and specifically on the application-specific data in the order. For example, a matchmaker in a ride-hailing market can establish a valid match between a passenger and driver. However, this match might be unacceptable for one of the matched parties (e.g., when the geographic separation between the parties is too large). If U_1 finds the proposal unacceptable, it sends a `reject` message to U_2.

If the proposal is acceptable, U_1 checks if it has any unlocked quantity in the offer O. If there is no unlocked quantity available for trade, U_1 responds with a `busy` message to U_2, indicating that it currently has no room for negotiation. In this situation, U_1 is already engaged in negotiations for that order with other users. Upon receiving a `busy` message, U_2 unlocks the locked quantity in R and re-adds the entry associated with the failed negotiation to the match queue of R, incrementing a by one. Re-adding this entry to the match queue will cause U_2 to initiate a negotiation with U_1 again later. To prevent a user from immediately retrying a failed negotiation, a user waits a random period (between 1 and

2 seconds) before sending out another `propose` message when processing a match queue entry with $a > 0$.

If offer O has unlocked quantity, U_1 checks whether the full proposed quantity can be traded. If so, U_1 sends an `accept` message to U_2, thereby accepting the proposal of U_2. U_1 also forwards the `accept` message to the matchmakers that proposed the match, so they can update the state of this order in their order book. If the proposed quantity is unavailable in O, U_1 makes a counter-proposal by locking as much quantity as possible in O and by sending a `proposal` message back to U_2 with this (lower) quantity. U_1 and U_2 keep sending `propose` messages with differing quantities until one of them responds with either an `accept` or a `reject` message.

It could be that one of the involved parties does not respond during a negotiation, e.g., to deliberately lock quantity in the orders of another user. To address this situation, all outgoing messages during order negotiation have a fixed *negotiation window*, indicated by W_{neg}, after which the user leaves the negotiation. When this window expires, any locked quantity for this negotiation is released and incoming messages regarding the expired negotiation are ignored.

3.4.5 Match Execution and Order Updates

Upon reaching a trade agreement between two negotiating users, it should be executed by the trading mechanism. To execute a trade agreement, one party sends the `propose` message to the TM and the other party sends the `accept` message to the TM (step ⑧ and ⑨ in Figure 3.2). Each message contains the digital signature of its creator. The TM executes the trade after having received both these messages. Next, each involved party individually inform the matchmakers that originally received their order about the match execution by sending an `update` message (step ⑩ and ⑪ in Figure 3.2). This message contains the order with an updated quantity, specifying the new interests of the order creator after the match has been executed. The `update` message contains an order with quantity 0 if it has been completed. Upon receiving an `update` message, matchmakers update the state of the changed order accordingly and remove orders that have been completed.

3.4.6 Privacy Considerations

We end our protocol description with a few notes on privacy. In MATCH, full order details are shared with matchmakers. While this is standard for application domains such as blockchain marketplaces, it is undesirable for some application domains such as financial markets. We note that the `data` field of an order in MATCH can be fully specified by the system designers. Therefore, the content of orders, and the quantity field, can be subjected to cryptographic protocols such as encryption. Only when an agreement has been reached between two parties, the order specifications can be revealed between the two interacting parties. To leverage the functionality of the match queue, however, it is desirable that incoming orders can be totally ordered.

There are some privacy-enhancing solutions for matchmaking in particular application domains, such as ride-hailing. ORide, for example, leverages homomorphic encryption to build a full protocol for privacy-preserving ride sharing [139]. We believe that, whether or not with additional research, the underlying ideas of existing solutions such as ORide can increase the privacy aspects of decentralized matchmaking.

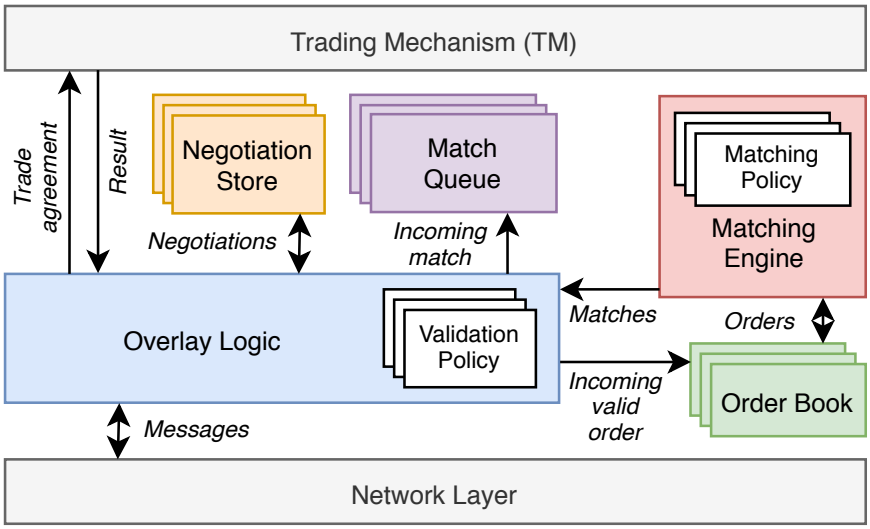


Figure 3.5: The MATCH middleware architecture.

3.5 The MATCH Middleware Architecture

We now present the MATCH middleware architecture, visualized in Figure 3.5. Each user and matchmaker deploy a single instance of the MATCH middleware as a shared runtime library. Communication with the middleware by external applications proceeds through an API, which specifications are included in our open-source implementation. We now elaborate on the components in the MATCH middleware architecture.

Network Layer. The network layer passes incoming messages to the MATCH overlay logic and routes outgoing messages to their intended destination. This layer can be implemented using any networking library with support for peer-to-peer communication and authenticated messaging.

Overlay Logic. The overlay logic processes incoming messages received by the network layer. It inserts incoming `match` messages in the appropriate match queue, discussed in Section 3.4.3. It contains policies for order *validation* which specify how the validity of an incoming order with a given type is determined. The validation policy takes into consideration the attributes in the `data` field of the order (see Listing 3.1), and checks whether they are valid with respect to a trading domain. For example, a validation policy for orders in a ride-hailing market should check whether the latitude and longitude coordinates are included and within a valid range (-90 to 90). We envision that developers share the implementation of these policies through some distribution medium (e.g., a website). To increase the trustworthiness and security of these policies, the policy implementations should be auditable and attestable by other developers and auditors. These validation policies can then downloaded by users and matchmakers that are interested in participation within a specific trading domain. We provide developers the means to create custom validation policies, enabling order validation in different trading contexts. Incoming orders deemed invalid by the validation policy are discarded and not processed.

Notation	Variable Description
W_{match}	Match window (fixed to 2 seconds)
W_{neg}	Negotiation window (fixed to 5 seconds)
m	Number of matchmakers
f	Order fanout
r	Fraction of malicious matchmakers

Table 3.1: An overview of the variables used in Section 3.6.

Order Books. Each matchmaker can host multiple order books, coloured green in Figure 3.5, which store orders with differing types. For example, MATCH can maintain an asset trading order book that stores orders buying or selling Euros for Dollars, and another ride-hailing order book that stores transportation requests and ride offers. Maintaining multiple order books is a key property of MATCH and results in a single and reusable matchmaking solution that can be deployed across different domains.

Matching Engine. Valid incoming orders are passed to the matching engine, coloured red in Figure 3.5. The matching engine attempts to match incoming offers and requests with existing requests and offers, respectively. It contains *matching policies* which predicate whether a specific offer and request match, based on the order type and specifications. For example, matchmaking in a ride-hailing market is often based on the geographic distance between a driver and a passenger. Likewise, asset orders would match if the price of an offer is equal to or lower than the price of a request. Similar to validation policies, matching policies are published on a website or on another public medium, and can be downloaded by interested matchmakers.

Negotiation Stores. To correctly process incoming messages from negotiation counterparties, MATCH requires state storage of outgoing messages during order negotiation. This state is stored in a *negotiation store*, coloured yellow in Figure 3.5. For each negotiation, a new negotiation store is created, a unique identifier is generated, and this identifier is appended to each message associated with this negotiation. Counterparties are required to include the same identifier in response messages. Incoming negotiation messages containing an unknown identifier are discarded by users. Negotiation stores time out after the negotiation window expires, on which the store and its contents are deleted.

Trading Mechanism. Negotiated trade agreements are passed to the trading mechanism that executes the trade. We consider this component external to MATCH. The trading mechanism notifies the overlay logic when the trade is executed. This notification includes one or more transaction identifiers and a boolean value indicating whether the trade was successful or not. We assume that the trading mechanism provides atomic guarantees: either the full negotiated match is executed or nothing is being executed. This guarantee is, for example, provided by smart contracts, applications that runs on top of a blockchain (also see Section 3.6.3) [128].

3.6 Experimental Evaluation

We implement the MATCH protocol and middleware in the Python 3 programming language, spanning a total of 6'511 lines of source code (SLOC), without comments. The imple-

mentation uses the `asyncio` library for asynchronous event processing. The network layer is implemented using our networking library, optimized for building peer-to-peer overlay networks and with built-in support for Network Address Translation (NAT) puncturing and authenticated messaging [63]. For efficiency, message exchange between users and matchmakers uses UDP. Order negotiation proceeds using TCP since this flow requires bilateral and reliable message exchange. All software artefacts of MATCH (source code, tests, and documentation) are available online.[1]

In Section 3.6.1 and 3.6.2, we subject the MATCH middleware to two workloads for ride-hailing and asset trading, reconstructed from real-world traces. These experiments demonstrate that MATCH maintains high matching quality, is resilient against malicious matchmakers, and is reusable across different trading domains. In Section 3.6.3, we compare our solution to matchmaking on an Ethereum blockchain and show that MATCH uses considerably less bandwidth and has superior order fulfil latencies.

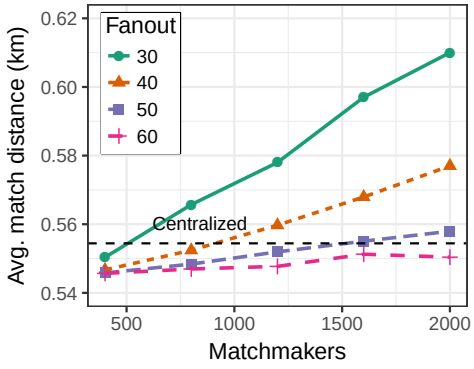
All experiments are conducted on our nation-wide university cluster, allowing us to run multiple instances of MATCH on different compute nodes [64]. It contains 48 compute nodes, each one equipped with dual 8-core E5-2630v3 CPU and 64 GB of memory, running CentOS 6. To account for network latencies, we source a distribution from the PlanetLab latency dataset and uniformly sample from it when sending messages [140]. This also accounts for runtime variability of latency present in real-world networks. Table 3.1 summarizes the variables that are used in this section. The negotiation window (W_{neg}) is fixed to five seconds, which is well above the highest observed round-trip time in the PlanetLab latency dataset. The match window (W_{match}) is fixed to two seconds. These values should be increased when MATCH is deployed in networks with higher link latency, since it then can take longer for `match` or negotiation messages to arrive.

3.6.1 Ride-hailing Experiments

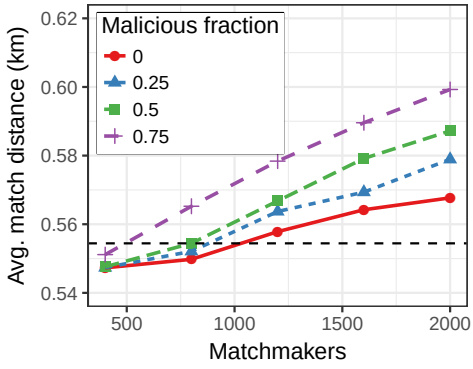
Unfair matchmaking in ride-hailing markets is a prominent threat to both drivers and passengers [71, 127]. We leverage the MATCH middleware to devise a decentralized alternative to ride-hailing platforms like Uber and Lyft. In this market, drivers perform matchmaking themselves. The first set of experiments focuses on the matching quality and fairness of MATCH in a ride-hailing environment. The used workload contains temporal information about ride offers and requests created by drivers and passengers, respectively. Each order in the workload has a quantity of one, ensuring that a ride request is matched with at most one ride offer.

Workload Specification. The workload is reconstructed from historical traces of taxi rides published by the government of New York [141]. We analyse the traces and subtract 2'100 ride offers and requests during the busiest period in 2015: November 1, 00:59:57 to November 1, 01:01:16 (datasets published after 2015 did not include geographic information on drivers and passengers). We assume a total of 1'100 drivers and 1'000 passengers, to resemble the situation where drivers are waiting idly for passengers. First, drivers indicate their willingness to transport passengers by creating offers containing their waiting location, during 55 seconds (we wait 50 milliseconds between the creation of subsequent ride offers). After this period, passengers submit requests containing their pick-up location, during almost 77 seconds. After all passengers have submitted their

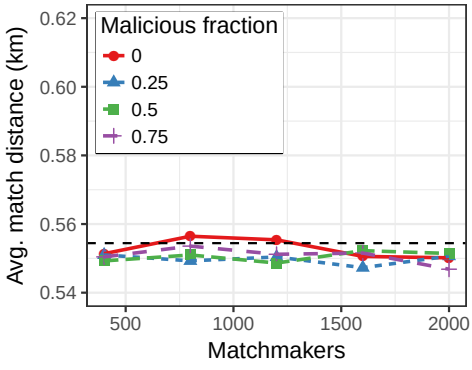
[1] See `https://github.com/Tribler/anydex-core/tree/match-middleware20`



(a) Matching quality with different fanouts f.



(b) Impact of selfish matching (f = 50).



(c) Impact of selfish matching (adaptive f).

Figure 3.6: The matching quality and impact of selfish matching when executing the ride-hailing workload in MATCH, while varying the number of matchmakers. In Figure (b) and (c), the order fanout f is either fixed (f = 50) or adaptive such that $P(R_{req} \cap R_{off} \neq \emptyset) \geq 0.95$.

request, we leave the experiment running for an additional 60 seconds, to ensure that all requests are matched with an offer. Since the workload does not provide information on the identity of individual passengers or drivers, it is assumed that each passenger creates one request throughout the experiment. This assumption does not lead to skewed results since a passenger does not create multiple ride requests within short time [139].

For this workload, we implement the matching policy such that it minimizes the distance between passengers and drivers, to reduce waiting times for passengers. Specifically, the policy computes the geographic (haversine) distance between the locations included in offers and requests. In this market, we define the matching quality as the average distance between matched passengers and drivers, which we also call the *match distance*. This quality metric is also used by related work on matchmaking in ride-hailing markets [139].

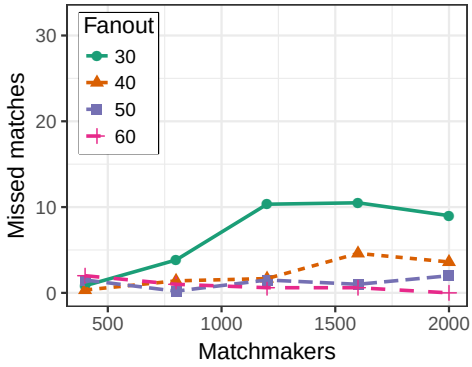
Matching Quality. We first quantify the matching quality of MATCH under the ride-hailing workload when increasing the number of matchmakers for different values of the order fanout, see Figure 3.6a. The horizontal axis shows the number of matchmakers

(m) and the vertical axis denotes the average match distance. For this experiment, all matchmakers act honest and execute the same matching algorithm (in other words, r = 0). As a baseline, we use the matching quality of a centralized matchmaking approach where a single server matches incoming orders in a FIFO manner (following the model in Figure 3.1). This centralized approach results in an average match distance of 0.544 km, and is indicated with a dashed horizontal line in the graphs of Figure 3.6. Figure 3.6a shows that the average match distance increases when there are more matchmakers under a fixed order fanout. Also, the match distance increases when the order fanout decreases. In particular, It becomes less likely that (good) matches for offers and requests are found when either the number of matchmakers increases or the order fanout decreases. The match distance increases significantly when f = 30 and m increases. E.g., for m = 2′000, the average match distance increases to 0.607 km, 9.5% higher compared to the baseline.

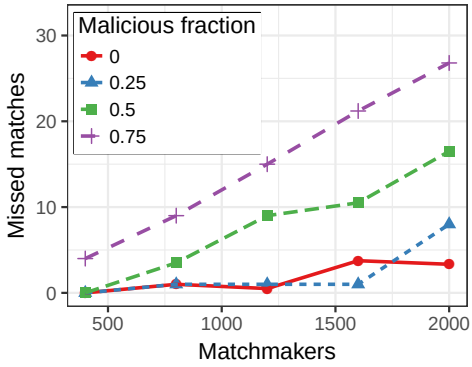
For lower values of f and m, *MATCH outperforms centralized matchmaking, in terms of matching quality.* We explain this behaviour as follows. With our ride-hailing workload, centralized matchmaking can immediately match a ride request with a ride offer. The overall match quality, however, might be improved when batching incoming ride requests, since it could have been better to assign an already-matched driver to a passenger that created its request later during the experiment. Call markets, for example, operate in batches, where incoming orders are first aggregated over time and then matched at predetermined time intervals. In MATCH, users aggregate potential matches during the match window, W_{match}, resembling this behaviour. Therefore, the matching quality in MATCH can exceed that of centralized FIFO matchmaking because of match aggregation by users, at the cost of a larger order fulfil time.

Impact of Selfish Matching. We show how selfish behaviour of malicious matchmakers impacts the matching quality. We model a malicious matchmaker as a driver that matches an incoming ride request from a passenger with its own service offer first. This captures the economic incentive of drivers to prioritize their own ride services.

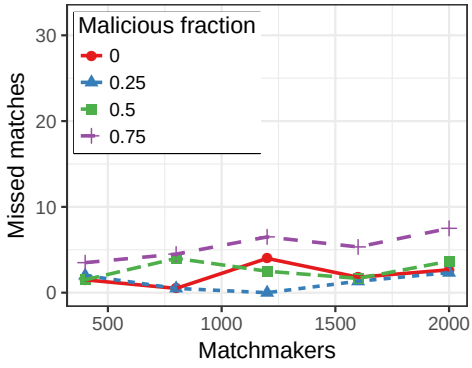
Figure 3.6b shows the average match distance under a fixed order fanout (f = 50) when increasing the number of matchmakers. We vary r, up to 75% of all matchmakers (r = 0.75). Figure 3.6b shows that increasing both m and r has a negative impact on the matching quality in MATCH. The problem is that a malicious matchmaker matches the requests of passengers with its own offer, while it likely would have established a better match if the matchmaker had been honest. Therefore, we also consider an adaptive order fanout, based on the values of both r and m. Specifically, f is fixed to the lowest integer value such that $P(R_{req} \cap R_{off} \neq \emptyset) \geq 0.95$. Figure 3.6c visualizes these results with an adaptive order fanout. The order fanout is 63 with m = 2′000 and r = 0. Formula 3.2 describes that when 50% of the matchmakers prioritize their own ride offer (r = 0.5), the order fanout increases to 78. Figure 3.6c shows that the average match distance remains largely the same, even when increasing the total number of matchmakers. These results show that in a network with 2'000 matchmakers, by increasing the order fanout by only 15, MATCH can tolerate with 50% of all matchmakers acting maliciously and still produce a matching quality that is on par with the situation where all matchmakers are honest. In practice, the exact value of r is not known a-priori and MATCH developers should therefore fix the order fanout to account for an upper bound for the malicious fraction (e.g., many BFT consensus algorithms tolerate up to $r = \frac{1}{3}$ [133]).



(a) Matching quality with different fanouts f.



(b) Impact of selfish matching (f = 50).



(c) Impact of selfish matching (adaptive f).

Figure 3.7: The matching quality and impact of selfish matching when executing the ride-hailing workload in MATCH, while varying the number of matchmakers. In Figure (b) and (c), the order fanout f is either fixed (f = 50) or adaptive such that $P(R_{req} \cap R_{off} \neq \emptyset) \geq 0.95$.

3.6.2 Asset Trading Experiments

We now evaluate MATCH with an asset trading workload. Driven by the popularity of blockchain-based assets, major peer-to-peer markets have emerged to facilitate cryptocurrency exchange between traders [142]. MATCH enables traders to perform matchmaking of orders themselves, without entrusting their orders to a market operator. Unlike our previous experiments, the workload used in our upcoming experiments involves offers and requests that be partially fulfilled and are commonly cancelled. We conduct the same matching quality and fairness experiments described in Section 3.6.1 with an asset trading workload.

To the best of our knowledge, there is no standardized definition for the matching quality of orders with partial fulfilment. Therefore, after each experiment with the asset trading workload, we determine the matching quality as follows: all orders that are not cancelled or fulfilled are added to a single order book, starting with the first order created. When adding these orders to the order book, we sum the number of matches returned

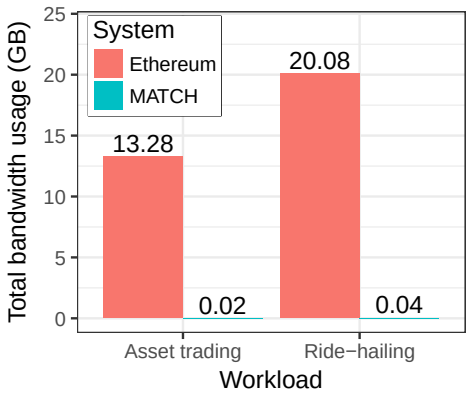
by the matching engine, which yields our matching quality. Intuitively, this definition indicates how many additional matches a central matchmaker would have found if it had knowledge of all open orders. By definition, the matching quality of centralized matchmaking is zero and therefore optimal with FIFO order matching. In the worst case, our middleware would have missed 6'450 matches, which is the situation where no matchmaker performs any matching. When running the asset trading workload, the matching engine matches orders according to the *price-time* matching policy [44].

Workload Specification. The asset trading workload contains buy and sell orders that have been published on the blockchain ledger of BitShares [143]. BitShares is a blockchain-based decentralized exchange where users can create, issue and trade digital assets. New orders are submitted to dedicated validator nodes, which include incoming orders in a block on the blockchain. We have analysed the entire BitShares blockchain since its inception and extracted all buy and sell orders, and order cancellations. To generate load on our system, we determined when most orders were created for five minutes. The result is a workload with 942 order cancellations, 12'253 offers and 3'342 requests involving 121 unique asset types. On average, traders create 52 new orders every second. Since our dataset does not contain temporal information on order creation and cancellation, we assume that each order is uniformly created in the time interval between the last block and the block that contains this specific order. We believe this approximates the actual creation timestamp of the order and that this does not skew the experiment results. Again, there is a 60 seconds experiment cool down period after the creation of the last order.

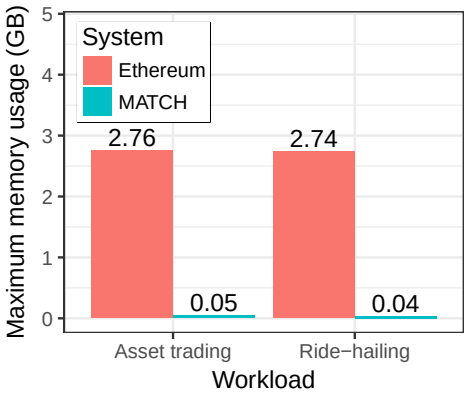
Matching Quality. Figure 3.7a shows the matching quality while varying the number of matchmakers and order fanout. By definition, a centralized approach to matchmaking would not miss any match. Similar to the matching quality experiment with the ride-hailing workload (see Figure 3.6a), matching quality decreases when there are more matchmakers and the order fanout is static. It particularly interesting to observe how even a relative low order fanout of 30 results in less than ten missed matches on average (only 0.61% of the maximum number of missed matches). Further analysis of the workload reveals that various users create multiple orders for the same asset pair within short time. Therefore, `match` messages for such orders received are inserted in multiple match queues, and thus re-used. Users creating multiple smaller orders with similar specifications are reaching more matchmakers and can potentially negotiate better matches.

Impact of Selfish Matching. We demonstrate the effect of malicious matchmakers on the matching quality in our asset trading workload, both with a fixed and adaptive order fanout. Under the asset trading workload, we model a malicious matchmaker as a node that purposefully does not inform the party behind an incoming order about the match with the best price. Essentially, a malicious matchmaker "hides" order book entries from the order creator.

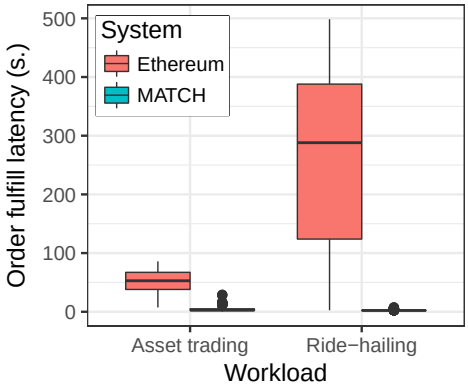
Figure 3.7b shows the number of missed matches with $f = 50$ when increasing m and varying the r. More matchmakers negatively impacts the matching quality, although its effect is relatively minor. In particular, even with $r = 0.5$ and $m = 2'000$, MATCH only misses less than ten matches on average. We repeat the experiment while adapting the order fanout such that $P(R_{req} \cap R_{off} \neq \emptyset) \geq 0.95$, see Figure 3.7c. It shows that for all settings, MATCH only misses under ten matches on average.



(a) Total bandwidth usage.



(b) Maximum memory usage.



(c) Order fulfil latency.

Figure 3.8: The total bandwidth usage and the distribution of order fulfil latencies of on-chain matchmaking on Ethereum and in MATCH, under the ride-hailing and asset trading workloads.

3.6.3 Comparison with On-chain Matchmaking

We compare the bandwidth usage and order fulfil latencies of MATCH with that of matchmaking on an Ethereum blockchain, using both the ride-hailing and asset trading workloads. Ethereum is the most mature blockchain platform that enables the execution of generic-purpose smart contracts, and is the most used platform to deploy smart contracts in general [2].

We set up a private Ethereum network consisting of 400 instances running `geth`, an Ethereum client written in Go.[2] Ethereum uses a Proof-of-Work consensus mechanism in which participants, also called miners, compete to include transactions on the blockchain. Specifically, each miner continuously solves an algorithmic puzzle and the first miner to find a valid solution to the puzzle, can extend the blockchain with one block with transac-

[2]See `https://github.com/ethereum/go-ethereum`

tions. We fix `geth` such that each instance mines on at most one CPU core. We fix the gas limit (indicating the maximum amount of computation that can be done within a block) to 10'000'000, in line with the public Ethereum network. To accurately compare the performance of MATCH and Ethereum, we run both workloads in MATCH with 400 instances, and adjust our workload accordingly. We fix $m = 400$, $f = 30$, and $r = 0$. Since a smart contract enforces the correct execution of a particular matching policy, we run MATCH with 400 honest matchmakers.

Smart Contracts. For both workloads, we implement a smart contract in the Solidity programming language. The smart contract for the ride-hailing workload maintains two lists containing open offers and requests. Submission of a new ride offer and request is done by issuing a transaction with geographic information that invokes the `ride_offer` and `ride_request` methods in the smart contract, respectively. Invocation of these methods triggers a loop through the list of active offers or request, and finds the matching order that minimizes the distance between the passenger and driver. The algorithmic complexity when matching a single offer and request is $O(n)$ where n is the number of open requests and offers, respectively. To avoid computationally expensive trigonometry operations when computing the haversine distance, latitude and longitude coordinates are projected to Universal Transverse Mercator (UTM) coordinates and the Manhattan distance is used as a norm in the smart contract. This results in an accuracy loss of only 0.35%.

For the asset trading workload, we adopt an existing and deployed order book implementation.[3] This smart contract implements a market to trade digital tokens that reside on the Ethereum blockchain. Orders are bundled in a limit order book and organized in distinct price levels. This allows for a strategic search for order matches and avoids the need for a full linear scan through all offers and requests. This order book organization is predominantly used by financial exchanges. To quantify the overhead of order matchmaking, we remove the operation that transfers token ownership after matching from the smart contract but leave the implemented price-time matchmaking logic intact.

Bandwidth Usage. We measure the aggregated bandwidth usage of all instance of MATCH and Ethereum, see Figure 3.8a. Ethereum requires over 20 GB of network traffic for the ride-hailing workload. In comparison, MATCH uses dramatically less bandwidth compared to Ethereum-based matchmaking. MATCH only requires 41.6 MB of aggregate network traffic under the ride-haling workload, and 20.7 MB under the asset trading workload. The high bandwidth usage of Ethereum is a direct consequence of the full replication of state. Specifically, each transaction and block is disseminated to all active Ethereum instances, resulting in a significant amount of network traffic.

Memory Usage. We measure the maximum memory usage of all running MATCH and Ethereum instances, see Figure 3.8b. At peak, Ethereum requires 2.8 GB of memory to store pending transactions. This is partially due to the specifications of the mining algorithm in Ethereum, which requires the storage of a 1 GB graph in memory. Furthermore, each Ethereum instance maintains all unconfirmed transactions and recent blocks in memory. In comparison, MATCH only requires around 50 MB of memory for both workloads, most of which is overhead of the Python interpreter.

Order Fulfil Latencies. In Figure 3.8c, we show the time it takes to complete orders in MATCH and Ethereum, for both workloads. Specifically, this is the time between or-

[3] See `https://github.com/makerdao/maker-otc/tree/master/src`

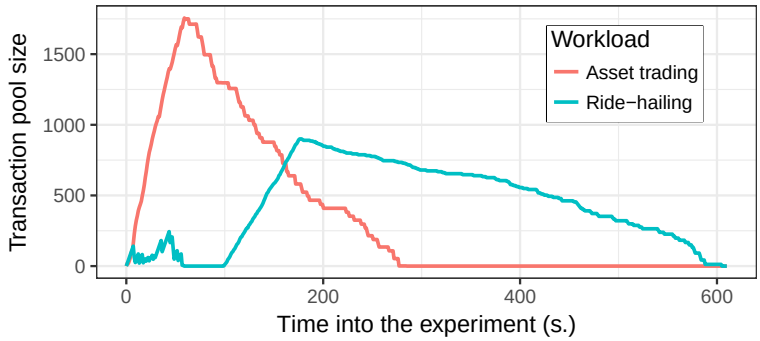


Figure 3.9: The size of the transaction pool during our Ethereum experiments, under the ride-hailing and asset trading workloads.

der creation and order fulfilment. For the ride-hailing workload, we only consider the completion time of requests made by passengers, since drivers are waiting for incoming requests. Figure 3.8c shows that the average order completion time of MATCH under the ride-hailing workload is 2.46 seconds and increases under the asset trading workload to 5.02 seconds. Since users aggregate `match` messages during the match window, the order fulfil latency in MATCH is at least W_{match} (which is fixed to two seconds in our experiments). In comparison, the average order fulfil latency in Ethereum is 258.2 and 53.6 seconds under the ride-hailing and asset trading workload, respectively. We argue that in a ride-hailing market, the latencies experienced when performing matchmaking on an Ethereum blockchain would be unacceptable for both passengers and drivers.

Ethereum Transaction Pool. To further analyse the large differences in order fulfil latencies between MATCH and Ethereum, we visualize the size of the Ethereum transaction pool (as maintained by a single Ethereum instance) during the experiment in Figure 3.9. This figure shows the time into the experiment on the horizontal axis and the number of transactions in the pool on the vertical axis. The transaction pool contains transactions that are not yet included in a block on the blockchain by a miner. Note how Ethereum becomes congested under the asset trading workload quickly after starting the experiment. Around 70 seconds into the asset trading experiment, the transaction pool contains 1'713 unconfirmed transactions that are buying or selling assets. 285 seconds after the start of this experiment, all orders are included in a block on the Ethereum blockchain.

The ride hailing experiment starts by drivers submitting ride offers to Ethereum instances. All ride offers are included on the Ethereum blockchain after 58 seconds into the experiment. Passengers start to submit ride requests 100 seconds into the experiment. Similar to the asset trading workload, the size increase of the Ethereum transaction pool shows that the blockchain is unable to handle the load of incoming transaction, causing congestion. 180 seconds into the experiment, the number of unconfirmed transactions decreases. Further inspection of the blockchain reveals that only around ten transactions with a ride request are included in each block. We explain this behaviour as follows. In Ethereum, the sum of gas usage of all transactions in a block cannot exceed 10'000'000

gas. The gas cost of matching ride requests scales with the number of open ride offers and decreases during the experiment since there are fewer offers to compare with. Note how after 500 seconds into the ride-hailing experiment, the number of unconfirmed transactions decreases quickly.

3.7 Related Work

Matchmaking (or brokering) is a core concept in publish/subscribe (Pub/Sub) architectures. In centralized Pub/Sub architectures, a single server brokers incoming messages between publishers and subscribers [144]. Decentralized approaches either flood events through the entire network, or route these events based on their topic or content [145, 146]. In contrast to Pub/Sub systems, MATCH does not ensure that events (orders) are eventually delivered to all subscribers (matchmakers).

Resource allocation, the assignment of compute resources to incoming jobs, also requires matchmaking [147]. Most work on resource allocation aims to find an optimal assignment between resources and jobs, whereas our work focuses on establishing any match [148]. In this context, we identify two matchmaking approaches described in literature. The first approach is to use market mechanisms that coordinate the matchmaking process, e.g., by a continuous double auction mechanism [149–151]. Market-based matching approaches increase allocation efficiency but compromise on computational efficiency since it requires synchronization mechanisms. The second approach to matchmaking is to deploy one or more dedicated (centralized) brokers [152–154].

Motivated by the scalability and load balancing issues of centralized matchmaking, various researchers explored the usage of multiple, independent matchmakers [155–158]. Matchmakers usually operate within their own administrative domain, acting as a broker for a specific set of nodes. The work of Shafran et al. evaluates a distributed matchmaking model where orders are cached by intermediate agents [159]. Their work, however, only considers one-to-one matching.

With the proliferation of blockchain-based tokens, there have been various proposals for matchmaking architectures that complement decentralized exchanges. These architectures aim to avoid transaction fees and expensive on-chain matchmaking by relying on an off-chain order matching service, and on-chain order execution. IDEX, an Ethereum-based decentralized exchange, uses a centralized server for order matchmaking [35]. In AirSwap, *indexers* mediate trade between makers, nodes who create an order, and takers, who fulfil existing orders [160]. In contrast to MATCH, a user can only send a new order to a single AirSwap indexer. The 0x protocol uses a similar matchmaking approach since traders send a new order to exactly one matchmaker [46]. The Loopring protocol is similar to the decentralized matching model of MATCH since traders submit orders to one or more *relay nodes* [50]. Their protocol description, however, lacks details on the dissemination strategy of orders to matchmakers.

Auctions are related to order matchmaking since they also provide a mechanism to allocate resources from sellers to buyers. PeerMart is a decentralized auction mechanism that uses sets of distributed brokers [161]. There have been various proposals to run Ethereum-based auctions while preserving the privacy of bidders [162, 163]. Yet, auctions and order matchmaking are different economic primitives with differing goals. In contrast to order matchmaking, time frames (and time limitations) are critical in auctions. Furthermore, auc-

tions have higher security requirements and need (time-bounded) coordination amongst participants, e.g., to determine the winning bidder.

3.8 Conclusions

We have presented MATCH, a decentralized middleware for fair matchmaking in peer-to-peer markets. Our work addresses fairness concerns associated with the use of in-house, proprietary solutions controlled by a market operator. In the MATCH protocol, users send new orders to a small, random subset of matchmakers, which inform users about potential matches. Users then engage in peer-to-peer negotiation about matches with other users. This approach makes MATCH resilient against matchmakers who deviate from a standard matching policy. We have devised the MATCH middleware architecture, suitable for deployment in a WAN environment. We have experimentally proven resistance against malicious matchmakers in a ride-hailing and asset trading domain, showing that MATCH still establishes high-quality matches. Our comparison experiments have showed that the resource usage of MATCH is considerably lower compared to that of matchmaking on an Ethereum blockchain.

4

XChange: A Universal Mechanism for Asset Exchange between Permissioned Blockchains

Permissioned blockchains are increasingly being used as a solution to record transactions between companies. Several use cases that leverage permissioned blockchains focus on the representation and management of real-world assets. Since the amount of incompatible blockchains is quickly growing, there is an increasing need for a universal mechanism to exchange, or trade, digital assets between these isolated platforms. There currently is no universal *mechanism for inter-blockchain asset exchange without a requirement for trusted authorities that coordinate the trade.*

In this chapter we address this shortcoming and present XChange, a universal mechanism for asset exchange between permissioned blockchains. To achieve universality and to avoid trusted authorities that coordinate a trade, XChange does not provide atomic guarantees but leverages risk mitigation strategies to reduce value at stake. Our mechanism records the specifications and progression of each trade within records on a distributed log. XChange reduces the economic gains of adversaries by bounding the total amount of fraud they can commit at any time. After having committed fraud, an adversary is forced to finish its ongoing trades before it can engage in new trades.

We first present a four-phased protocol that coordinates an asset exchange between two traders. We then outline how trade records can be stored on TrustChain, which is a lightweight distributed ledger specifically built for the tamper-proof storage of data elements. We implement XChange and conduct experiments. Our experiments demonstrate that XChange is capable of reducing the economic gains of adversaries by more than 99.9% when replaying a real-world trading dataset. A deployment on low-resource devices reveals that the latency added to a trade by XChange is only 493 milliseconds. Finally, our scalability evaluation shows that XChange achieves over 1'000 trades per second and that its throughput, in terms of trades per second, scales linearly with the system load.

4.1 Introduction

Bitcoin, introduced in 2008, has revolutionized the field of digital currencies by demonstrating that it is possible to devise a secure cash system without a bank [1]. The goal of Bitcoin is to realize a payment system through the secure management of a native currency on a distributed ledger. The creation of this currency is controlled by miners participating in a voluntary process known as mining. The collective efforts of miners ensure the security of Bitcoin and prevent illegitimate coin creation. Miners invest computational power to include valid transactions on the blockchain, which is a tamper-proof distributed ledger that consists of blocks. One of the compelling features of a blockchain is the ability to securely record and validate user-issued transactions without trusted authorities, even in the presence of mutual distrust between participants.

Participation in many deployed blockchains is open for everyone and does not require the explicit approval from authorities unlike traditional banking systems. Even though blockchain technology provides the means to maintain a distributed ledger without trusted authorities, open enrolment is not required for many industrial use cases, or is even undesirable. For instance, when two companies leverage blockchain technology to securely record their transactions, read and write access to the distributed ledger is most likely limited to a few selected employees or operators. Over the past few years, there has been a sharp increase in the development and deployment of private, or *permissioned* blockchains [164–166]. In contrast to a public blockchain like Bitcoin, membership in a permissioned blockchain is managed by an authority that approves the participation of each peer. The identity under which a peer operates is linked to a real-world persona, which reduces the likelihood of Byzantine behaviour and network threats like the Sybil Attack [101]. Permissioned blockchains usually adopt a classical consensus model designed for networks with static membership, e.g., Practical Byzantine Fault Tolerance (PBFT) [133]. Considerable efforts in permissioned blockchains have been made by projects such as Hyperledger Fabric [164], R3 Corda [32], Quorum [167] and BigchainDB [168]. Permissioned blockchains have the potential to increase the efficiency of traditional business processes in industries like logistics, energy management and trade supply chains [166].

Several use cases that record transactions on a permissioned blockchain revolve around the representation and management of real-world assets on a distributed ledger [169]. Advancements in blockchain technology have resulted in numerous platforms on which companies can issue and manage digital assets. There currently is a proliferation of different types of assets, fragmented across many blockchain implementations [170]. In public blockchains, almost 200'000 different assets are being managed on the Ethereum blockchain only.[1] A recent Forbes report reveals that at least 50 major companies, each valued at least at $1 billion, are exploring blockchain technology for asset management and trading [171]. As industry's adoption of blockchain technology is increasing, a similar asset proliferation will occur with permissioned blockchains. Unfortunately, there is no *universal* mechanism to exchange (trade) assets between isolated distributed ledgers without the involvement of a trusted third party. Research and developments in distributed ledger technology mostly focus on the deployment of new domain-specific blockchains, whereas interoperability issues are mostly ignored [172, 173]. In particular, there is a lack

[1] See `https://etherscan.io/tokens`

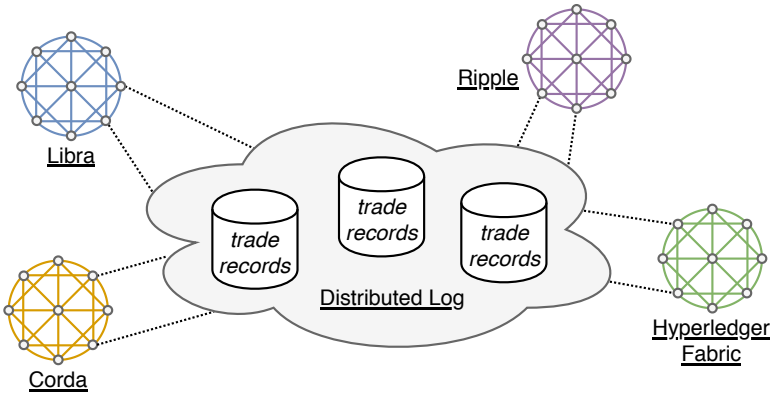


Figure 4.1: XChange coordinates the asset exchange between permissioned blockchains by storing trade records in a distributed log. This enables traders to detect if a party has committed fraud during an ongoing trade.

of research on the interoperability of permissioned, industry-grade blockchains [174, 175]. Interoperability concerns are particularly relevant when leveraging distributed ledgers for trading, as a single trade consignment can involve various isolated blockchains [176]. Given the inevitable growth of permissioned blockchain platforms, we argue that a universal mechanism for asset exchange between these platforms is a growing necessity.

We present *XChange*, a universal mechanism for asset exchange, or *trade*, between permissioned blockchains.[2] XChange coordinates trade between separate permissioned blockchains by storing trade records in a distributed log, also see Figure 4.1. Our solution is independent of the technical characteristics of the involved blockchains and does not require modifications to blockchain applications that are already operational. An asset exchange in XChange proceeds through a sequence of alternating, unilateral asset transfer operations (payments) between two parties. This is comparable to how many electronic markets (e.g., eBay) operate, where a party only initiates a payment back to the counterparty after having received a payment first. Sequential payments, however, introduce a risk of losing economic value to the other party, since the other party is now able to "steal" assets during a trade [177]. This fraud is called *counterparty fraud* and it is a severe concern in many electronic marketplaces that facilitate peer-to-peer trading [178]. For this reason, we argue that any asset trading mechanism must either prevent counterparty fraud or punish a participant that has committed this fraud upon its detection.

To address counterparty fraud, blockchain-based asset exchange often provides *atomic* guarantees. Atomicity in this context implies that a trade either exchanges all assets between involved parties or exchanges nothing. We find that the security of existing trade solutions either (1) relies on (semi-)trusted authorities to ensure that assets are securely exchanged, or (2) relies on the availability of specialized transactions by the blockchains that manage the assets being traded. Relying on authorities is the standard approach when trading assets managed by public blockchains, e.g., by using the services of a cryptocurrency exchange. In a permissioned setting, however, this approach requires the participation

[2]We use the terms "exchange" and "trade" interchangeably in this work.

of these intermediaries in the involved blockchains, which is not always allowed by their network operators. Asset exchange mechanisms that depend on specialized transactions, e.g., atomic swaps [54], are not universal enough to support asset exchange between any pair of permissioned blockchains.

In contrast to existing solutions, XChange particularly focuses on the *detection* of counterparty fraud. We argue that the detection of counterparty fraud during a trade is sufficient, since misbehaviour can always be traced back to a real-world identity, and optionally be punished by an external authority. To detect counterparty fraud, XChange requires traders to append tamper-proof trade records to a distributed log. By recording the initiation of each trade, conducted payments, and the completion of a trade, participants can detect if a malicious trader has committed fraud and then refrain from trading with that party.

XChange does not provide atomic trade guarantees; however, it bounds the economic gains of adversarial parties by introducing two risk mitigation strategies. First, XChange allows a trade to gradually complete through multiple, smaller payments. We refer to this technique as *incremental settlement*. With incremental settlement, traders themselves decide how much risk they are willing to take, and specify how much economic value they put at stake. Our second risk mitigation strategy is to bound the value that traders are entrusted with during ongoing trades. This bound is decided by traders themselves and enables a trader to still be engaged in multiple lower-risk trades. XChange forces an adversarial party to finish its ongoing trades first before it can engage in other high-valued trades. We prove that this approach bounds the economic gains of adversaries. Since XChange assumes static membership through well-defined identities, it prevents a situation where a participant that has committed counterparty fraud can re-joins the network under a new digital identity and commit fraud again (the whitewashing attack [179]).

In this work we first present and classify existing mechanisms for cross-blockchain asset exchange. We then outline our solution and describe the XChange protocol. We deploy XChange using a tamper-proof, distributed log with low overhead, a technology that pre-dates Bitcoin [96]. Specifically, we leverage an existing solution, TrustChain, that is built for the secure logging and accounting of generic data elements [98]. Our experiments with real-world trading data reveal that our risk mitigation strategies can reduce fraud gains by 99.9%. By conducting a trade between two Raspberry Pis, we quantify that the added latency by XChange is only 493 milliseconds. Additional experiments on our compute cluster reveal that XChange can handle over 1'000 trades per second and that its throughput scales linearly with the system load.

The main contribution of this work is five-fold:

1. We present the XChange *trading protocol* which specifies how assets are exchanged between permissioned blockchains by storing trade records in a distributed log (Section 4.5).

2. We devise two *risk mitigation strategies* that lower the risk for traders and bound the economic gains of adversaries committing counterparty fraud.

3. We improve TrustChain, a tamper-proof, distributed log used by XChange. Our improvements enable concurrent transactions and increase scalability (Section 4.7).

4. We provide a functional, open source *implementation* of the XChange trading protocol (Section 4.8.1).

5. We present *experimentation* around the security, resource usage and scalability of XChange, conducted on multiple low-resource devices and our compute cluster (Section 4.8.2 – 4.8.4).

4.2 Related Work and Problem Description

Achieving interoperability between blockchains is a challenging problem and remains largely unsolved [175, 180, 181]. Most research in this direction considers cross-chain interactions between permissionless blockchains [177]. There is little research on how to achieve interoperability between permissioned blockchains, even though this is also a concern in private environments. We first discuss existing solutions that address asset exchange between different blockchains, ranging from approaches that rely on a central authority to trust-less trading mechanisms using specialized transactions or intermediate blockchains. Based on our findings, we then formulate the requirements for our asset exchange mechanism.

4.2.1 Central Authorities

A common approach to exchange blockchain-based assets is by using the services of a central authority. A trade using a central authority completes as follows: two parties that agree on a trade transfer the assets for sale to one of the wallets owned by the authority. When this intermediary has received both assets, it finishes the exchange by transferring the appropriate assets to the other party. In this approach, the authority holds (temporary) ownership of the assets to be traded. Relying on a central authority removes counterparty risk for the trading parties, but it requires both parties to have faith that the intermediary does not default or compromise their assets.

Trade through a central authority can facilitate value exchange between an extensive range of different blockchains, as long as the intermediary maintains wallets on the involved blockchains and can issue transactions in these systems to transfer the assets. This is usually not an issue in permissionless blockchains since anyone can create accounts or wallets by generating a new cryptographic key pair. Centralized cryptocurrency exchanges often facilitate asset trading across numerous permissionless blockchains. Some cryptocurrency exchanges process transactions worth millions of dollars in total daily.[3] In a permissioned blockchain environment, however, a central authority coordinating an asset exchange requires explicit approval from the operator to read and write transactions on the involved distributed ledgers. Allowing new parties in a permissioned blockchain might be undesirable by operators since it introduces additional legal and operational risks.

There have been various efforts to mitigate the trust issues surrounding centralized exchanges and trusted authorities while still maintaining a centralized infrastructure. TEX is a centralized exchange that uses an off-chain settlement solution for trust-less asset trade [182]. TEX is resilient against the front-running attack where insider information is exploited to gain a financial advantage while trading. Tesseract leverages trusted hardware, e.g., Intel SGX, to build a secure cryptocurrency exchange that also addresses the

[3]See `https://coinmarketcap.com/rankings/exchanges`

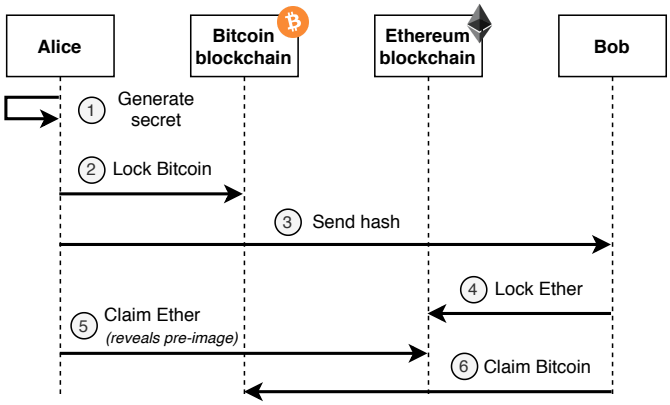


Figure 4.2: Sequence diagram of a successful HTLC-based atomic swap between Alice and Bob.

front-running attack [142]. The Arwen trading protocol is another protocol to securely trade cryptocurrencies through a centralized exchange without giving up ownership of the assets to the exchange [183].

4.2.2 Atomic Swaps

The *atomic swap* is a protocol that is commonly used to exchange assets between different blockchains, without need for a central authority [53]. This protocol enable two parties to exchange blockchain-based assets in an atomic manner: the asset exchange either completes or fails for both parties at any given time.[4] Atomic swaps eliminate the risk of losing assets to an adversarial trader during the exchange. The main idea is that trading users lock their assets in a specialized transaction on the blockchain in such a way that no single party can claim both locked assets. This is achieved with *Hash-Timelock Contracts* (HTLCs), a special transaction that leverages hash locks and time locks. A hash lock is a restriction that prevents the transfer of assets until the pre-image of a provided hash is revealed. A time lock is a primitive that locks assets until a specific time. They prevent the assets being traded from being locked up indefinitely during an atomic swap. This time lock should be well above the block confirmation time of the underlying blockchain to prevent the loss of assets during a blockchain reorganization. In practice, the duration of the time lock is often fixed to several hours.

We further explain the atomic swap by considering a trade with Bitcoin and Ether (the native token of the Ethereum blockchain). As a reminder, we repeat below the steps of the atomic swap process we described in Section 1.3.3. Figure 4.2 visualizes an atomic swap between two parties, Alice and Bob, where Alice sells her Bitcoin in return for Ether. The basic atomic swap, described by Tier Nolan [54], consists of the following six steps:

Step 1. Alice generates a secret value s and computes $H(s)$, where $H(\cdot)$ is a secure hash function.

[4]We remark that the atomicity of the atomic swap protocol depends on the security of the underlying blockchains. If one of the blockchains is compromised by adversaries, atomicity during asset exchange cannot be guaranteed and one of the parties can lose its funds to the counterparty.

Step 2. Alice submits a hash-timelock transaction T_1 to the Bitcoin blockchain, locking her Bitcoin and using $H(s)$ for the hash lock. A party can claim the Bitcoin held by T_1 with another transaction that provides s, within a specific time duration.

Step 3. Alice sends $H(s)$ to Bob using any communication medium.

Step 4. Bob submits a hash-timelock transaction T_2 to the Ethereum blockchain, locking his Bitcoin and also using $H(s)$ for the hash lock.

Step 5. Alice claims the Bobs' Ether locked in T_2 by submitting a transaction, T_3, to the Ethereum blockchain, containing s. T_3 unlocks the hash-lock in T_2. This reveals pre-image s to Bob.

Step 6. Bob now claims Alice's Bitcoin locked in T_1 by submitting a transaction, T_4, to the Bitcoin blockchain, containing s. The asset exchange is now complete.

The above protocol requires a total of four transactions. Note how Alice is not able to claim her assets without providing the opportunity for Bob to claim his assets.

Atomic swaps enable asset exchange between a wide range of blockchains. Even though they are an interesting proposition for cross-chain asset trade, we describe three deficiencies of this technique. First, atomic swaps can only be used when trading assets between distributed ledgers with support for specific programming constructs, such as time-locked and hash-locked transactions. Both blockchains are also required to support the same hashing algorithm. Second, atomic swaps require traders to lock their assets using a hash-timelock transaction. This enables a Denial-of-Service attack where a party can intentionally retain the assets of a counterparty, denying the counterparty from using the locked assets for other purposes. Third, atomic swaps can be unfair for one of the parties since the swap initiator has a time window after both parties have locked their assets, during which it can decide to abort the swap [184]. This window enables price speculation by the swap initiator by keeping the assets of the other party locked until the asset price goes in the favour of the initiator.

4.2.3 Notary Schemes

Notary schemes are another solution for asset exchange where the approval by a group of credible nodes (notaries) is required to perform some operation. Notary schemes aim to partially alleviate the trust issues arising when relying on a central authority through the approval by a group of semi-trusted notaries instead. These notaries reach consensus on the occurrence of particular events, e.g., on the inclusion of a transaction on a distributed ledger. Compared to an asset exchange through a central authority, notary schemes assume a weaker trust model and can often withstand adversarial behaviour of a fraction of the notaries.

AgentChain is an asset exchange system that is based on multi-signature schemes [185]. Each user can act as a trading operator, which together form trading groups. Assets are locked in a multi-signature wallet that requires a multi-signature to unlock. Users can choose to trade within a specific trading group, e.g., based on the reputation of the trading group. If a trading group acts malicious, a user can upload evidence to the blockchain upon which all assets managed by that trading group are transferred back to users.

An earlier version of the Interledger protocol, ILPv1, used intermediate notaries (also called *connectors*) to conduct payments across different ledgers [59]. These payments are realized through conditional payments and are coordinated by a different group of con-

nectors for every involved ledger. Interledger uses payment paths where additional intermediate platforms and their connectors are used to exchange assets between ledgers that do not have a direct connection. Only when a particular condition is met, the payment is conducted.

4.2.4 Blockchain Bridges

Another approach to cross-chain trade uses bridging techniques, where an intermediate blockchain mediates asset exchange between different blockchains. Most bridging approaches execute the atomic swap protocol for the exchange process of assets but provide additional primitives and interoperability features for communication between blockchains.

Blocknet is a platform for inter-blockchain routing and facilitates the exchange of cryptocurrencies between blockchains [186]. Blocknet consists of two main components: XBridge and XRouter. XBridge is a decentralized protocol that coordinates atomic swaps between permissioned and permissionless blockchains. XRouter provides a peer-to-peer overlay network consisting of clients running the SPV protocol, therefore avoiding the need to download the full blockchain to verify the inclusion of particular transactions. Blocknet secures its transactions through a Proof-of-Stake consensus protocol. Furthermore, Blocknet provides a decentralized exchange where traders can indicate their trade interests through orders. A blockchain connected to Blocknet requires the implementation of time-locked transactions.

ARK is a platform for cross-chain asset exchange that shares similarities with Blocknet [187]. ARK enables users to build custom blockchains (a "BridgeChain") that is powered by the ARK blockchain. To facilitate asset exchange between different blockchains, ARK acts as an intermediate blockchain in the trade process. The latter is achieved through the smart bridge protocol, relying on atomic swaps to exchange value across chains. The ARK blockchain achieves transaction security through a Delegated Proof-of-Stake (dPoS) consensus algorithm, where stakeholders vote for a small committee that appends blocks to the ARK blockchain.

The Proof-of-Authority (POA) blockchain is an Ethereum-based permissioned blockchain that provides several tools for interoperability [188]. The POA blockchain is secured by the Proof-of-Authority consensus mechanism, where validating nodes stake their reputation to secure the blockchain. The TokenBridge protocol enables users to not only exchanges assets between Ethereum-based platforms, but also facilitates arbitrary data transfer.

4.2.5 Sidechains

Sidechains provide the means to exchange assets between blockchains that share similarities, e.g., that run a particular consensus algorithm [189, 190]. In essence, a sidechain is a blockchain that is attached to a parent chain. With a two-way pegged sidechain, assets residing on the parent chain can securely be moved to the sidechain and vice versa. These transfers lock the assets on one chain and re-create them on the connected sidechain or parent chain. A related scheme is federated pegged sidechains [191]. In a federated pegged sidechains, assets moving to another chain are controlled by a group of notaries, making this approach comparable to notary-based solutions (see Section 4.2.3). Liquid is

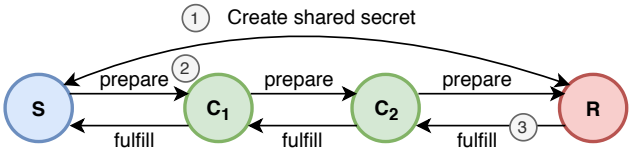


Figure 4.3: A successful ILPv4 payment from a sender S to a receiver R, using two connectors C_1 and C_2.

a deployed sidechain to the Bitcoin blockchain and can be used to quickly trade Bitcoin-derived currencies [191].

4.2.6 Internet-of-Blockchains

We now describe two solutions that aim to devise an "Internet-of-Blockchains", where a single blockchain controls many sub-chains. The Cosmos project, introduced by the Interchain Foundation, builds a network of heterogeneous blockchains that can seamlessly interact with each other [192]. The Cosmos Hub is the leading blockchain that connects many other blockchains, called zones. Each zone can have its own governance rules and is secured using the Tendermint BFT consensus protocol. Tokens can quickly be exchanged between the Hub and zones using the Inter-Blockchain Communication (IBC) protocol. To interact with blockchains external to Cosmos, there is a particular zone, called a *bridging zone*. The bridging zone keeps track of transactions and blocks persisted on external blockchains, e.g., Ethereum.

The system architecture of Polkadot, introduced by Gavin Wood, is similar to Cosmos [193]. Polkadot introduces a single relay chain that is responsible for the coordination of one or more parachains. Polkadot secures its chains through a custom consensus algorithm, inspired by Tendermint [194] and HoneyBadger [195]. Compared to Cosmos, Polkadot aims for a more generic message-passing algorithm between parachains that can not only transfer value.

Both Cosmos and Polkadot can facilitate the effortless exchange of assets between zones or parachains. However, they have limited capabilities for interaction with external blockchains. To benefit from the advantages that Cosmos and Polkadot provide, all involved companies must fully commit to the same blockchain platform, which is hard to achieve in practice. Therefore, the advantage of Internet-of-Blockchains is questionable for industrial use cases, and a less demanding approach might be preferred when trading assets.

4.2.7 The Interledger Protocol V4 (ILPv4)

The Interledger Protocol V4 (ILPv4) is a protocol for conducting payments between different ledgers [196]. Although the protocol primarily resolves around one-way asset transfers, it can also be used to exchange assets between different ledgers. ILPv4 maintains a peer-to-peer payment network consisting of different connectors that can transfer value across heterogeneous networks within ILP packets. In comparison to ILPv1 (discussed in Section 4.2.3), ILPv4 is designed around the fast transfer of low-valued payments. ILPv4 drops the requirement for ledger-based payments since they can be slow to complete and can lead to capital retention, similar to atomic swaps. A sender and a connector are as-

sumed to have funds on some shared network, e.g., they can maintain a unidirectional or bidirectional payment channel if an appropriate blockchain is used.

An ILPv4 payment between a sender *S* and a receiver *R* using two connectors proceeds as visualized in Figure 4.3. First, *S* and *R* create a shared secret, which will act as the condition for the payment (step ①). Then, *S* will prepare a `prepare` packet that contains the details of the upcoming payment and the details of the agreed-upon condition (step ②). This packet is sent to an available connector, which forwards the packet to subsequent connectors until the packet reaches the receiver *R*. When receiving the `prepare` packet, *R* determines the validity of the payment as stipulated by a higher-level protocol and can either reject the payment by sending a `reject` packet back, or accept the payment by responding with a `fulfill` packet (step ③). The `fulfill` packet contains the pre-image of the agreed-upon condition. Connectors forwarding a `fulfill` packet verify the included pre-image against the payment condition in the previously received `prepare` packet.

The Hyperledger Quilt project provides a Java implementation of the Interledger protocol for permissioned blockchains [197]. The project provides a set of rules for enabling ledger interoperability, formats for network packets and a framework for designing applications that leverage ILPv4.

4.2.8 Information Exchange

We end with a brief discussion on techniques for the exchange of private information across different ledger implementations. Whereas asset exchange involves transfer of ownership, information exchange requires that the buyer does not learn the information without the seller receiving something in return. An information exchange is said to be *fair* when this aforementioned property holds [57].

The FairSwap protocol is the most advanced approach in this direction and ensures a fair exchange of digital goods by leveraging smart contracts and arithmetic circuits [56]. The protocol introduces a *proof-of-misbehaviour* that proves if a seller misbehaves during an exchange. This proof is computationally cheap to construct. The OptiSwap protocol extends FairSwap by incorporating an interactive dispute resolution protocol, reducing the communication overhead of FairSwap [198]. Delgrado et al. describe a protocol for fair data exchange based on the Bitcoin scripting language [199]. The protocol is based on a new primitive, private key-locked transactions, that allow the atomic exchange of a private key for Bitcoin. This private key is then used to decrypt the traded information.

4.2.9 Comparison and summarization

Table 4.1 summarizes existing approaches to cross-chain asset trading, and also shows the approach proposed in this work. We further assess these approaches based on the following three criteria:

1. **Universal.** does the approach enable asset exchange between any permissioned blockchain?

2. **Avoids Trusted Parties.** does the approach require a trusted party to mediate in the trade? We also consider trusted committees or notaries as a trusted party, even though approaches leveraging semi-trusted authorities often assume a weaker trust model.

Approach	Universal?	Avoids trusted parties?	Guarantees atomic exchange?
Central Authorities	✔	✘	✘
Atomic Swaps	✘	✔	✔[1]
Notary Schemes	✔	✘	✘
Bridging	✘	✔	✘
Sidechains	✘	✔	✔
Internet-of-Blockchains	✘	✔	✔
Interledger Protocol V4	✔	✘	✘
Information Exchange	✘	✔	✔
XChange (this work)	✔	✔	✘[2]

[1] If the involved parties claim there assets before the time lock expires.
[2] But the economic gains of adversaries are limited.

Table 4.1: A comparison of approaches to exchange assets between permissioned blockchains.

3. **Guarantees Atomic Exchange.** does the approach provide an atomic exchange of assets? An atomic exchange guarantees that both parties either exchange all assets, or nothing happens.[5]

Table 4.1 shows that five out of the eight discussed approaches for asset trading are not universal and cannot facilitate asset exchange between any permissioned blockchain. Asset exchange through a central authority or notaries can support an extensive range of different ledgers but requires the active participation of these authorities in the involved blockchains. The Interledger Protocol is specifically designed for broad adoption and high interoperability between blockchains, but requires semi-trusted connectors to facilitate the payment. We observe that most asset trading mechanisms avoid the need for trusted parties and leverage cryptographic techniques to facilitate trade between different blockchains. Finally, we notice that half of the identified approaches do not guarantee an atomic asset exchange.

4.2.10 Problem Description

Our analysis of existing asset exchange approaches indicates that no solution is universal, avoids trusted parties, and guarantees an atomic exchange. We also observe that there are no solutions that are both universal and avoid trusted parties, to the best of our knowledge. We argue that any mechanism with these two properties requires a compromise on the atomicity criteria. As pointed out by literature on e-commerce, trade atomicity can be addressed by either (1) leveraging specific cryptographic techniques or (2) by using escrow services [200]. Approach (1) violates the universality criteria: it lowers the applicability of our solution since the involved blockchains now require the availability of cryptographic techniques. Approach (2) violates the criteria to avoid trusted parties since an asset exchange is now executed by an escrow.

Even without atomic trade guarantees, we can ensure that the risks of losing funds to the counterparty are manageable. We believe that the Interledger Protocol V4 is the closest to our envisioned universal cross-blockchain value exchange since it makes no assumptions on the technical capabilities of involved payment networks and operates with manageable risks. However, value exchange with the Interledger Protocol does not di-

[5] In some problem domains, this is also referred to as a *fair* exchange [57].

rectly proceed between two traders and is coordinated by intermediate connectors instead. We now formulate three requirements for our asset exchange mechanism:

1. **Universality.** We require that our mechanism enables the exchange of assets between a large range of permissioned blockchains. In particular, asset exchange using our mechanism should not be limited to a selected number of blockchain architectures with specific features or with support for particular transaction types. We argue that this requirement is critical for broad adoption of our mechanism.
2. **Avoid Reliance on Trusted Parties.** We require that our mechanism avoids dependence on trusted parties to settle a trade. Asset exchange should proceed through direct interactions and payments between traders.
3. **Manage Counterparty Fraud.** To achieve universality, we believe that we have to forego the atomicity requirement. Without atomic guarantees, we must address the situation where a trader might actively try to commit fraud for economic gains. Our solution requires adequate measures to manage counterparty fraud during ongoing trades.

These requirements directly lead to the following research question: *how can we devise a universal mechanism to exchange assets that are stored on different permissioned blockchains, without having trusted authorities involved in the exchange and with manageable counterparty risk?*

4.3 Solution Outline

To avoid the involvement of an intermediary during trades, we leverage an *accounting mechanism* to make all trade activities public and openly accessible to involved traders. Individual accountability is a long-standing and widely used approach in electronic commerce to detect malicious behaviour and to deter fraudsters [115, 201]. By logging full trade specifications, a trader can build a profile of other traders and decide whether it wants to engage in a particular trade, without the involvement of trusted authorities. This approach enables traders to operate according to their own business rules and to manage the economic value at stake.

In this section we outline XChange, our universal mechanism for asset exchange between permissioned blockchains. In XChange, a trade between two traders A and B is modelled as a sequence of payments between the trading parties. At the minimum, a trade involves two payments, one from A to B and one from B to A. W.l.o.g., assume that A initiates the first payment to B in a particular trade. A complication during this trade could arise when B refuses to conduct a payment back to A, after having received a payment from A. In this situation, B has committed *counterparty fraud* since it compromised the assets that A has sent to B. In general, the party that conducts the first payment during a trade is exposed to *counterparty risk* where this party can lose assets to the counterparty without receiving a payment in return.

4.3.1 Recording Trades

We address fraud concerns by storing trade records in a tamper-proof distributed log. This distributed log then enables XChange traders to detect if a party might have committed

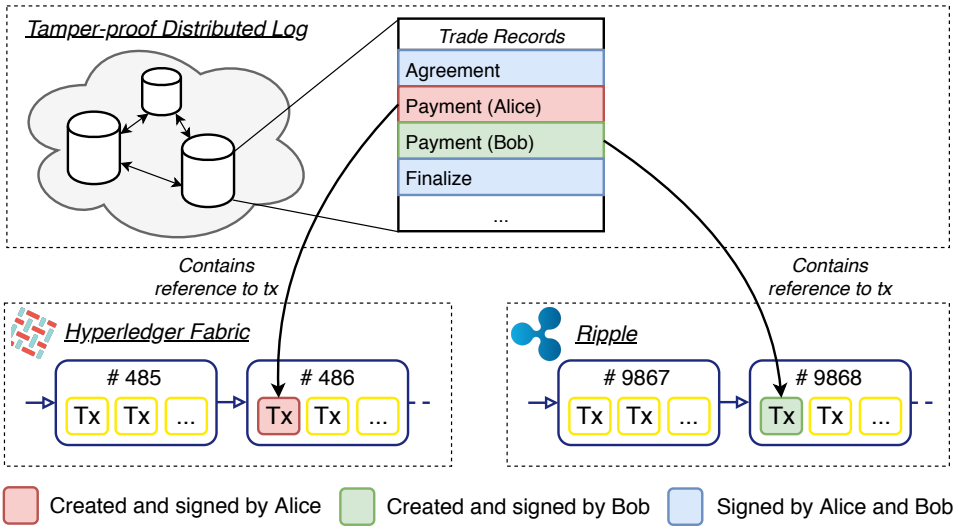


Figure 4.4: High-level overview of our XChange trading mechanism. In this example, Alice sells some FabTokens that are managed by Hyperledger Fabric to Bob, who pays Alice in XRP (Ripple) tokens. Full trade specifications are stored in a distributed log.

fraud during an ongoing trade, further discussed in Section 4.3.2. If so, a trader refrains from starting a trade with the suspected party. We store records of every trade, which makes it difficult for a trader to hide the existence of a specific trade or to unilaterally revert the status of an ongoing trade to a prior state. Each record in the distributed log is digitally signed by its creator and therefore irrefutably created by a specific peer. We envision that the distributed log can also be audited by external authorities to resolve potential disputes that would arise during the trade procedure. However, we consider the details of such audits beyond the scope of this work. The technical requirements of the distributed log are later discussed in Section 4.4.

Before we show how trade specifications are recorded, we first elaborate on two implications of using a shared log. The first implication is that our solution requires participants to agree on the same distributed log when trading assets using XChange. However, in contrast to Internet-of-Blockchains solutions such as Cosmos and Polkadot, XChange does not require businesses to migrate their deployed ledger applications to a different environment. Instead, businesses can voluntarily leverage our mechanism and join the XChange peer-to-peer network without any changes to existing applications. This approach lowers the adoption barrier of XChange by interested parties. The second implication pertains to privacy concerns, arising from the full accounting of trade specifications. We acknowledge that it might be undesirable to publicly record trade information in specific situations since the records can reveal sensitive business practices. However, since privacy preservation will likely require additional mechanisms and cryptographic techniques, we consider privacy concerns beyond the scope of our work.

We have considered an alternative design where trade records are stored by the ledgers that are involved in the trade. Even though this design would avoid the need for a shared

log, it would result in the fragmentation of trade records across potentially many ledgers, making it infeasible to accurately determine in which trades a specific trader is currently involved. Furthermore, a user can be unable to accurately build profile information of another trader since this user might not have the appropriate credentials to inspect the records and transactions on a specific ledger. This design would also require logic to store XChange trade records within all supported blockchain environments, requiring significant implementation efforts.

We show a part of the distributed log in Figure 4.4 and highlight four records that together describe a completed trade between two traders, Alice and Bob. This trade exchanges tokens that are managed by a Hyperledger Fabric and a Ripple ledger. The lower part of Figure 4.4 shows parts of the Hyperledger Fabric and Ripple ledger. A completed trade that has been stored in the distributed log consists of the following three record types:

1. An `Agreement` record contains the specifications of an upcoming trade, e.g., the agreed amount of assets that will be exchanged between the traders. It also includes information on which party conducts the first payment during the upcoming trade. The `Agreement` record bears the digital signature of both traders and can be appended to the distributed log by any of the traders. We further describe this record type, and the other two record types below, in our protocol description (see Section 4.5).

2. A `Payment` record contains the details of a specific payment that has been conducted during a trade. This record includes the identifier of the newly issued transaction that transfers assets in the involved blockchain network. For instance, the `Payment` record created by Alice in Figure 4.4 contains a reference to the transaction that she submitted in the Hyperledger Fabric network. Likewise, the `Payment` record created by Bob points to his transaction in the Ripple network. By including the identifier of the transaction in this record, the trading counterparty, and other traders, can verify if the payer transferred the assets. Others can verify the validity and inclusion of the transaction reference by the `Payment` record by inspecting the appropriate blockchain.

3. A `Finalize` record completes a trade. A `Finalize` record is appended to the distributed log by the party that received the last payment during the completed trade.

In addition to these three record types, XChange also includes the `Order`, `CancelOrder`, and `CancelTrade` records. The `Order` and `CancelOrder` records are used when creating a new order and when cancelling an unfulfilled order, respectively. These two record types are further discussed in Section 4.5. The `CancelTrade` record can be appended to the distributed log to unilaterally abort the trade if one of the parties becomes inactive during a trade. This feature is later discussed in Section 4.3.3.

4.3.2 Risk Mitigation

Even though the distributed log provides traders with an overview of ongoing and finished trades, XChange does not yet address the situation where a trader conducts *counterparty fraud* during a trade. As a result, the economic gains of adversaries may be unbounded

since a malicious trader can commit fraud in many trades. Consider a simple trade between Alice and Bob, where Alice sells 2 FabTokens for 40 XRP, and Bob sells 40 XRP for 2 FabTokens. Both Alice and Bob are expected to individually send their respective assets to each other. Since we do not assume atomic exchange, one of the parties, say Alice, has to initiate the first transfer. As soon as Alice sends 2 FabTokens, she is exposed to *counterparty risk*, as Bob *may not* send back the respective 40 XRP.

In this section we present risk mitigation strategies of XChange that limit the gains of traders committing counterparty. These strategies mainly aim at minimizing the assets at stake by dividing each trade into smaller chunks (Section 4.3.2) and by bounding the total amount of obligation a party can enter into (Section 4.3.2). Throughout the paper, we name the party that is exposed to counterparty risk as *risktaker*, while the other party is called *risky*. Determination of the trade roles (who becomes the risky party and who becomes the risktaker) in a prospective trade is explained in Section 4.5.

Incremental Settlement

The first risk mitigation strategy we introduce is *incremental settlement*, where a trade is incrementally completed in k near-equal, smaller payments. Assume that in our fictitious trade between Alice (A) and Bob (B), parties agree to use an incremental settlement with $k = 2$. The total trade, therefore, would consist of four consecutive payments as illustrated in Figure 4.5. Alice is the *risktaker* in this trade, and she does the first payment. Notice that after each payment by Bob, the parties are on par with each other.[6] Termination of the trade at this state would not cause an economic loss for any of the parties. On the other hand, after each payment by Alice, the trade is in a state where Bob has an economic gain of 1 FabToken and Alice experiences an economic loss. With incremental settlement and $k = 2$, Alice is risking only a loss of 1 FabToken, instead of 2 FabTokens. We refer to the amount of risked assets as the *assets at stake*.

Similar to making multiple, smaller payments in the Interledger protocol, traders in XChange can gradually complete a trade in smaller steps and thus keep the risks manageable. On the one hand, in a trade where each party transfers value v to the counterparty, the economic gains of an adversary is reduced to $\frac{v}{k}$. On the other hand, incremental settlement prolongs the trade since more payments are made, and as such more transactions must be included on the blockchains that are managing the assets being traded. In general, a trade completed using incremental settlement requires $2k$ `Payment` records in the distributed log. In XChange the value of k is determined by the risktaker party of the trade and recorded in the `Agreement` record associated with the trade.

As we experimentally show in Section 4.8.2, incremental settlement reduces the value at stake during ongoing trades. This strategy is not applicable when a trade cannot be completed incrementally, e.g., when exchanging property titles or securities. Such assets are usually represented by non-fungible tokens on the ledger and gain their value from uniqueness. Even though these assets cannot be exchanged using incremental settlement, traders can still benefit from the second risk mitigation strategy that bounds the economic gains of adversaries.

[6]We assume here that a trade exchanges an equal amount of value between both traders. In practice, there are usually small profit margins where one party would gain slightly more in value when the trade is complete.

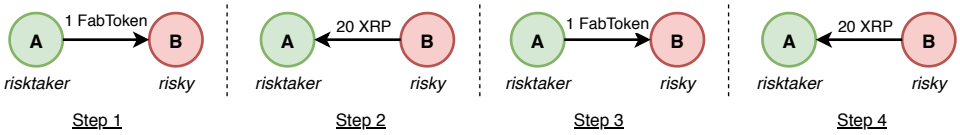


Figure 4.5: An asset exchange with $k = 2$ between Alice (A) and Bob (B), trading a total of 2 FabTokens and 40 XRP. During this trade, Alice is the risktaker since she is exposed to counterparty risk. Bob is the risky party since he is able to commit counterparty fraud after step 1 and 3.

We briefly comment on the economic implications of incremental settlement. Since this approach prolongs the trade duration, it might happen that the price of assets being traded goes in the favour of one of the trading parties. This is particularly true for high-volatile assets being offered on open, public blockchains. Price volatility enables the situation where a party can deliberately prolong a trade to profit from price fluctuations. Ideally, this is something that should be taken into consideration when counterparties create their offer. We acknowledge that dealing with this economic effect is currently an open issue of our trading system. At the same time, we believe that this issue is less prominent in permissioned blockchains since the price of such assets are usually not defined by trading volume but rather are priced based on real-world assets (e.g., stablecoins).

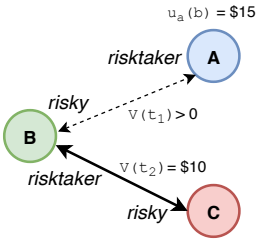
Bounded Obligations

Even though incremental settlement reduces the number of assets the risktaker puts at stake, it does not prevent an adversary from taking part in multiple concurrent trades as a risky party and commit counterparty fraud. In the simple trade example above, consider the case where Bob initiated another trade as a risky party with Charlie before finalizing his trade with Alice. Assume further that both trades are in a state where both Alice and Charlie have made their payments and are waiting for Bob's response. There is no restriction for Bob to enter into another trade before fulfilling its trade obligations to Alice and Charlie.

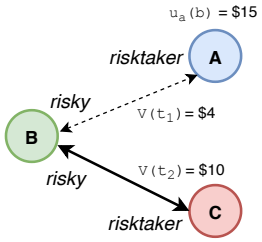
By devising rules that describe when a trader will start a trade with another party, we can bound the economic gains of adversarial parties under the assumption that non-adversarial traders follow the protocol. We notice that the *risky* party of a trade has no reason to refrain from engaging in an upcoming trade since it has nothing to lose. A trader becoming a risktaker in a prospective trade, however, must assess the risky party by inspecting the distributed log to determine if it is "safe" to engage in a trade with it.

One way to mitigate the risk of counterparty fraud would be to forbid a trader from being risky in simultaneous trades. However, this approach may lead to a situation where a risktaker can arbitrarily delay the trade duration, preventing the risky party to engage in trades with others. Instead, we choose to bound the *obligations* a trader enters into, by limiting the total amount of *assets at stake* within trades where a particular trader is involved in as a risky party. In other words, XChange employs trade restrictions which ensure that a malicious trader can only commit counterparty fraud up to a specific value.

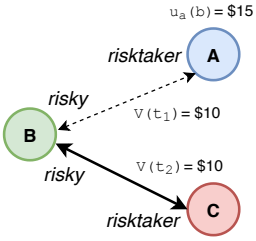
In XChange, every trader a assigns a *trust threshold* $u_a(b)$ to every prospective trader b, and refuses to enter into the trade with b if the total amount of *assets at stake* in all *open* trades in which b is the risky party is larger than $u_a(b)$. Open trades are the ones which



(a) Trader a will agree with t_1 since b is not risky in any ongoing trade.



(b) Total (ongoing + prospective) obligations of b is smaller than the trust threshold $u_a(b)$. Trader a will agree with t_1.



(c) Trader a will not agree on t_1 with since total obligations of b exceed the threshold. a and b may negotiate increasing k.

Figure 4.6: Three scenarios in which a trader a has to decide on starting a prospective trade t_1 in which b will become risky. a agrees with the trade in (a) and (b), and refuses to trade in (c). A solid line represents an ongoing trade whereas a dashed line represents a prospective trade.

do not have a respective `Finalize` record for its `Agreement` record in the distributed log. Formally, given a distributed log $\mathscr{L}$, let $P_{\mathscr{L}}$ be the set of all open trades and $P_{\mathscr{L}}(b)$ be the set of open trades in which trader b is the risky party. Let $V(t)$ be the *assets at stake* of a trade t. This value represents how much value a risky party can seize during a trade. The total value of obligations of a trader b is referred to as $B(b)$ and is as follows:

$$B(b) = \sum_{t \in P_{\mathscr{L}}(b)} V(t) \tag{4.1}$$

A trader a accepts to be a risktaker in a prospective trade t' with trader b if the following holds:

$$u_a(b) \geq B(b) + V(t') \tag{4.2}$$

We illustrate the idea of bounded obligations in Figure 4.6 which shows three scenarios involving traders a, b and c. In all the scenarios, a trader a has to decide if it wants to start a prospective trade t_1 with trader b. We assume that the value of assets involved in trades can be expressed into another asset type, say in United States Dollars ($). This conversion

could be based on the market price of the involved assets.[7] The value of *assets at stake* in prospective trade t_1 is \$10, and both parties have agreed that a becomes the *risktaker* and b becomes the *risky* if the trade starts. Trader a determines a trust threshold $u_a(b) = \$15$ for trader b.

Assume b is already involved in another trade t_2 with the trader c and that t_2 is not finalized.

In Figure 4.6a, trader b has the role *risktaker* in t_2. Since b is not the risky party in t_2, it does not have any obligations, i.e., $B(b) = 0$. Therefore, as long as $u_a(b) > 0$, trader a can decide to start a trade with b. In Figure 4.6b and 4.6c, b is the risky party of t_2 where $V(t_2)$ is equal to \$4 and \$10, respectively. In Figure 4.6b, trader b's obligations stemming from ongoing trades amount to \$4, i.e., $B(b) = 4$. Since b's prospective obligations $V(t_1) + B(b)$ is smaller than the trust threshold, a agrees to trade with b. In Figure 4.6c, the prospective obligations of b amount to \$20 and thus exceed the threshold $u_a(b)$, which would result in the refusal of t_1 by trader a. However, even in this scenario, traders may agree to reduce *assets at stake* by increasing the k. Using $k = 2$, for example, lowers $V(t_1)$ to \$5.

Additional Comments on Risk Mitigation

Flexible Conformance. We note that a trader can always ignore the risk mitigation strategies described in this section and engage in other trades *at its own risk*. Doing so, however, does not provide restrictions on the economic gains of adversarial parties but it enables participants to engage in trade with traders with which there is an existing trust relation (e.g., the traders know each other in real life). Trades that parties started at their own risk do not impact the obligations of the risky party in such trades. Such trades contain a special flag in the associated `Agreement` record.

Subjectivity and Trust. We note that the threshold function u_a is a subjective matter for a trader a and is highly dependent on the notions of trust and reputation, which are outside the scope of our work. Without loss of generality and for simplicity, we assume in the rest of the paper that $u_a(b)$ is equal to a constant U for all trader pairs a and b.

Determination of k. Parameter k signifies the number of payments each party does in a trade. This value is proposed by the risktaking party during trade negotiations and is included in the `Agreement` record in the distributed log. We note that both sides of a trade are concerned with the value of k. For the risktaker, k determines the *assets at stake*, i.e., the value that the risktaker may lose in case of counterparty fraud by the other party. For the risky party, k affects the maximum rate of trade a party can be involved in as a risky party. While lowering the value of k brings together low-risk advantage for the risktaker and trust advantage for the risky party, it, in return, increases the duration of a trade, i.e., the number of transactions needed to settle the trade.

4.3.3 Cancellation of a Trade

We note that a trade may never complete if a risktaker goes offline. The total amount of *assets at stake* in all such *stalled* trades in which a party b is a risky party may reach a point where no-one wants to trade with b, even if b is not at fault. Therefore, the ability of a trader b to trade with others may be forever restricted. To address this situation,

[7] It can also be that traders have differing opinions on the market price of a particular asset, e.g., based on their buy and sell orders.

we allow a risky party to explicitly cancel an ongoing trade by including a `CancelTrade` record in the distributed log. This record can only be included by the risky party, and is only acknowledged by other traders if (1) the risktaker is currently responsible for transferring assets to the risky party during the trade, and (2) at least some time Δ_t has elapsed since the last activity in trade t. The value of Δ_t should be well above the confirmation times of transactions submitted to the involved blockchains, to avoid the situation where one might consider a trade as stale while a transaction is still being finalized in the involved blockchain. When a trade is cancelled, no other assets should be exchanged. After the risky partner cancelled participation in a trade, it loses its risky status and can then participate in other trades.

We note that a risky party a can try to trick another party b into acknowledging a `CancelTrade` record by publishing a `Payment` record with a non-existent transaction identifier. Therefore, b needs to inspect the involved ledger to determine the validity of a `CancelTrade` record created by a. However, b might not have the appropriate credentials to read transactions on this ledger. Even though the `CancelTrade` transaction might be valid, we assume that b will not acknowledge the `CancelTrade` record when it cannot accurately determine its validity. We argue this is reasonable since this particular situation is likely to be infrequent. We also believe that this design decision does not significantly limit the efficiency of our mechanism.

4.4 System Assumptions and Threat Model

We first discuss the XChange system model. This includes our assumptions on the blockchains that are managing the assets being exchanged, the requirements of the distributed log used by XChange, and the specifications of the underlying XChange network. We then present the threat model of XChange, and state the goals and capabilities of adversarial parties.

4.4.1 Blockchain, Distributed Log, and Network Specifications

The XChange mechanism coordinates asset exchange between permissioned blockchains. W.l.o.g., we denote the blockchains that are managing the assets being exchanged by $\mathscr{B}_a$ and $\mathscr{B}_b$ respectively. XChange only requires that $\mathscr{B}_a$ and $\mathscr{B}_b$ can represent assets and transfer assets to another owner. The consensus mechanisms deployed by $\mathscr{B}_a$ and $\mathscr{B}_b$ might be fundamentally different. We assume that for each involved blockchain, the fraction of adversarial parties is bound by the threshold necessary to ensure safety and liveness properties. In PBFT-based consensus algorithms, this threshold is usually $\frac{1}{3}$ of all nodes involved in the consensus algorithm [133].

XChange stores trade records in a distributed log, denoted by $\mathscr{L}$. We require that the entries stored by $\mathscr{L}$ are immutable and append-only. If entries in $\mathscr{L}$ would be mutable or can be removed, a trader could trick a counterparty into starting a trade, commit counterparty fraud, and remove all traces of the trade. Similar to how participation in $\mathscr{B}_a$ and $\mathscr{B}_b$ is explicitly approved, participation in $\mathscr{L}$ should be managed by an authority. We envision that a trader joining XChange re-uses the well-defined identity under which it participates in one of the permissioned blockchains. We remark that $\mathscr{L}$ can, for example, be realized through a blockchain with support for smart contracts.

XChange users participate in a peer-to-peer network, which is used to send point-to-point messages to other users. This network is particularly used during trade negotiation, as we further specify in Section 4.5. We assume that peers in the XChange network know the network addresses of other peers.

4.4.2 Peer Model

We now elaborate on the assumptions of peers participating in XChange.

Each peer in the XChange network owns a cryptographic key pair consisting of a public and a private key. The public key of a specific peer is known to others and uniquely identifies it in the network. Their private key is used to digitally sign data such as records appended to $\mathscr{L}$, or outbound messages in the peer-to-peer network.

As we discussed in Section 4.4.1, the digital identity of each peer in the XChange network uniquely identifies a real-world user. Identity validation should be performed by a Registration Authority (RA), which is external to our system. The RA could be the same authority that approved participation in $\mathscr{B}_a$ or $\mathscr{B}_b$. We assume that the RA does not collude with traders in XChange. In XChange, well-established digital identities are necessary to prevent misbehaviours such as a Sybil Attack and a distributed denial-of-service attack [101, 202]. We also use verified identities for accountability purposes, where misbehaviour in a trade can be traced back to a real-world persona.

Whereas existing work primarily focuses on *how* assets are exchanged, the XChange mechanism also includes primitives for traders to specify trade interest through orders, and to find trading partners that can fulfil these orders. We distinguish between *makers* and *takers*. A maker is a peer that creates a specific order, whereas a taker is a peer that fulfils an order. Makers introduce trading opportunities and liquidity to the XChange network. A peer in XChange can act as both maker and taker, for distinct orders. The maker-taker order model is also adopted by related protocols that enable the exchange of tokens on the Ethereum blockchain, namely 0x and AirSwap [46, 47]. System designers can also consider to leverage more advanced decentralized matchmaking solutions, e.g., as described in our prior work [203].

4.4.3 Threat Model

Adversarial parties in XChange aim to *maximize their economic gains* by committing counterparty fraud in ongoing trades. Adversarial parties could attempt to append invalid records to $\mathscr{L}$, intentionally ignore incoming messages in the peer-to-peer network, or refuse to respond to messages during trade negotiation. They also could strategically ignore the risk mitigation strategies described in Section 4.3.2. We assume that adversaries cannot compromise the integrity of the distributed log $\mathscr{L}$ used by XChange and cannot undermine the security of the blockchains that are hosting the assets being traded, $\mathscr{B}_a$ or $\mathscr{B}_b$. We also assume that the cryptographic primitives used by all involved blockchains are secure (e.g., digital signatures cannot be forged) and that the computational capabilities of adversaries are bounded.

4.5 The XChange Trading Protocol

We now present the XChange trading protocol for asset exchange between permissioned blockchains and specify all operations conducted by peers that are participating in a trade. We assume the system and threat model described in the prior section. The protocol consists of four phases. In the first phase, makers specify their trade interest by appending new orders to the distributed log $\mathscr{L}$. During the second phase, takers negotiate with makers about orders they would like to fulfil and append an `Agreement` record to the distributed log when they reach an agreement. During the third phase, the maker and taker execute the trade by exchange assets through payments. The trade is finalized in the fourth phase with a `Finalize` record.

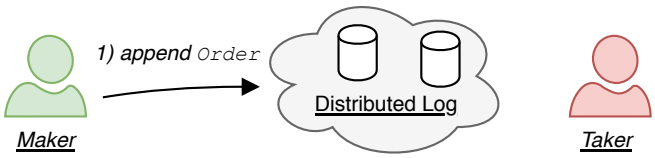


Figure 4.7: Phase I of the XChange trading protocol: makers (depicted in green) indicate trade interests by appending an `Order` record to the distributed log.

Phase I: Order Creation and Cancellation

During the first phase of the XChange protocol, makers create new orders and append these orders to $\mathscr{L}$, see Figure 4.7. When a trader intends to buy or sell some assets, it constructs a new order which we denote by O. O contains details on the quantity and the type of assets that the maker desires to buy and sell. The order creator provides this information as a two-tuple of asset quantities, also called an *asset pair*. The first asset quantity in the asset pair indicates the assets that the order creator wants, and the second asset quantity indicates what the order creator offers in return. An asset quantity is described by the combination of an integer value and a string that indicates the asset type. For example, if a trader intends to sell 2 FabTokens for 40 XRP tokens, it creates an order with asset pair (2 FabToken, 40 XRP).

O includes an integer value, k, that specifies the order creator's preference regarding the number of partitions each payment is divided in. As discussed in Section 4.3.2, one way how XChange reduces value at stake is by using incremental settlement. The inclusion of k in O indicates the risk that the maker is willing to take in an upcoming trade that fulfils O if the order creator would become the risktaker. Furthermore, O includes the address of the wallet in which the order creator wishes to receive assets from a prospective trader during an upcoming trade. By including this information, a taker knows to which address it should transfer its assets. This information is also used by other traders to verify if the maker has received assets from a taker.

After adding all required fields to O, the order creator serializes the order and embeds it in an `Order` record. The order creator then appends the `Order` record to $\mathscr{L}$. The order identifier can be determined by taking the hash of the record content, which we denote by $H(O)$.

A maker can cancel any of their non-expired orders that are not being fulfilled by an ongoing trade. This is achieved by the maker appending a `CancelOrder` record containing $H(O)$ to $\mathscr{L}$.

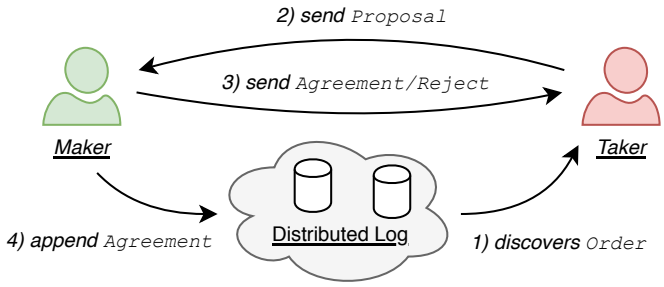


Figure 4.8: Phase II of the XChange protocol: a maker and taker negotiate a trade agreement. Upon a successful negotiation outcome, a dual-signed `Agreement` record will be appended to the distributed log.

Phase II: Trade Negotiation

During the second phase of the XChange trading protocol, a maker and taker negotiate a trade, see Figure 4.8. If the negotiating maker and taker agree to trade, one of the parties appends this agreement to $\mathscr{L}$. We now describe this negotiation process.

This phase starts when a taker discovers an order O, included on $\mathscr{L}$, that it wishes to fulfil. Assume that this order has been created by a maker M. Before sending a trade proposal to M, the taker performs two checks that determine if the taker should trade with M. First, the taker checks if it is willing to trade with M as a person. For instance, M could have attempted to commit counterparty fraud in the past, which could be a reason for the taker to refrain from trading with M. Second, the taker determines if it is safe to trade with M, according to the *bounded obligations* strategy described in Section 4.3.2. The taker checks the trades in which M is involved by inspecting the latest records on $\mathscr{L}$ involving M. If M is already involved in a trade T, the information on $\mathscr{L}$ also reveals if M in T is a risky party or a risktaker.

When both checks pass, the taker creates and sends a `Proposal` message to M. A `Proposal` message contains a proposal for M to fulfil order O. A taker includes four pieces of information in a `Proposal` message. First, it includes the identifier of O, so the maker knows which order the taker wants to fulfil (a trader could have created multiple orders). Second, the taker includes its destination wallet address to which M should send its assets during the trade. Third, the taker includes an integer value, k, that indicates how much risk the taker is willing to take if it would become the risktaking party. Finally, the taker includes a boolean value in the proposal indicating if the taker becomes a risktaker in the upcoming trade. At a high level, a `Propose` message represents a new order that indicates the taker's trade preferences.

When M receives a `Proposal` message from taker T, it also verifies whether it wants to trade with T. Specifically, M performs the same two checks as the taker did. Furthermore, M verifies if it agrees with the role classification proposed by the taker. If validation fails, the maker immediately sends a `Reject` message back to the taker, containing the identifier

of the rejected order and, optionally, why M has rejected the proposal. If M agrees with the proposal and also wishes to trade with T, M constructs an `Agreement` record, which includes the identifier of the order being fulfilled and the proposal created by the taker (including the taker's signature). This `Agreement` record is signed by M, sent back to T, and appended to $\mathscr{L}$. Inclusion of the `Agreement` record on $\mathscr{L}$ binds the maker and the taker to the trade agreements. Since the risktaker is exposed to counterparty risk in the upcoming trade, the preferred value of k by the risktaker is used during the upcoming asset exchange. If the maker is the risktaker, the value of k as specified in the `Order` record describing O is leading. Otherwise, if the taker becomes the risktaker, the value of k specified by the taker in its proposal is leading.

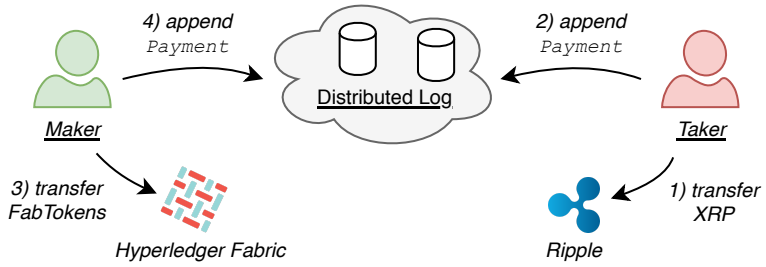


Figure 4.9: Phase III of the XChange protocol: a maker and taker trade by exchanging assets. In this trade, the taker is the risktaker (initiating the first payment) and the maker is the risky party.

Phase III: Trade Settlement

During the third phase of the XChange trading protocol, assets are exchanged between the maker and taker, and the trade is settled. Figure 4.9 visualizes a trade between a maker and taker, with $k = 1$, where the maker sells FabTokens, a token managed by Hyperledger Fabric, and gets XRP (Ripple) tokens in return from the taker. This trade, fulfilling order O, consists of two payments, one from the maker to the taker, and one from the taker to the maker.

Asset exchange starts by the risktaker (the taker in this specific example) issuing a transaction to the Ripple network managing the XRP tokens. This Ripple transaction transfers XRP tokens from the wallet specified in the `Proposal` message to the wallet address that was specified by the maker in the `Order` record associated with O. After the taker has issued this transaction in the Ripple network, it appends a `Payment` record to $\mathscr{L}$, which contains the identifier of the order being fulfilled, and the identifier of Ripple transaction. The `Payment` record allows the maker (and other traders) to verify that the taker has transferred the correct amount of assets to the maker.

After the maker has verified that it received the agreed amount of XRP tokens, it conducts the next payment by issuing a transaction to the Hyperledger Fabric network. This transaction transfers FabTokens from the wallet specified in the `Order` record to the wallet that was specified by the taker in the `Proposal` message. The maker then appends a `Payment` to $\mathscr{L}$, which includes the identifier of the transaction in the Hyperledger Fabric

network. This payment process repeats until all assets have been exchanged between the maker and the taker.

There is a risk that a trade does not progress when one of the traders becomes inactive. As pointed out in Section 4.3.2, a stale trade is only a minor concern for the risktaker since this party can still engage in other trades after Δ_T time has elapsed. A risky party can explicitly cancel an ongoing trade to dismiss its responsibility as a risky party by appending a `CancelTrade` record to $\mathscr{L}$. This record only contains the identifier of the order currently being fulfilled. Other traders should verify that the `CancelTrade` adheres to the rules as outlined in Section 4.3.2, to prevent the risky party from illegally cancelling a trade after having committed counterparty fraud.

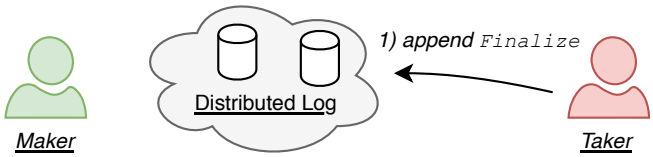


Figure 4.10: Phase IV of the XChange protocol: the taker finalizes the trade.

Phase VI: Trade Finalization

When all assets have been exchanged, the party receiving the final payment during a trade creates a `Finalize` record and appends it to $\mathscr{L}$, see Figure 4.10. Since the risky party conducts the final payment during a trade, finalization is always performed by the risktaker. Inclusion of a `Finalize` record on $\mathscr{L}$ completes a trade, say T_1, between the maker and taker, and both parties can now start new trades with others.

4.6 Security Analysis

We now analyse the security of the XChange mechanism. First, we prove that the economic gains of adversarial parties committing counterparty fraud are upper-bounded. We then discuss the scenario where multiple adversaries collude to gain an advantage as a group.

4.6.1 Counterparty Fraud Limitations

We further analyse the *bounded obligations* strategy presented in Section 4.3.2. This strategy define an upper bound on the obligations an adversary can enter into, under the condition that all honest traders act rationally and try to minimize their risk.

Limiting the Gains of Adversaries. To show that XChange limits the amount of fraud, we assume —for clarity— that all honest peers fix a unit trust threshold U, and the number of payments in each trade t is fixed to K. Specifically, where P is the set of traders and T is the set of all trades, the following is assumed:

$$(u_i = U) \quad \forall i \in P \quad \text{and} \quad (k_j = K) \quad \forall j \in T.$$

Under these two assumptions, XChange guarantees that:

(1) the total value of assets an adversary can gain as result of counterparty frauds is limited to U,

(2) the loss of an honest party in a trade is limited to V/K, where V is the *assets at stake* during the trade.

Assume an adversarial trader B is involved in $(n-1)$ trades denoted by $t_1, t_2, \ldots, t_{n-1}$ in which B is the risky party and is in trade negotiations with an honest trader A for the prospective trade t_n. Assume $\sum_{i\in\{1,\ldots,n-1\}} V(t_i) \leq K \cdot U$ and $\sum_{i\in\{1,\ldots,n\}} V(t_i) > K \cdot U$. Under these conditions, trader A does start a trade with B, given that A follows XChange protocol.

We can now show the correctness of the statement (1) and (2) above. There are two ways a malicious party can commit counterparty fraud. Firstly, it can choose to become inactive in a trade and not conduct a payment back to the risktaker. Secondly, B could append a `Payment` record to $\mathscr{L}$ that points to a non-existent or invalid transaction, attempting to trick the risktaker counterparty and other traders. In both cases, the value of fraud in a trade t cannot exceed $V(t)/K$, which verifies statement (2). Therefore, even when we assume that B commits counterparty fraud in all the active trades it is involved, the total value of assets B can gain does not exceed U, which verifies statement (1).

We can now relax our assumptions on the objectivity of trust threshold (U) and the number of payments (K). Assuming each trade t has its own number of payments k_t agreed by the trading parties, the amount of assets the risktaker can lose in t is limited by $V(t)/k_t$. When $u_A(B)$ is the trader A's subjective trust threshold assigned to trader B, then trader A does not start a trade t with B if the existing obligations of B exceed $u_a(b) - V(t)/k_t$. Assuming k_t is not bounded and that $V(t)/k_t$ may converge to zero, a trader B can start a trade as a risky party with a trader b, only if its obligations occurred from ongoing trades is limited by $u_a(b)$. Accordingly, defining $\bar{u} = \max\{u_j(b) : j \in V\}$ where V is the set of all prospective traders with b, the total amount of assets that b can seize in XChange cannot exceed $\bar{u}$.

Limit on the Risktaker's Loss. In XChange, the risktaker's loss in a single trade is bounded with the trust threshold u associated with the risky party. Furthermore, as the risktaker is involved in the determination of number of payments (k) during trade negotiations, it can reduce its own risk to any extent. Nonetheless, we note that XChange does not introduce a theoretical bound on the loss of a risktaker over time, but delegates the risk management to the risktaker by assuming a trust mechanism to determine the trust threshold. Specifically, XChange provides the risktaker with two important parameters u and k with which the risktaker can minimize its own risk, relying on the trust mechanism.

Malevolent Cancellation of a Trade. We now analyse the situation where a risky party B cancels its ongoing trade t_1 with A by appending a `CancelTrade` record to $\mathscr{L}$, before the trade is finalized and when it is B's turn to make payment. When a third party C considers entering in trade t_2 with B, it will discover the `CancelTrade` record in $\mathscr{L}$ and check whether the trade cancellation by B is legitimate (see Section 4.3.3). Recall that the cancellation of a trade by B is legitimate if it is currently the responsibility of the risktaker to conduct the next payment. The trade cancellation of t_1 is therefore not valid since B has committed counterparty fraud. If the trade cancellation were legit, B would not have been under no obligation in trade t_1, since it would have transferred assets to A. Now, when C

verifies the status of B and detect an illegitimate trade cancellation, C will not engage in trade t_2 with B.

4.6.2 Collusion Resistance

In a collusion attack, a group of traders follows a common strategy to subvert the network or gain advantages as a collective. The XChange mechanism is highly resistant against collusion attacks since adversarial parties are not able to gain more economic gains when working together, given that $\mathscr{L}$ provides secure storage of included trade records. We argue that the resistance against collusion can be addressed to the absence of group-based coordination in XChange. Tasks that would involve coordination among a group are usually vulnerable to attacks where a majority of the group colludes to gain advantages over non-colluding users. In XChange, trade proceeds through the direct interaction between the involved traders and therefore, cannot be influenced by groups of colluding adversaries.

4.7 Distributed Logging of Trade Records

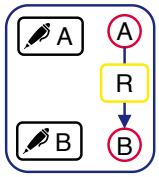
The XChange trading protocol described in Section 4.5 requires a distributed log to securely and irreversibly store `Order`, `CancelOrder`, `Agreement`, `Payment`, `Finalize` and `CancelTrade` records. We choose to build XChange upon TrustChain [98] which is a shared data structure with a sharp focus on tamper-resilience and trustworthy record storage. In this section we motivate our choice for TrustChain and outline how TrustChain is used to store XChange records.[8]

4.7.1 TrustChain: A Scalable Ledger for Accounting

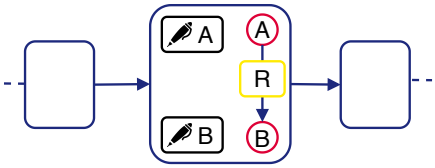
Based on the idea of blockchain ledgers that order transactions in a directed acyclic graph (DAG), Otte et al. designed, implemented, and deployed TrustChain. TrustChain is a ledger that is optimized for lightweight, tamper-proof accounting of data elements [98]. The key idea is that individuals maintain and grow their *individual ledger* with records. Other users verify these records according to some pre-defined rules. This makes TrustChain similar to solutions for tamper-proof, distributed accounting, such as PeerReview [96]. TrustChain does not aim to prevent integrity attacks on the data structure, e.g., fork creation, but instead guarantees eventual *detection* of these attacks. This yields superior scalability compared to other ledgers but allows for the situation where some malicious activity targeted at the ledger might go undetected for some time, for example, hiding specific transactions. In TrustChain, this can be addressed by waiting longer before accepting a record as valid. Individuals in TrustChain are not required to store all records in the network and might choose to store different parts of the global DAG ledger. TrustChain does not reach a global consensus over all records but relies on participants to detect inconsistencies in individual ledgers.

We argue that TrustChain is a suitable ledger to store XChange records, for the following four reasons. First, TrustChain allows participants to verify the integrity of other individual ledgers themselves, and determine whether a party is already involved in a

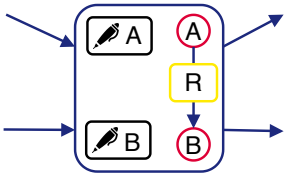
[8]The TrustChain architecture described in this chapter is an earlier version of ConTrib, which has been presented in Chapter 2.



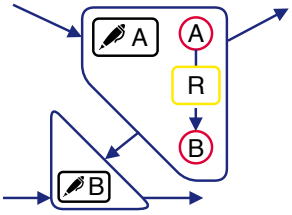
(a) A block with a record (*R*) between two users (*A* and *B*). Each block contains a single record, two digital signatures and two digital identities.



(b) An individual ledger. Each block in the chain contains a hash that describes the prior block.



(c) To increase security, each block also references the previous block in the chain of a record counterparty.



(d) To improve scalability, we extend the TrustChain structure to support concurrent block creation.

Figure 4.11: Storing records in TrustChain.

trade or not. There is no requirement to reach a global consensus on the integrity of included records. Second, TrustChain does not require network-wide replication of all records but enables individuals to selectively share parts of their individual ledger with others. This feature reduces storage requirements and allows XChange to also run on devices with storage limitations, as we demonstrate in Section 4.8.3. Third, the TrustChain structure is optimized to store bilateral records that are signed by two parties. This aligns well with the XChange trading protocol since many operations could benefit from support for bilateral records (for example, trade agreements). Finally, TrustChain is already being used by various decentralized applications that require accounting features, such as self-sovereign identities and inter-bank payments [105, 115]. At the time of writing, the public TrustChain ledger contains over 160 million records, created by 96'000 unique identities.[9]

4.7.2 Storing TrustChain Records

We now outline how a record between two interacting users *A* and *B* is recorded in TrustChain, see Figure 4.11. Each record is stored within a block. Figure 4.11a highlights one block containing a record between *A* and *B*. Each block contains a single record (*R*). A record can be a generic description of any interaction between users, for instance, a trade agreement or a payment. Both interacting parties digitally sign the block with the record by using any secure digital signing algorithm. These signatures are included in the block and ensure that participation by both parties is irrefutable. It also confirms that both parties agree with the record itself. Others can effectively verify the digital signatures included in a block. After all required signatures have been added to a block, the block is

[9]See `http://explorer.tribler.org`

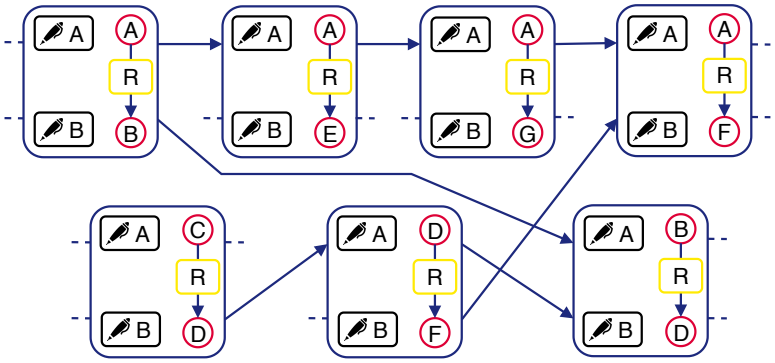


Figure 4.12: The TrustChain ledger, with seven blocks created by seven participants.

committed to the local databases of the two interacting parties and broadcast to a limited number of random peers in the network.

The security of stored blocks is improved by linking them together, incrementally ordered by creation time. In particular, each block is extended with a description (hash) of the previous block. Each block has a sequence number that indicates its position in the individual ledger. This results in the structure shown in Figure 4.11b. As a result, each user maintains their individual ledger, which contains all records in which they have participated. This sets TrustChain apart from the structure of traditional blockchains, where the entire network maintains a single, linear ledger.

Note how the blockchain structure in Figure 4.11b allows A to modify blocks in their individual ledger without being detected by others. In particular, A can reorder the blocks in its individual ledger since validity can quickly be restored by recomputing all hashes. In most blockchain applications, the global consensus mechanism prevents this kind of manipulation. TrustChain uses a more efficient approach: each block is extended with an additional (hash) pointer that points to the previous block in the individual ledger of the counterparty. This is visualized in Figure 4.11c. Each block now has exactly two incoming and two outgoing (hash) pointers, except for the last block in an individual ledger, which only has two incoming pointers. Modifications of the individual ledger by A, like reordering or removing blocks, can now be detected by one or more counterparties. To prove this fraud, a counterparty reveals both the correct block and the invalid block created by A.

When two parties transact and create a block, their chains essentially become entangled. When users create more records with others, it leads to the directed acyclic graph (DAG) structure, as shown in Figure 4.12. Figure 4.12 shows seven blocks, created by seven unique users. Each block is added once to the individual ledger of all parties involved in the record. For a more advanced analysis of the technical specifications and security of TrustChain, we refer the reader to the original paper by Otte et al. [98].

4.7.3 Improving TrustChain Scalability

According to Otte et al., TrustChain is designed to scale [98]. However, we identify that its design limits a user to one pending block creation at once. The main issue is that the digital signature of a counterparty is required before a new block can be appended to an

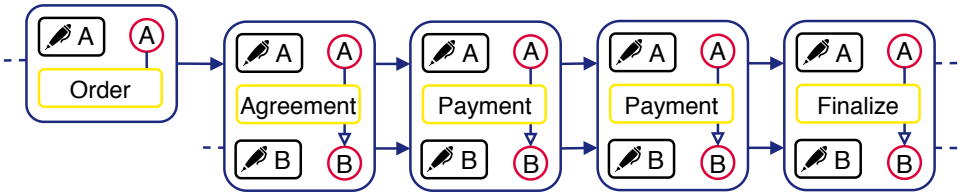


Figure 4.13: A part of the TrustChain ledger, storing an order created by a maker *A*, and full specifications of a finished trade between *A* and *B*.

individual ledger (since the input for the hash of each new block includes all signatures in the previous block). This enables an attack where a malicious user can purposefully slow down the block creation of others by delaying the signing process of a bilateral transaction it is involved in. It also limits the growth rate of individual ledgers and reduces the overall scalability of TrustChain.

We contribute to TrustChain and improve its scalability by adding support for *concurrent block creation*. The idea is to remove the requirement for a digital signature of the counterparty when appending new blocks to an individual ledger. We believe that this concurrency is necessary since it allows traders to append new records without reliance on other parties.

Our solution is visualized in Figure 4.11d. It shows a record between users *A* and *B*, initiated by *A*. We partition a block in two parts, and each block partition is appended to the individual ledger of exactly one party. Construction of a block between *A* and *B* now proceeds as follows: first, user *A* creates a record by constructing a block partition with the record content and its digital signature. User *A* adds this block partition to its individual ledger immediately (note that it does not include the digital signature of *B*). *A* now sends the block partition to *B*. If *B* agrees with the transaction, it signs the block partition created by *A*, adds it to its individual ledger, and sends his block partition (with their signature) back to *A*. User *A* stores the block partition created by *B* in its local database. The participation of both parties in this record can now be proven with both block partitions. This mechanism allows users to be involved in multiple block constructions at once.

4.7.4 Logging Trade Records on TrustChain

We now outline how `Order`, `CancelOrder`, `Agreement`, `Payment`, `Finalize` and `CancelTrade` records are stored on TrustChain. Figure 4.13 shows a part of the TrustChain ledgers of traders *A* and *B*. It includes a sequence of records that indicate a finished trade between *A* and *B*. Trade agreements, created during the second phase in the XChange protocol, are stored within a bilateral `Agreement` record and digitally signed by both involved traders. Individual payments are stored within bilateral `Payment` records. A `Payment` record signed by both parties indicates that the payer has conducted the payment and that the payee has observed the payment. Finally, a trade finalization is stored within a bilateral `Finalize` record. Since the overhead of creating new TrustChain records is low, we also store orders as unilateral `Order` records in individual ledgers. A unilateral record only contains the digital signature of its creator. Figure 4.13 shows an `Order` record, created by maker *A*. Furthermore, `CancelOrder` and `CancelTrade` records are also included as unilateral records

in one's individual TrustChain ledger.

A particular issue is that the fragmented nature of the TrustChain DAG makes it difficult for takers to discover interesting orders quickly. Specifically, `Order` records are by default only stored in the individual ledger of the order creator. Therefore, we introduce *matchmakers*, peers that continuously collect the TrustChain records of other peers in the network, and organize the information in `Order` records in a local database. Matchmakers aggregate orders, and takers can query the database of matchmakers to find interesting orders. Makers also send their TrustChain blocks with an `Order` record to known matchmakers after creation. The role of matchmakers in XChange is comparable with that of relay nodes in 0x [46] and indexers in AirSwap [47].

4.8 Implementation and Evaluation

In this section we present the implementation of XChange and our experimental evaluation. The evaluation answers the following three questions: (1) how effective is XChange at reducing fraud gains? (2) what is the overhead of XChange, in terms of trade duration, when XChange is deployed on low-resource devices? And (3) How scalable is XChange in terms of throughput and trade duration when increasing the system load?

The following experiments quantify the effectiveness, performance and overhead of our XChange mechanism. During these experiments, we assume that asset settlement is instant. As such, we do not actually connect a permissioned ledger to XChange for asset transfers. We believe this is a reasonable experiment setup since our aim is to evaluate the scalability and overhead of our approach without the interference of external systems.

4.8.1 Implementation Details

We have implemented the XChange in the Python 3 programming language. Our implementation spans a total of 4'702 lines of code and uses an event-based programming model, powered by the built-in `asyncio` library. The implementation is open source and all software artefacts (source code, tests, and documentation) are published on GitHub.[10]

Networking. We have built XChange on top of an existing networking library that is also used by TrustChain. This library provides the functionality to devise decentralized overlay networks and has built-in support for authenticated network communication, custom message definitions, and UDP hole punching [63]. For efficiency reasons, the UDP protocol is used for message exchange between peers.

Request Stores. To correctly process incoming messages during trade negotiation (phase II of the XChange protocol, see Section 4.5), XChange stores the state of outgoing messages. The state of outgoing messages is stored in distinct *request stores*. For each outgoing message that has a state attached, a unique identifier is generated, a new request store containing this identifier is created and the generated identifier is appended to the outgoing message. Traders that receive a message with this identifier are required to include the same identifier in their response message. Incoming response messages with an unknown identifier are discarded and not processed further. Each request store can have an optional timeout, indicating the duration after which the request store times out. When a request store times out, it is deleted.

[10] See `https://github.com/tribler/anydex-core`

Wallets. XChange organizes different types of assets within wallets. These wallets provide a convenient interface to the information provided by connected blockchain platforms. Wallets expose functionality to query the existence of specific transactions, fetch the content of specific transactions, and to transfer available assets to another trader.

Our implementation contains a `Wallet` base class that can be extended by programmers to create wallets that store different types of assets. For testing purposes, we have implemented a `DummyWallet`, which is used when executing the unit tests and when running the experiments described in this section. This wallet does not interact with any blockchain and simply waits for some duration before returning a (fake) response.

4.8.2 Reducing Fraud Gains

Our first experiment quantifies the effectiveness of reducing fraud gains when trading with XChange. We experimentally show the effectiveness of the risk mitigation strategies discussed in Section 4.3.2.

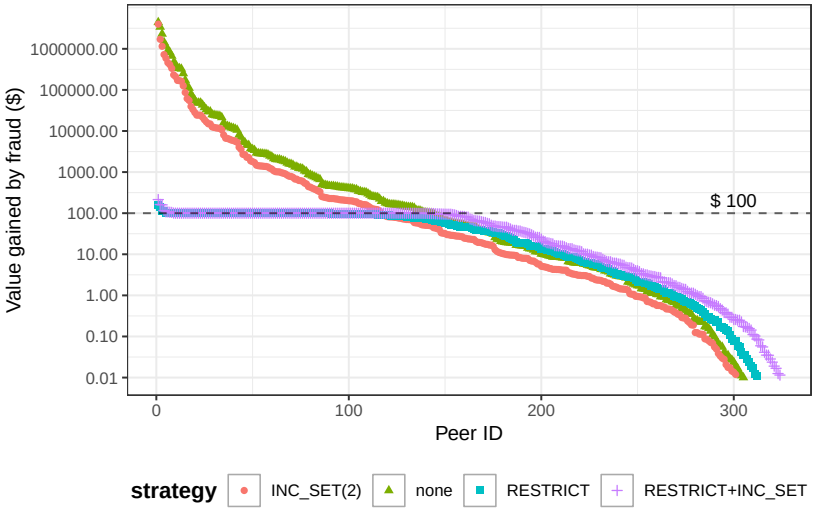
Setup and Workload. For this experiment, we reconstruct a real-world dataset, containing buy and sell orders published on the BitShares blockchain [49]. The BitShares platform enables users to issue custom assets and to trade these assets with others. We extract the buy and sell orders made during the last week at the moment of writing, and replay them with XChange. This results in a dataset with 230'000 orders, consisting of 125'527 buy orders, 104'423 sell orders and 212'489 cancellation events of existing orders. These orders have been created by 1'161 unique users and involve 243 different assets. Our data includes the orders created between November 11, 2020, and November 18, 2020. Since our dataset does not contain granular temporal information on order creation and cancellation, we assume that each order is uniformly created in the time interval between the last block and the block that contains this specific event. To accurately apply the *bounded obligations* strategy (see Section 4.3.2), we compile a list with the market price of all assets in our dataset, expressed in USD, by crawling a major BitShares block explorer.[11] We were unable to accurately determine the market price for 36 assets since they have a low or zero trading volume. We ignore the orders trading such assets during our experiment. We have published all scripts to construct this dataset in a separate GitHub repository.[12]

Since it is impractical to replay all the events in our dataset in real-time, we substitute our networking library with a custom discrete event simulator that is fully compatible with the `asyncio` library. At the start of the experiment, we create wallets for all peers with an unlimited amount of assets. To test the limitations of our mechanism in a highly adversarial setting, we model all peers as fraudsters, where they steal incoming assets whenever possible. Specifically, they commit counterparty fraud by not issuing a subsequent asset transfer after receiving some assets. During our experiment, a single peer acts as matchmaker and notifies traders about opportunities for their buy and sell orders. We fix the trust threshold U to \$100 for all peers when the *bounded obligations* strategy is enabled, meaning that the economic gains of an adversary are at most \$100.

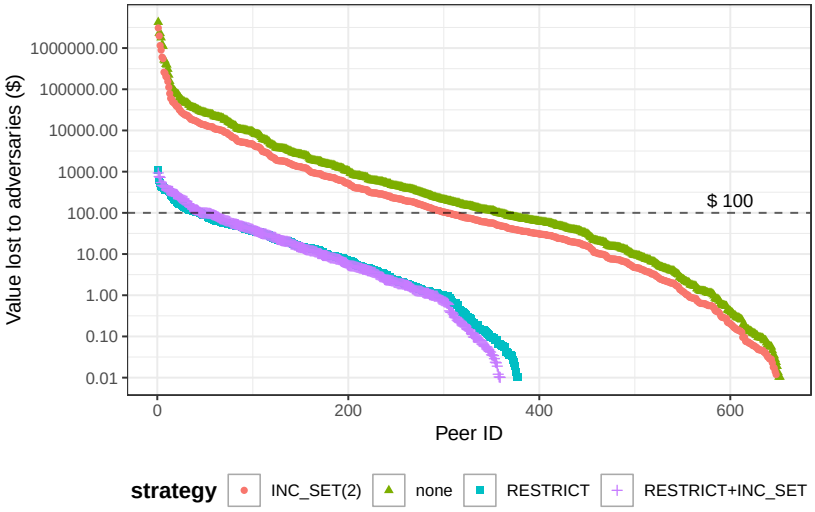
We test the effectiveness of our mechanism with combinations of the two risk mitigation strategies discussed in Section 4.3.2. With the `INC_SET(k)` strategy, we refer to the incremental settlement strategy where each trader makes k payments to the counterparty

[11] See `https://cryptofresh.com/api/docs`

[12] See `https://github.com/devos50/bitshares-orderbook-scripts`



(a) The economic gains of adversaries.



(b) The economic losses of traders.

Figure 4.14: The economic gains of adversaries and the losses of traders when replaying 230'000 BitShares orders with XChange, for different risk mitigation strategies. We have fixed the trust threshold U to $100.

during a single trade. The RESTRICT strategy denotes the strategy where a trader follows the *bounded obligations* strategy to verify whether it should trade with another party or not (see Section 4.3.2). We consider four experiment settings in total, with combinations of the RESTRICT and INC_SET strategies, and when no risk mitigation strategy is active. We note that the number of incremental payments when enabling the RESTRICT strategy is not fixed and depends on the current and prospective obligations of a counterparty.

Results. We show the results of our fraud experiments in Figure 4.14. Figure 4.14a shows the economic gains of adversaries, for combinations of the risk mitigation strategies discussed in Section 4.3.2. We show the value gained by adversaries on a logarithmic vertical axis. During our experiment we keep track of the fraud committed by adversaries, and sort these peers by the amount of fraud they have committed in USD. The horizontal axis shows the identifier of these peers. The total fraud gain without any risk mitigation strategy is $18.5 million. This number is reduced to just $18'609 under the RESTRICT+INC_SET strategy, *a reduction of 99.9%*. Under the RESTRICT strategy, the total fraud gain is $16'260, lower than the gains under the RESTRICT+INC_SET strategy. We address this due to the fact that some trade proposals are being denied since they cannot be completed without incremental settlement; these trades, however, might have been possible when using incremental settlement, which would have resulted in more fraud instances. We also note that a few adversaries have committed fraud with a total value of over $1 million when running without any risk mitigation strategy, and under the INC_SET(2) strategy. These successful adversaries likely created orders with competitive market prices, resulting in more trade proposals and opportunity for fraud.

Figure 4.14a clearly shows the effectiveness of the *bounded obligations* strategy. However, we observe that a few adversaries were able to commit fraud with a total value that exceeds our bound of $100. For the RESTRICT+INC_SET strategy, twelve adversaries have committed fraud with a total value over $100. We have identified that this issue arises from the weak consistency guarantees by TrustChain. Specifically, a trader A can be involved in the negotiation about many other orders at the same time, which together would exceed the bound of $100. When all counterparties query the individual ledger of A around the same time, all these parties might decide that it is safe to trade with A and as a result engage in trade with A. In addition, a counterparty might deliberately refrain from send its latest record(s) back, which we refer to as the *record withholding attack*. To address these issues, we suggest two extensions to XChange and TrustChain. First, XChange can also record all the trade proposals, and their responses by counterparties (accept or reject). Counterparties can take the outstanding trade proposals into consideration when applying risk mitigation. To address the record withholding attack, a party can disseminate the latest record of its counterparty in Distributed Hash Table (DHT), e.g., Kademlia [37]. Traders can then query the DHT network to fetch the latest record of a party A, and then query the individual ledger of A up to that record. Even with this timing issue present, Figure 4.14a still shows that our risk mitigation is highly effective and is capable of significantly reducing fraud gains.

Figure 4.14b shows the economic losses of traders, for different risk mitigation strategies. Again, we notice that the *bounded obligations* strategy significantly reduces the economic losses of traders. The maximum individual loss when applying no risk mitigation strategy and under the RESTRICT+INC_SET strategy is $4'255'731 and $928.79, re-

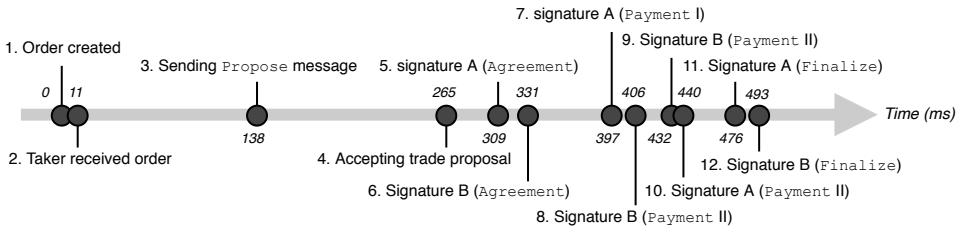


Figure 4.15: A timeline of the events during a single trade between a maker *A* and a taker *B*. The experiment is conducted on two hosted Raspberry Pis (3rd generation, model B+). The total duration of the trade is 493 milliseconds.

4

spectively. Figure 4.14b also highlights the effectiveness of incremental settlement under the `INC_SET(2)` strategy, compared to when no risk mitigation strategy is applied. In comparison to the fraud gains by adversaries, the economic losses by individual traders are not bounded but they are manageable.

Conclusion. Our experiment with real-world data proves that the risk mitigation strategies by XChange are effective and significantly reduce fraud gains of adversaries. In particular, we have experimentally proven that incremental settlement indeed decreases fraud losses, and that bounded obligations bounds the economic gains by adversaries.

4.8.3 Trading on Low-resource Devices

Our second experiment quantifies the latency added by XChange when conducting a trade between two low-resource devices.

Setup and Workload. This experiment is conducted with two hosted Raspberry Pis (3rd generation, model B+). The devices run the Raspbian Stretch operating system and the Python 3.5 interpreter. One device assumes the identity of trader *A*, and the other device acts as trader *B*. Furthermore, one device creates a new order, and the other device fulfils the order. The experiment is executed in an isolated environment: there is only network communication between the two Raspberry Pis. For this experiment, we use two different subclasses of `DummyWallet`, representing different assets. To measure the overhead of XChange, we configure these wallets such that assets instantly arrive when being transferred to another wallet. During the experiment, we log the timestamp of several events. At $t = 0$, the maker creates a new order. The trade is finished when both trading parties have signed a `Finalize` record and have committed this record to their individual ledgers.

Results. Figure 4.15 shows a timeline of the events during a single trade between the two Raspberry Pis. The full trade sequence, from the moment of order creation to mutual possession of a dual-signed `Finalize` transaction, completes in 493 milliseconds, less than half a second. Almost half of the trade duration, 254 milliseconds, is spent in phase II of the XChange trading protocol, the trade negotiation phase. During this phase, a trader determines whether a counterparty is already involved in a trade by inspection of the records in the TrustChain ledger of the other party.

Conclusion. This experiment shows that a full trade, including order creation, can be completed within half a second on low-resource devices if asset transfer would be instant.

Based on this experiment, we argue that the deployment of XChange in an Internet-of-Things (IoT) environment would be viable since its communication and transaction creation overhead is minimal. Asset management is a common feature in IoT [204]. XChange can be used to coordinate asset exchange between different IoT environments. However, our trading protocol requires periodic inspection of blockchain during an ongoing trade. Since maintaining a full transaction history is not realistic given the storage restrictions of IoT devices, XChange should rely on dedicated full nodes that have to appropriate credentials to participate in a specific blockchain. We believe that devices with less processing capabilities than Raspberry Pis are still capable of maintaining and securing TrustChain records. This belief should be verified with further experimentation through a small-scale deployment of XChange in an IoT environment where blockchain-based assets are managed and traded.

Even though the low trade duration on low-resource devices is a promising result, the experiment is not representative of a realistic trading environment where there are many traders creating orders and exchanging assets simultaneously. Furthermore, the prior experiment does not reveal the impact of our risk mitigation strategies on performance. Therefore, our next experiment focuses on the scalability of XChange and shows how our mechanism behaves under a higher system load.

4.8.4 Scalability of XChange

We now perform scalability experiments to quantify the performance of XChange as the system load and network size increases.

Setup and Workload. To explore the limitations and overhead of XChange, we conduct scalability experiments on our university cluster. The detailed specifications of the hardware and runtime environment can be found online.[13] Our infrastructure allows us to reserve computing nodes and deploy instances of XChange on each node. We use the Gumby experiment framework [65] to orchestrate the deployment of XChange instances onto computing nodes and to extract results from experiment artefacts. The scalability experiment is controlled by a *scenario file*, a chronologically ordered list of actions which are executed by all or by a subset of running instances, at specific points in time after the experiment starts. Each run is performed at least five times, and the results are averaged.

We increase the system load, namely the number of new orders being created every second. As the system load grows, so does the number of traders in the network. We devise a synthetic dataset to determine the performance of XChange under a predictable arrival rate of orders. In a network with n peers running XChange, n orders are created every half a second. To avoid the situation where all instances create new orders at the same time, the starting time of this periodic order creation is uniformly distributed over all peers, based on their assigned IDs (ranging from 1 to n). Each peer acts as a matchmaker and sends a new order to four matchmakers, which each peer randomly selects when the experiment starts. The experiment lasts for 30 seconds, after which $30n$ orders are created in total. Each order buys a single token in return for another token, to make matchmaking a predictable process. After 30 seconds, the experiment is terminated.

Scalability is measured as follows: first, we analyse the peak *throughput* observed during the experiment, in terms of trades per second. Second, we consider the average

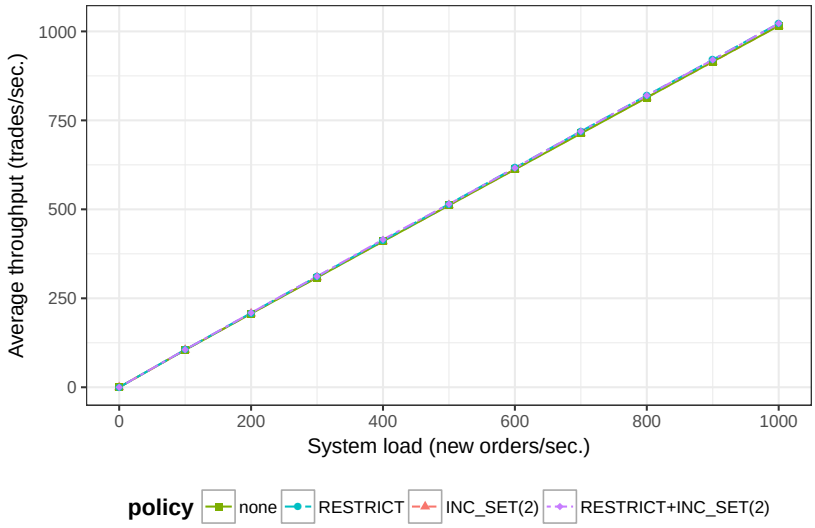
[13] See `https://www.cs.vu.nl/das5/`

order fulfil *latency*, which is the time between the creation of an order and the time until this order has been completed (the order creator has exchanged all assets as specified in the order).

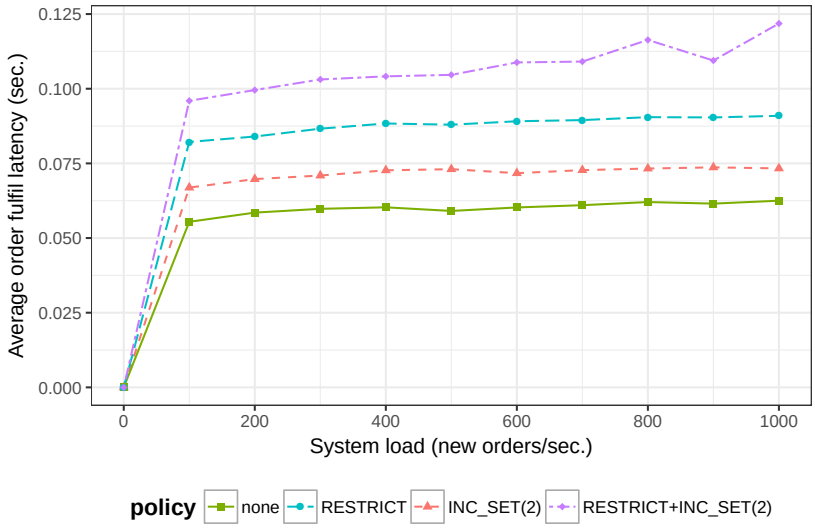
Results. The results of the scalability experiments are presented in Figure 4.16. We run each experiment with a specific system load up to 1.000 deployed instances (which is close to the limitations of the used hardware). Figure 4.16a shows how the peak throughput (expressed in trades per second, vertical axis) behaves with respect to the system load (horizontal axis). All experiment settings hint at linear scalability as the system load increases. Furthermore, enabling risk mitigation strategies does not appear to have a notable effect on the peak throughput. Experimentation on more compute nodes should reveal whether this trend continues when the system load exceeds 1.000 new orders per second.

Figure 4.16b shows the average order fulfil latency when the system load increases, for the four risk mitigation strategies. The average order fulfil latency remains largely constant when the system load grows. Applying the restriction and incremental settlement strategies increases the average order fulfil latency, since more operations have to be performed to successfully complete an order. We observe a moderate increase of latency when applying the `RESTRICT+INC_SET` strategies when the system load grows to 1.000 trades per second. The high system load is likely to increase the duration of individual trades beyond 0.5 seconds, which means that the `RESTRICT` strategy prevents traders from initiating a new trade with others. Since a trader now has to find a new party to trade with, the average order fulfil latency increases.

Conclusion. The main finding of this experiment is that the throughput (trades per second) scales linearly with respect to the system load and network size. We also observe that the average order fulfil latency remains largely constant as the system load grows. Further experimentation should reveal whether these trends continue with an even higher system load.



(a) Peak throughput.



(b) Average order fulfil latency.

Figure 4.16: The peak throughput and order fulfil latencies as the system load increases.

4.9 Conclusions

We have presented XChange, a universal mechanism for asset exchange between permissioned blockchain. XChange facilitates asset exchange without relying on particular transaction types or trusted third parties to mediate in the trading process. XChange records the initiation of a trade, individual payments, and the completion of a trade in a distributed log. By devising a set of rules that define when a party should engage in a new trade, we have limited the economic gains of adversarial parties. Specifically, when an adversary commits counterparty fraud, any further trade with this adversary are refused by honest parties until the fraud is resolved. Incremental settlement further reduces economic gains by splitting each payment into multiple, smaller ones.

We have implemented XChange and open-sourced its implementation. By replaying a dataset containing orders published on the BitShares blockchain, we have showed that XChange can significantly reduce fraud gains. We have also demonstrated the viability of trading on devices with low hardware capabilities. A single trade can be completed within half a second if asset transfers on external blockchain platforms would finish instantly. With a scalability experiment on our compute cluster, we achieved over 1'000 trades per second and found that the throughput of XChange in terms of trades per second scales linearly with the system load and network size.

Finally, we highlight two promising research directions for further work. First, it would be helpful to extend our mechanism with privacy-enhancing features that do not reveal full trade details to the network. Second, our mechanism would benefit from a more extensive risk model that allows a trader to further reason about other traders while judging their trade proposals. This risk model can, for example, also take into consideration the "response time" of a counterparty and classify slower counterparties as riskier.

5

Internet-of-Money: Real-time Money Routing by Trusting Strangers with your Funds

In this chapter we introduce a mechanism to significantly reduce the settlement duration of inter-bank payments. The key idea is to break up a particular inter-bank payment into a series of intra-bank payments between strangers which are quick to complete. Specifically, we address the challenging problem of giving money to others and relying on them to forward it. To identify fraud, we record money transfers between interacting strangers on a scalable, distributed ledger. This work represents a small step towards a generic infrastructure for trust, moving beyond proven, single-vendor platforms like eBay, Uber, and Airbnb.

Expanding upon trust relations, we design, implement, and evaluate our Internet-of-Money overlay network. Internet-of-Money is capable of real-time money transfers to different banks by routing funds through money routers. *A money router manages bank accounts at different banks. This removes the need for financial intermediaries, e.g., central banks, to handle an inter-bank payment. Our network reduces traditional payment durations from a day or even a few days in weekends, to mere seconds. With real-world experimentation, we prove that Internet-of-Money enables fast money forwarding. We also show that our overlay network is capable of discovering a majority of available money routers well within a minute, ensuring quick availability for end users. Finally, we demonstrate how the profit of cheating routers is limited and that misbehaviour is punished.*

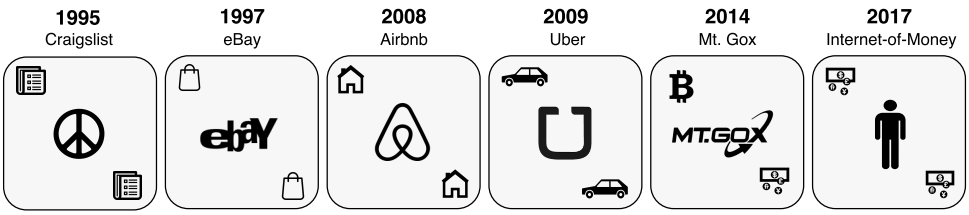


Figure 5.1: Influential milestones in the evolution of digital trust.

5.1 Introduction

Creating trust between strangers is at the core of numerous successful Internet companies. Starting 22 years ago, Craigslist offered an unmoderated mailing list of advertisements and gossip on which buyer and seller could be trusted. eBay formalized this in 1997 and introduced a star-based rating system that enables traders to build a trustworthy profile [205]. This e-commerce platform was launched at a time when people were still hesitant to use their credit card on a technology called The Internet. Nowadays, people let strangers sleep in their houses using Airbnb (since 2008). We trust Uber (since 2009) with our physical security and get into cars late at night with a driver that has never undergone a criminal background check or earned the appropriate qualifications to act as taxi driver. These influential milestones in the evolution of digital trust are shown in Figure 5.1.

We continue this evolution of building trust. We created an operational platform for one of the most challenging and sensitive applications, having other people handle your money. As we will show, solving this problem allows us to significantly reduce the duration of international payments between banks, from days to mere seconds.

Bitcoin is the first operational system that manages money without the need for a bank [1]. In the past, people were required to trust a central bank and a host of other intermediaries when making payments [206]. The fundamental technology of Bitcoin, blockchain, radically reduced the need to trust financial middlemen. It bootstrapped an economy where no one can be stopped from spending their money. Despite widespread speculation and ecosystems being worth billions, blockchain in general suffers from scalability issues due to inefficient mechanisms for fraud prevention, specifically the need to reach consensus on the transactions to be executed. Bitcoin is theoretically limited to seven transactions per second and Ethereum [2], a popular blockchain platform to deploy self-enforcing contractual logic, has a throughput of around twenty transactions per second [111]. Despite various scalability efforts like proof-of-stake and sharding, broader adoption of blockchain technology stays out [207].

While a majority of Internet users trust the company that operates and manages popular platforms, the events involving Mt. Gox highlighted how digital trust can be established and compromised [208]. Mt. Gox was at one point the largest Bitcoin exchange worldwide. In 2014, hackers stole Bitcoin, worth around $460 million at that time from the Mt. Gox exchange. This event, together with major data breaches in 2017 at high-profile companies like Uber and Equifax, exposed the weakness of centralized architectures [209]. They motivate research around decentralized technologies, like blockchain.

The generic problem of building trust between strangers resides on the edge of technol-

ogy, sociology, and behavioural science [210]. Whether someone can be trusted depends on properties like personality, level of authority, culture, and past behaviour. In this research, we address the trust problem from a technological perspective, using tamper-proof interactions on a scalable blockchain to explicitly record trust relations. This blockchain structure is built to detect fraudulent behaviour and misrepresentation. We explore whether a trust model based merely on historical encounters is sufficient to trust strangers with your money.

With established trust relations, we demonstrate how one can transfer money within seconds between different banks by relying on others to act as financial intermediaries. In comparison to most proven platforms, our solution is designed to be fully decentralized and autonomous. Our work is motivated by slow money transfers to other banks using existing systems. Particularly, inter-bank payments often take up to a day or even a few days during weekends to arrive in the account of a beneficiary since payments are usually processed in bulk.

The main contribution of this work is four-fold:

1. A trust model, based on repeated interactions that are stored on a tamper-proof, scalable blockchain.

2. *Internet-of-Money*, a novel and decentralized overlay network that allows real-time money routing to other banks.

3. A framework to programmatically interact with multiple banks and to initiate payments to others using Internet-of-Money.

4. Experimental quantification of the performance of our trust model, the speed of money transfers, and the efficiency of our overlay network.

5.2 Problem Description

Trust and fraud are essential problems to address when trusting others with your money. While most cryptocurrencies use a lottery system to stumble upon trustful executors, we rely on game theory to ensure honest behaviour has the largest rewards. We focus on the effective detection and punishment of fraudulent behaviour. While it is a common belief that money transfer systems should be safe against all kinds of fraud, we argue that it is sufficient for fraud to be detectable and punishable. This is comparable to the operation of credit card companies, which have to deal with a considerable amount of fraud on a daily basis. Detection of such fraud, however, is non-trivial.

The trust problem in this work can be modelled by the prisoner's dilemma, where two entities can either cooperate or betray each other [211]. Betrayal is also called *defection*. In the iterative prisoner's dilemma, players cooperate or defect iteratively and are able to punish opponents for their past decisions. Within the domain of money transfers, we assume a *send and forward* model where a user first sends money to another user, who in turn forwards the money to someone else. Forwarding funds is considered cooperation whereas keeping the incoming money is seen as defection. Detecting whether an entity has defected is a key requirement. Not cooperating should be punished by digital ostracism.

Many companies rely on centralized reputation mechanisms to estimate the trustworthiness of platform participants. In general, this leads to two problems. First, a solid track record built in one platform is often not reusable on other platforms. Second, building and maintaining an interaction history on multiple platforms simultaneously leads to fragmentation of one's trustworthiness scores. Users are increasingly being protected from such data silos by regulation [212]. Our aim is to devise a decentralized and generic reputation mechanism.

A mature research community exists around the design of decentralized reputation systems [103, 213, 214]. A notorious attack in decentralized systems occurs when a user first builds a high reputation by acting honest for some time and then abuses this accumulated trust for personal enrichment. This is also called the "pump and dump" method. Another challenging attack in decentralized networks is the Sybil Attack, where an individual creates multiple fake identities and initiate transactions with them to increase his or her standing in the community [101]. The Sybil Attack is hard to solve without trusted third parties, particularly in decentralized networks with open enrolment.

5

5.3 Settlement of Traditional Payments

Prior to elaborating how we can use the services of other users to realize real-time money transfers, we briefly explore the process of performing a payment with existing infrastructure. International payment systems are often proprietary and lack transparency. The largest inter-bank communication network is SWIFT, the Society for Worldwide Interbank Financial Telecommunication [215]. In April 2017, SWIFT recorded an average of 28.38 million payments per day or around 328 per second. This legacy network was founded in the 1970s and programmed in a language from the 1950s (COBOL). While a majority of financial institutions worldwide rely on SWIFT, joining the network is an expensive and involved process. Due to the high costs when initiating cross-border payments, many users and companies are shut out of the system. It is estimated that back-office costs for international payments need to drop by 90% to 95% for banks to remain competitive [216].

The SWIFT network exposes high processing or *settlement* times for inter-bank payments, in particular for international payments. While many companies and banks are working on new platforms to enable instant (international) payments, there are various issues that should be addressed [31]. These issues include real-time fraud detection and robust messaging standards [216].

A payment to another bank usually involves an intermediate settlement institution that is responsible for settling a payment between two parties [206]. This is often a central bank such as the European Central Bank (ECB). The settlement institution acts as an intermediary in the payment chain, addresses interoperability issues, and reduces settlement risks. Instead of handling numerous payment instructions and settling them individually, settlement institutions usually aggregate outstanding payments and settle them all at once on predetermined times. This is called *net settlement* or *netting*.

While inter-bank payments take a considerable amount of time to settle, moving funds within the books of the same bank is significantly faster. This is called an *in-house payment*. In-house payments have a relatively low settlement duration, usually a few seconds, since no inter-bank coordination is required.

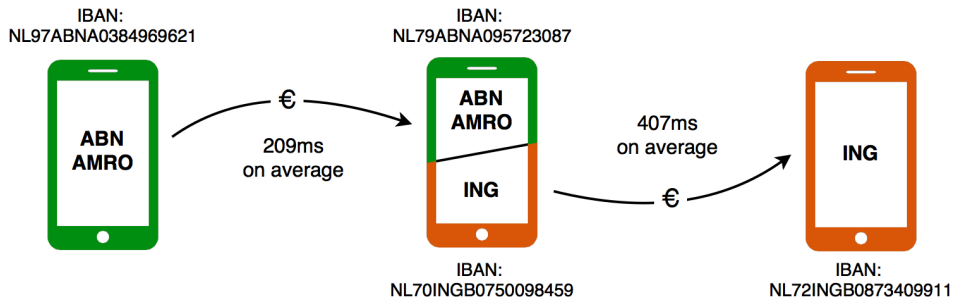


Figure 5.2: The essence of fast payments and our Internet-of-Money architecture: the smartphone in the middle acts as a money router using both ING and ABN AMRO bank accounts.

5.4 Our Money Routing Mechanism

Our solution to perform real-time payments is based on the observation that in-house payments are settled fast. For the banks that we tested, intra-bank money transfers are settled within mere seconds (see Section 5.7.1). Instead of using a central bank as settlement institution, we build a network of individuals that have bank accounts with multiple banks to complete an inter-bank payment. This approach is visualized in Figure 5.2 and works as follows. Assume a Dutch buyer, holding an ABN AMRO account, intends to pay another Dutch merchant that holds a bank account with ING. When this buyer initiates a payment with existing software to the merchant, the funds can take up to a day to arrive in the bank account of the merchant.[1] However, when using an intermediary that holds accounts both at Rabobank and ING, the buyer first sends the funds to the Rabobank account of this intermediary after which the buyer instructs the intermediary to forward the same amount of money from his ING account to the ING account of the merchant.

Since this way of sending money only involves two in-house payments, the merchant receives the funds within a few seconds. We call a payment conducted this way a *fast payment*, and we refer to the intermediary settling the transaction as a *money router*. We use the terms *initiator* and *beneficiary* to indicate the initial sender and final receiver of a fast payment, respectively. A fast payment can be facilitated by multiple routers to increase efficiency and availability. Note that fast payments lead to mutations in the account balances of the involved money routers and might deplete one of the bank accounts, resulting in service unavailability. This problem is addressed in Section 5.6.

Fast payments have three major advantages for users. First, it creates an open ecosystem for settlement activities, which benefits transparency and reduces the need for a central bank. Second, inter-bank settlement durations are significantly decreased, from days to seconds. Third, we reduce costs for inter-bank payments since no communication between banks is required, except when restoring balances (see Section 5.6).

Our system shares characteristics with the services provided by Wise. Wise is a currency exchange service that offers a cheaper alternative to established institutions when

[1]This problem is now addressed by Instant Payments, a settlement mechanism used by major Dutch banks since 2019. International payments, however, still have a high transaction fee due to the lack of an international real-time settlement system.

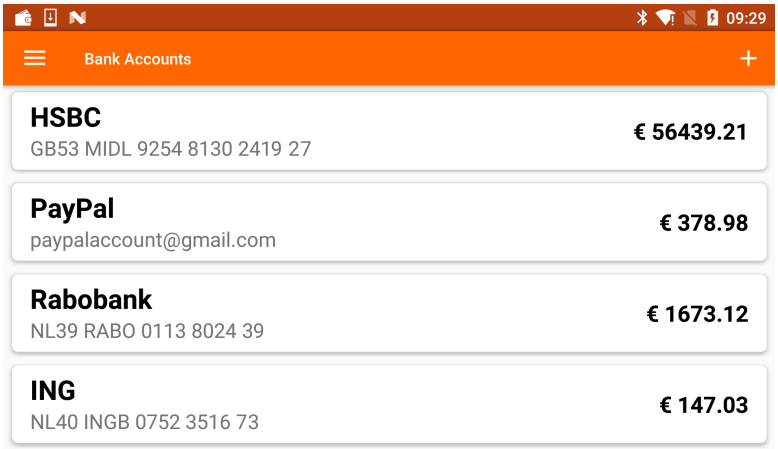
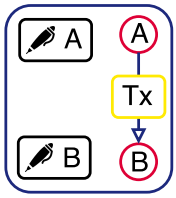


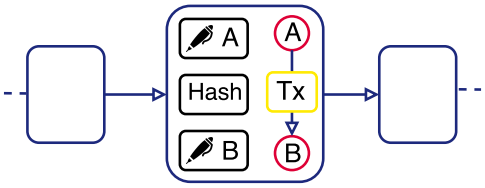
Figure 5.3: Our Android application to interface with different banks and to route money in real-time.

making international payments. It routes payments not by transferring the sender's money directly to the recipient, but by redirecting them to the recipient of an equivalent transfer going in the opposite direction. The essential idea is to convert international money transfers into a sequence of local transactions. Their approach is comparable with our money router mechanism, as it also aims to reduce fees and improve efficiency of traditional payments. However, international payments with Wise can still take a few days to complete, depending on the settlement duration of involved banks.

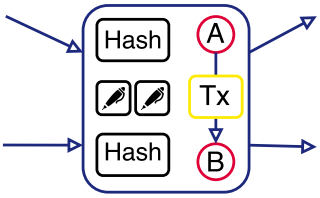
In the remainder of this work, we elaborate on our trust model and present the technical specifications of money routing. This includes an overlay network where any individual is able to quickly route money between bank accounts. For end users, we have built an Android application which interface is shown in Figure 5.3. Our mobile application allows users to add their bank account, to interface with different banks, and to initiate real-time inter-bank payments using our overlay network.



(a) A transaction (*Tx*) between two users (*A* and *B*), with two digital signatures.



(b) A blockchain of transactions. Each record in the chain points back to the previous one.



(c) To increase security, each record also references a record in the chain of the other transaction participant.

Figure 5.4: Recording a transaction between two users *A* and *B* in TrustChain.

5.5 Building Trust using Blockchain Constructs

The money routing mechanism described in Section 5.4 forwards money to intermediate money routers. Users are required to trust that these money routers correctly forward incoming funds to the beneficiary. We address this problem by accounting all money transfers in the system on a blockchain fabric.

We now explain our deployed, scalable blockchain fabric to gradually build trust between fast payment initiators and money routers: *TrustChain*. TrustChain is designed around transacting entities and is able to accurately capture interactions between users. We have presented preliminary experimentation with TrustChain for bandwidth accounting, attestations, and decentralized trading in prior work [217]. For an elaborate evaluation of TrustChain, we refer the reader to our published article [98].[2]

Figure 5.4 illustrates how a transaction is recorded between two users on TrustChain. Figure 5.4a shows a single transaction (*Tx*). Both parties sign the transaction with any cryptographically secure digital signature algorithm (our implementation uses ECDSA). This makes participation irrefutable and acts as an agreement for the transaction specifications. These digital signatures can efficiently be verified by others. After signing, the transaction is committed to the local databases of both transacting users.

A natural way to order records in a database is to chain them together, ordered by creation time. This is shown in Figure 5.4b where each record is extended with a pointer that points back to the prior record. In particular, this pointer is a hash computed from the description of the prior record using any cryptographically secure hashing algorithm (our implementation uses SHA256). Each record is equipped with a sequence number $s \in \mathbb{Z}$ (the sequence number of the genesis record is 1). This database organization resembles a

[2]The TrustChain architecture described in this chapter is an earlier version of ConTrib, which has been presented in Chapter 2.

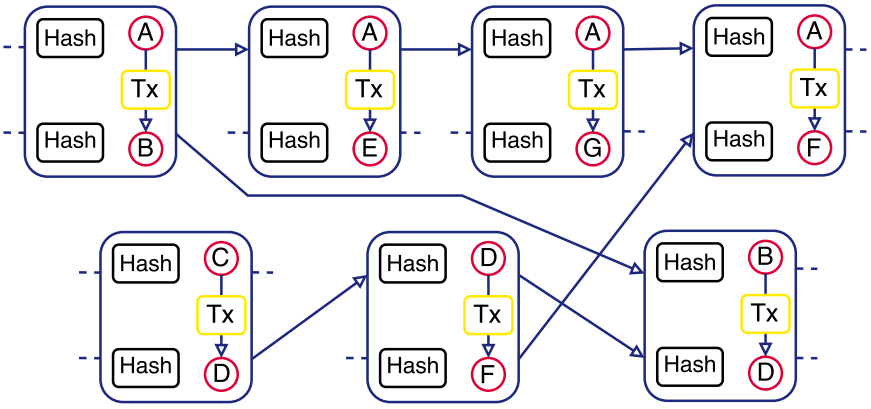


Figure 5.5: The tamper-proof TrustChain data structure to record transactions.

5

blockchain data structure. While cryptocurrencies like Bitcoin and Ethereum operate on a global blockchain, TrustChain gives each user their own personal chain.

Outside for the user operating a chain, the structure shown in Figure 5.4b is void of any control. Consequentially, a user is able to tamper with his historical transactions. For instance, individuals could try to remove transactions that are not beneficial for their standing in the network, e.g., a money router could try to hide the evidence that it has received some incoming funds. After modification of a record, validity of the chain can simply be restored by recomputing all prior pointers. To protect against local modifications, we extend each record with an additional pointer that points to the prior record in the chain of the transaction counterparty. This ensures that each record has exactly two incoming and two outgoing pointers, as shown in Figure 5.4c.

When two users transact, their chains essentially become interleaved or "entangled". This property makes fraud impractical to hide since a counterparty is able to proof malicious activities by revealing his record of the disputed transaction. When users initiate more transactions with others, they quickly become entangled in the network, leading to a directed acyclic graph (DAG) structure as shown in Figure 5.5. This figure shows seven records, created by seven unique participants. Users are able to collect records stored by others. This ensures adequate replication of TrustChain records throughout the network.

Recording Payments

We use the TrustChain data structure to record money transfers between individuals. Each payment and fast payment is assigned a unique identifier, generated by the initiator of this operation. We define two different transaction types:

1. `commit`: This transaction is a public commitment by a money router to forward received funds. It is signed by the initiator of a fast payment and other money routers, prior to transferring any money. The transaction includes the identifier of a fast payment and account address of the money router that should forward funds.

2. `sent`: This transaction type is signed by two parties involved in an in-house payment and implies that money has been sent and received. This transaction includes the

fast payment identifier and a boolean that is true if and only if the payment volume is above a threshold t.

We deliberately choose to hide the exact payment volumes due to privacy considerations, at the cost of reduced information.

Detecting and Punishing Fraud

The most challenging scenario occurs when a money router promises to forward money, but fails or refuses to do so. Since TrustChain provides us with a public ledger, disputes can be detected between transacting entities. Consider the situation when a router promised to forward money to Alice and claimed to have done so but Alice has not observed these funds in her bank account (yet). Other individuals can detect this situation when they observe a `commit` transaction, signed by both parties, and a `sent` transaction that is only signed by the money router that didn't forward the funds yet. As soon as such a dispute is detected, we do not consider this money router as intermediary for future fast payments until the dispute has been resolved. Note that a dispute can also occur when a router is unable to forward money, e.g., due to downtime of involved banks or insufficient account balance.

In addition to the aforementioned scenario, a user can intentionally lie that he or she has not received funds from a money router. It is impossible to make statements about the status of a specific payment without access to both bank accounts involved in a payment. To resolve disputes, we propose to use input from a dispute arbitrator in the form of an official, digitally signed statement. The dispute arbitrator can be any company that is able to query involved bank accounts. For instance, the bank involved in a fast payment could act as dispute arbitrator. A statement by the dispute arbitrator provides the status of a fast payment with a specific identifier and should be published on the TrustChain ledger. To discourage users from purposely creating disputes, the arbitrator should charge a small fee for publishing a statement. This fee should be covered by the party that made a false statement about money being sent or received. Note that dispute arbitration enables a new business model for banks within our system.

Quantifying Trustworthiness

We now discuss a mechanism to quantify the trustworthiness of honest money routers. These trustworthiness scores can be used as selection criteria for money routers when issuing a fast payment. Our proposed solution is based on past settlement services provided by money routers. We define a credit network G that models how much money a participant trusts to another individual. The graph is built using collected, dual-signed TrustChain transactions from others. Let $T_{a,b,R}$ indicate a successful money transfer from user a to b using the routers in the set R. Let (a, b, w) indicates a directed edge in G from user a to b with weight w. Now, each identity in our TrustChain network is modelled as a node in G. For each $T_{a,b,R}$ and each router $r \in R$, we create two directed edges: (a, r, w) and (b, r, w) where $w = min(0.01, t)$ (the minimum monetary value we trust to someone is €0.01). These edges represent trust in routers that have forwarded incoming money in the past. We exclude any money router that is currently involved in an ongoing fast payment.

To determine trust scores, we use an algorithm which has been studied extensively in related work, personalized PageRank [218]. The algorithm assigns a score between 0 and 1

to each node in G. These scores are used to pick intermediaries for money forwarding (see Section 5.6). We consider the node in G that performs the computation as trusted source. Using a reputation algorithm based on random walks is attractive due to its high scalability and low computational complexity. However, one might also consider using a reputation algorithm based on maximum network flow to compute trust scores. Such reputation algorithms are, in general, more resistant against active manipulation. In particular, we believe the Bazaar algorithm is suitable for this use case and provides additional security at the cost of increased computational requirements [219].

Preventing the Sybil Attack
Our current solution is vulnerable to a Sybil Attack where an attacker operates multiple entities that use the same bank account for money routing. We propose a mechanism called *router validation* to ensure that a specific bank account can only be operated by a single money router. The effectiveness of this method comes from the difficult and costly process of opening many accounts with different banks internationally. A router first registers a bank account by sending €0.01 to a trusted third party (TTP), for instance, a bank. The digital identity of TTPs are publicly available. When the payment is observed, TTPs sign and store a so-called *verify* transaction on TrustChain with the money router being validated as interaction counterparty. This transaction uniquely connects a bank account to a money router. Routers reusing bank accounts across multiple identities can be identified by inspecting TrustChain records and blacklisted by users and other routers.

5.6 System Design of Internet-of-Money

We expand upon fast payments and our trust model by designing a novel overlay network named *Internet-of-Money*. It operates on top of existing inter-bank payment systems, similar to how The Internet was built on top of the legacy telephone infrastructure.

The Money API
Our money routing solution requires money routers to forward funds to the beneficiary. A key requirement is to automate this process since the settlement duration of payments would otherwise be constrained by the response time of the operating users.

Except for the German FinTS payment protocol, there are no open standards yet for online banking. European legislation called PSD2 is forcing all EU banks to create open interfaces (APIs) [220]. We created one of the first open implementations capable of communicating with numerous banks. We combined banks in the Netherlands (Rabobank, ING, and ABN AMRO), the British bank HSBC, and the Luxembourg payment provider PayPal [221, 222]. We have partial support for banks in Italy, Greece, Sri Lanka, Turkey and Germany. We devised a single API to communicate with all these banks, called *The Money API*. The Money API provides primitives to login, fetch account balance, query mutations, initiate payments to other accounts and register devices. This library is designed to be extensible and is currently being tested. Due to legal considerations, we are currently unable to open source our library.

Money Routers
Each money router must offer settlement services with at least two different bank accounts. Having many money routers in the network directly benefits availability and load

balancing. A study conducted by NGData indicated that 37.7% of the respondents held accounts at different banks and are able to act as settlement intermediary for money transfers [223]. To create incentives for users to operate a money router, we include transaction fees. Transaction fees can be either fixed, e.g., defaulting to €0.01, or can be a percentage of a fast payment volume. We argue that these fees are necessary to cover costs enforced by banks when initiating cross-border payments during router rebalancing (see Section 5.6) or when using business accounts to route money. In addition, users can specify a minimum account balance to provide services in order to avoid taking on costs when their balance becomes negative. In the remainder of this work, we assume transaction fees are fixed. We also consider economic analysis of monetary incentives beyond the scope of this work since it is not critical to prove the technical feasibility of our overlay network.

Note that our design also allows the role of money router to be fulfilled by a single trusted third party or by a few selected trustworthy entities (i.e., financial institutions). A more centralized approach would mitigate some of the trust and security issues that arise from full decentralization. However, we consider open enrolment, the opportunity for any user to act as a money router, a cardinal property of our system.

Router Discovery

We design a gossip protocol for the discovery of available money routers, based on utility. If Alice wishes to discover a new router, she asks one of her known peers, say Bob, to introduce a router to her. Now, Bob tries to introduce a router to Alice through which she can route money. In general, the algorithm prioritizes routers that provide the most benefit to Alice. If Bob has no router in his set of known peers that are able to provide new services to Alice, he will introduce a random router to Alice. Repeating this gossiping protocol quickly converges to a network with connections between individuals able to provide routing services for each other. An evaluation of this mechanism is given in Section 5.7.2.

Building a Money Circuit

Prior to transferring money, an initiator of a fast payment starts by selecting eligible routers that are capable of handling the upcoming fast payment. We define a *money circuit* as the set of peers that are involved in a fast payment. This set contains at least one initiator and one beneficiary, and optionally one or more money routers. A money circuit that contains n money routers is called a n-hop circuit. Building a money circuit proceeds in a depth-first manner and starts with the initiator selecting a router, say r, that is capable of routing money to another account. Next, the initiator sends an *extend* message to r which contains the payment volume and the destination bank account of the fast payment. r responds with a boolean that indicates whether r has sufficient funds to handle the transfer. The response also includes a list of routers that are able to extend the money circuit, and the transaction fee charged by r. If r is able to handle the transfer, the initiator picks a router to extend the circuit with and sends an *extend* message again. These routers are picked based on trustworthiness scores. This process repeats until the initiator built a money circuit that can handle the fast payment. Users are able to change the maximum number of routers in a circuit, which defaults to three. Additionally, users can provide the maximum transaction fee that they are willing to cover for a particular payment which is taken into consideration when constructing the money circuit.

The trust model discussed in Section 5.5 is based purely on past transactions. It is useful to consider other properties when picking eligible money routers, such as transaction fees, availability, reliability, or network latency. Depending on the situation, one might favour low network latency or competitive transaction fees over trustworthiness. We leave this enhancement to future work.

Transferring Money

We now elaborate the process of transferring money over a n-hop circuit. If $n = 0$, money is sent directly to the beneficiary using exactly one in-house payment and no money routers. A single `sent` transaction is created between the fast payment initiator and beneficiary.

When a money circuit involves one or more money routers ($n \geq 1$), the fast payment is facilitated by intermediaries. Let r_i indicate the i-th router in the circuit (r_1 represents the first router). The initiator starts by sending a message to r_1, containing the payment volume and all subsequent routers involved in the money circuit, including the final beneficiary of the fast payment. Next, the initiator initiates a `commit` transaction with r_1 and sends the money. r_1 now starts to poll for the money and finally constructs a `sent` transaction when funds are observed. r_1 forwards the funds to the next router or the beneficiary and this process repeats until the money arrives in the bank account of the beneficiary. The final transfer to the beneficiary does only result in a `sent` transaction. Thus, a fast payment with n intermediaries results in $2n + 1$ new TrustChain records.

Risk Mitigation

In addition to our trust model, we propose two risk mitigation techniques to reduce counterparty risk when using money routers:

1. *Incremental settlement*: A key risk mitigation technique is to avoid making a single, large payment at once. Instead, a payment is divided into n smaller inter-bank payments. While this approach increases the duration of a fast payment by a factor n, it significantly reduces risk and incentives for intermediaries to compromise money. Also, depending on the volume of a payment, we believe that reduced risk for some increased latency is a desirable trade-off in Internet-of-Money.

2. *Multi-flow payments*: We uniformly divide a fast payment amongst multiple, distinct money circuits. This results in smaller payments through intermediaries and less value at stake. With multi-flow payments, the end-to-end latency of a payment is determined by the slowest money circuit.

While these individual strategies are viable to mitigate counterparty risk, combining them results in a significant reduction of the value at stake, at the cost of additional latency when using incremental settlement and communication overhead. We evaluate the effectiveness of these strategies in Section 5.7.2.

Router Recharging

Since funds arrive in one account and leave another, money routers might become unable to route additional funds at one point in time. This can be addressed by handling fast payments going in the opposite direction, which restores account balances. However, initiation of these fast payments is outside the control of money routers. Balances can

also be restored by initiating a payment from the account with excessive balance to the other bank account. Since this involves an inter-bank payment, settlement might be slow and in turn, this negatively impacts router availability.

This problem is also recognized in off-chain payment networks that are using channels between users with limited capacity. Revive is a mechanism for rebalancing payment channels and avoids the need for (often expensive) transactions on a blockchain to recreate the channel [224]. Alternatively, this issue can be addressed by the route discovery mechanism where routing decisions aim to minimize the amount of rebalancing required while maximizing user profits.

For our mechanism we envision an infrastructure where routers help each other to restore balances, effectively creating a two-sided market with capacity supply and demand. For instance, a router can offer PayPal capacity in return for HSBC funds. Rebalancing payments are handled by the Internet-of-Money mechanism. While this is an efficient method to restore balances, only requiring in-house payments, we consider the design and implementation of such a mechanism as future work.

5.7 Experiments and Evaluation

We now evaluate the performance of money routers, speed of router discovery within Internet-of-Money, and the effectiveness of our trust model.

5.7.1 Performance of Money Routing

We now present experiments that evaluate the performance of fast payments using money routers. All these experiments are conducted with real bank accounts and real money.

Settlement Duration of In-house Payments

To determine the settlement duration of in-house payments for each bank, we send €0.01 ten times between two accounts with different holders, within the same bank. By adding a unique identifier to the description field of a payment, we are able to track payments and accurately measure settlement times. The experiment is executed with two clients on two different computers, with a polling interval of 500 milliseconds, to avoid hammering the bank servers. Polling starts when the payment request has been finished by the sending party. The results are shown in Figure 5.6, with a non-linear vertical axis. Only one bank, ABN AMRO, has sub-second settlement times with an average duration of 320 milliseconds. ING is slower with 1109 milliseconds on average. PayPal and Rabobank show settlement durations that are an order of magnitude slower, averaging to 4.82 and 7.61 seconds respectively. When performing measurements for the Rabobank, we observed a notable outlier with a settlement time of 320 milliseconds. This observation can be explained if we assume that similar internal payments might be handled in different ways by the Rabobank. This experiment demonstrates that in-house payments within the evaluated banks are usually settled within seconds.

International Real-time Money Routing

Next, we focus on the performance of an international fast payment and measure the duration of a money transfer from Rabobank to ABN AMRO, using two intermediary money

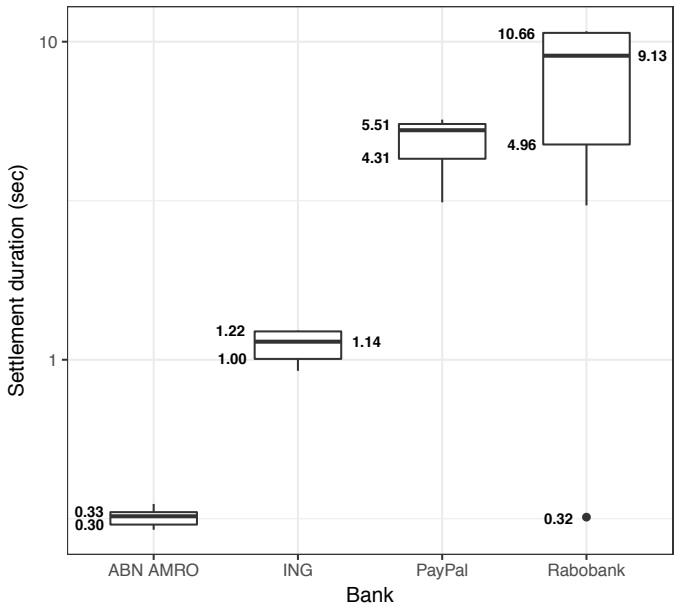


Figure 5.6: Settlement durations of in-house payments for four supported banks.

routers. This experiment aims to show the viability and speed of Internet-of-Money. Figure 5.7 shows the experimental setup and timeline of our experiment.

First, an initiator sends funds from his or her Rabobank account to the first router (holding an account at Rabobank and PayPal), and informs it about the sent funds. Next, the first router starts polling for incoming funds, with an interval of 500 milliseconds. When the first router observes the funds, it forwards them to the second router (holding an account at PayPal and ABN AMRO) and informs this router. When the second router observes the funds, it forwards the money from it's ABN AMRO account to the ABN AMRO account of the beneficiary. In total, three in-house payments are made across six different bank accounts.

From Figure 5.7, we conclude that it takes 15.85 seconds in total for money to arrive in the bank account of a beneficiary when using two intermediate routers. A significant amount of time is spent on waiting for the funds to arrive in the PayPal account of the second router, around 6 seconds or 38% of the total duration. The average time to perform a payment is 2.14 seconds and initiation of payments take 41% of the total duration. The average time that a transaction is in transit is 3.02 seconds. The total time to perform a fast payment is heavily influenced by the number of intermediate routers and their bank accounts. This experiment demonstrates that Internet-of-Money is capable of real-time money routing to other banks.

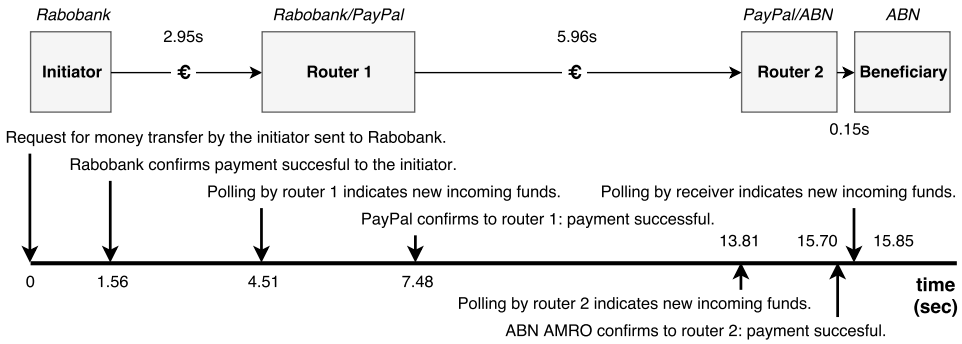


Figure 5.7: Timeline of an international fast payment from Rabobank to ABN AMRO, using two money routers.

5.7.2 Overlay Evaluation

The purpose of the following experiments is to quantify the performance of our money router overlay. This includes an evaluation of our trust model and effectiveness of fraud detection. We implemented our trust model and Internet-of-Money overlay network in the Python programming language. Our implementation is built upon the Dispersy framework, providing primitives for peer discovery, decentralized communication, and secure messaging [225].

Experimental Setup

The following real-world emulations are executed on the DAS5 supercomputer, using 50 instances per node [64]. We deploy our experiment using the Gumby experiment framework [65] and we create a scenario file where we schedule actions at specific times. All code used during these experiments is open source.[3] Due to the limited number of accounts we own and to avoid a large load on the banking infrastructure, we simulate accounts during this experiment. We assume a total of five different banks and devised a basic RESTful banking server that handles account creation, payments, balance queries and mutation requests. Distribution of bank accounts amongst users follows the data as published in the NGData customer banking survey (we assume that every user owns at least one bank account) [223].

Router Discovery

We evaluate the efficiency of the router discovery protocol discussed in Section 5.6. We consider quick bootstrapping in the network important since fast payment initiators might not always remain connected in the Internet-of-Money overlay, e.g., to conserve battery life of mobile devices. During the experiment, we log the connected peers for each user every 5 seconds. At each interval, we determine whether each user is capable of transferring money to all five different bank accounts, using at most one, two and three intermediate money routers respectively.

Figure 5.8 shows the performance of router discovery in the Internet-of-Money overlay. The horizontal axis denotes the time into the experiment. The vertical axis indicates the

[3]See `https://github.com/devos50/gumby/tree/iom_experiment`

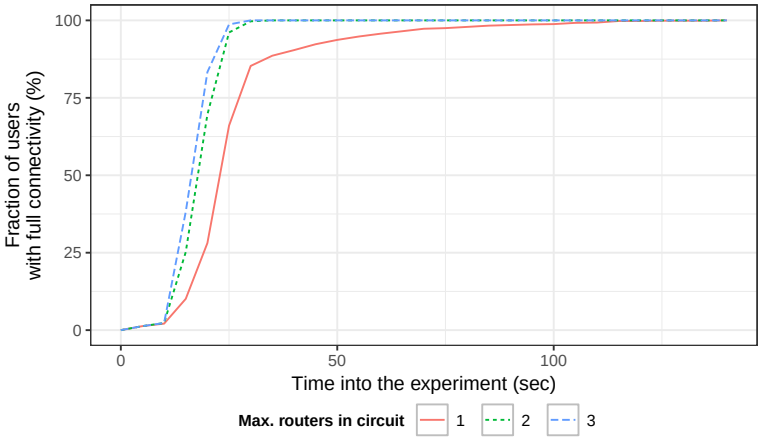


Figure 5.8: Performance of router discovery under a varying number of maximum hops in a money circuit.

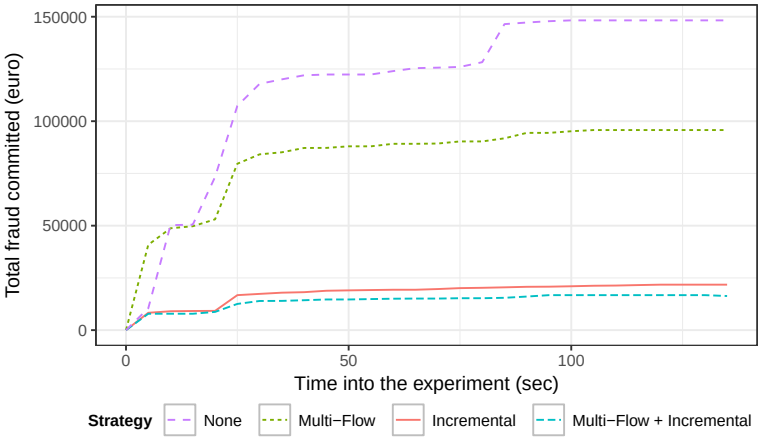


Figure 5.9: The effectiveness of fraud prevention, with different risk mitigation strategies.

percentage of users that are able to make fast payment to all five banks, in other words, are fully connected. We vary the maximum number of routers in a money circuit. As expected, it takes longer before users are able to build circuits to all other banks using only one router, compared to three routers. However, the differences are marginal. In general, router discovery happens fast: 50% of all users are able to make fast payments to all banks within 25 seconds after the experiment starts. 40 seconds into the experiment, this percentage increased to 90%. Note that it takes longer before *all* users are fully connected using at most one intermediate router: 140 seconds.

Fraud Detection
Our final experiment focusses on the effectiveness of fraud detection (see Section 5.5). To this end, we emulated 200 users with one or more bank accounts. Every five seconds, each user with a single account initiates a fast payment to another entity that has exactly one account of a different type. This forces a money router in the established circuits. The volume of each fast payment is picked from a uniform random distribution between €0.01 and €1000. We assume an extremely adversarial scenario where every user with at least two different bank accounts is malicious and has a 50% probability of committing fraud, i.e., not forwarding received funds during a fast payment. To improve router availability, we connect all peers together before the experiment starts. In total, we schedule payments whose volume sums up to €1,251,848.35.

The results are shown in Figure 5.9. The horizontal axis denotes the time into the experiment in seconds, after users start performing fast payments to each other. The vertical axis shows the total amount of fraud committed in Euro. We run the experiment four times with different risk mitigation strategies, namely incremental settlement (we split each fast payment in five equal parts) and multi-flow payments. We average the results across all runs. The figure hints that the amount of fraud is capped and that malicious routers are eventually excluded from money circuits. Without any risk migration strategy, malicious routers are able to steal €1,544 on average during the whole experiment, indicating that fraudulent routers are able to commit fraud multiple times. This can be addressed to the fact that they are included in multiple money circuits roughly at the same time. If we enable risk mitigation strategies, we see that the combination of multi-flow payments and incremental settlement leads to the lowest amount of fraud possible, on average €174. Using exclusively incremental settlement leads to a slightly higher amount of fraud.

5.8 Discussion

We now discuss this research from various perspectives.

Legal
The idea of directly sharing funds with others, without a central bank involved, is highly experimental and challenges existing regulation. Routing money through other bank accounts resembles activity performed by financial settlement institutions and might require a legal prerequisite in the form of a banking licence. The PSD2 regulation states that trusted third parties (TTPs) can be authorized by end users to perform financial activities on their behalf [220]. However, it remains unclear whether the definition of a TTP includes money routers, even after discussion with legal experts. Another consideration is responsibility when a mistaken payment is initiated. At the same time, this consideration also applies to blockchain technology where payments cannot manually be reverted once they are finalized on the ledger. Also, compatibility of our system with (inter)national anti-money laundry regulations is highly uncertain. Exploring legal compliance of this work is a fundamental requirement for further work and additional deployment trials.

Limitations
While we have proven the technical viability of our idea, there are many limitations that must be addressed prior to broader adoption. We noticed that banks are not used to our

dynamic way of initiating money transfers and our accounts got blocked several times due to suspected fraudulent behaviour. An open ecosystem for settlement demands changes by banks and it is an open question whether they are willing to do so, given the conservative nature of the (regulated) financial ecosystem. However, many banks are already forced to innovate their legacy systems to remain competitive [216].

Additionally, we observed that some banks require two-factor authentication when transferring funds to unknown bank accounts. This limits the automation of money transfers since a manual action by the user is required for a payment to proceed. A potential approach is that a user can whitelist trusted accounts and use these accounts as subsequent hops in a money circuit. This approach is related to Interledger, a payment protocol to transfer value through trusted *connectors* across isolated ecosystems [59].

Privacy
We consider privacy an important requirement of our open platform and expose minimal information about money flows. The current privacy model in Internet-of-Money is effective but open for extension. Decentralized path-based transaction networks, for instance, SpeedyMurmurs, can be leveraged to address this specific problem [226]. In addition, we can draw inspiration from privacy-preserving coins like Monero, and their adopted cryptographic techniques, such as ring signatures and zero-knowledge proofs [108, 109].

Scalability
Our overlay network has the potential to scale, mostly since it avoids the need for global consensus. However, techniques like incremental settlement lead to additional payments and a higher load on the banks. In addition, the choice of reputation mechanism used in Internet-of-Money influences scalability.

5.9 Related Work

The last few years, there has been a steep increase in Fintech start-ups, eager to disrupt existing financial services. Hawala is an informal system to transfer value, without actually moving money [227]. It consists of a network of hawala brokers, that take a small commission. In contrast to our system, trust in hawala is cultivated in an analogue manner whereas our model depends on a digital solution.

Innovation in the financial sector has been catalysed by the popularity of Blockchain technology, aiming to build trust between strangers without involvement of centralized authorities. Bitcoin has proven that a sustainable currency can be built without a central bank in control [1]. However, wide-spread adoption stays out due to its volatile pricing, high transaction fees, relatively slow confirmation times, and unsure future. The Lightning Network aims to improve scalability of Bitcoin by providing bi-directional payment channels between users [228]. Payments between two users not directly connected with a payment channel, are realized by routing payments through channels of other users. This has similarities with money routing in Internet-of-Money. New usages of blockchain technology are focussed around the way users transfer money and other assets. The Ripple project, supported by various major banks, attempts to build a connected network of financial institutions and payment providers [229]. Their solution aims to significantly speed up traditional money transfers, lower costs, and provide support for high-volume

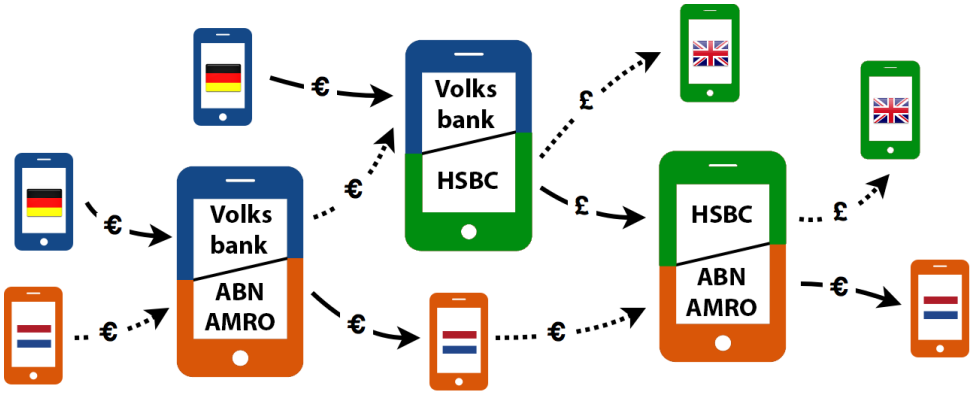


Figure 5.10: Fast international money transfers and currency conversion using Internet-of-Money. Distinct money circuits are indicated by different arrow styles.

transactions. R3 Corda can be compared to TrustChain since they share the idea that a ledger with global consistency is often not necessary [32].

While blockchain solutions are slowly being adopted, the aforementioned systems all aim to increase utility by building a financial network from scratch. In comparison, Internet-of-Money is built upon existing, proven infrastructure, making migration towards our system effortless.

5.10 Conclusions

We explored a new stage in the evolution of digital trust and addressed the problem of trusting strangers with your money. The tamper-proof TrustChain structure provides a scalable and public trace of historical interactions, and allows detection and punishment of potential fraud. We expand upon this with an overlay network to transfer money within seconds to others, using other network participants as financial intermediaries. This mechanism depends on the fast settlement of in-house payments. Our open ecosystem dramatically improves speed when initiating cross-border payments while preserving privacy and scalability. Our experiments demonstrated the efficiency of in-house payments and effectiveness of money routers. Additionally, we have proven that our fraud detection mechanism, together with incremental settlement and multi-flow payments, limits misuse and punishes malicious behaviour. However, there are various legal issues and limitations that should be addressed, mostly by financial institutions, before broader usage can be realized.

This work is an important milestone in our ambitious vision to create the *programmable economy*. Ongoing work towards this goal addresses self-sovereign identity, scalable blockchain consensus compatible with TrustChain, and decentralized marketplaces. Upcoming experimentation will focus on expanding Internet-of-Money to support additional banks, currency conversion, and international bank transfers. For this experiment, we utilize additional type of money switches to send money across countries in seconds. The experiment setup is visualized in Figure 5.10 and is the first step towards fast and trusted international payments.

6

dAppCoder: A Decentralized Marketplace for dApp Crowdsourcing

Decentralized applications, also known as dApps, are the new paradigm for writing business-critical software. Crowdsourcing the development of such applications is gaining popularity. At the same time, finding developers with appropriate qualifications and skills for this activity is key, yet challenging. The main problem is that the portfolio of developers is usually scattered across centralized crowdsourcing platforms and vendor locked-in. This can result in an incomplete impression of their capabilities.

In this chapter we address these problems and first introduce a unified, blockchain-based portfolio for developers, named DevID. *Over time, a DevID portfolio enables developers to build up a trustworthy collection of records that showcase their capabilities and expertise. They can import data assets from third parties into their portfolio, add projects and skills, and receive endorsements from others. All portfolio records are stored on an existing, scalable ledger, named TrustChain, and managed by developers themselves. TrustChain enables tamper-proof data accounting with low overhead and without network-wide consensus.*

We then build a decentralized crowdsourcing marketplace for the development of dApps, named dAppCoder. *dAppCoder allows clients to publish projects and developers can find work, all without trusted intermediaries. dAppCoder utilizes DevID portfolios to match these clients and developers. We fully implement DevID and dAppCoder, and conduct a deployment trial. Our trial demonstrates that DevID and dAppCoder are efficient at storing portfolio records.*

6.1 Introduction

Decentralized applications, also known as dApps, enable the deployment of business logic that runs without the need for trusted intermediaries [230]. At the time of writing, there are thousands of decentralized applications deployed on numerous blockchain solutions, most of which are on Ethereum. The dApp ecosystem is continuously expanding and recently has been accelerated by the rise of decentralized finance (DeFi) solutions [231].

Since dApps usually require a high level of security and robustness, the engineering of these solutions is considered a challenging task and there is a shortage of talent capable of building them [232]. For this reason, crowdsourcing the development of decentralized applications is becoming an increasingly popular alternative to training in-house talent [233]. Crowdsourcing is a relatively new model for software development where an open call is made for the documentation, design, coding, and testing of software [234]. Matching clients and developers is at the core of numerous intermediary crowdsourcing markets such as Upwork and Fiverr. These profit-driven platforms usually charge a commission for their services, e.g., by taking a cut of all payments between a client and a developer.

Unfortunately, existing crowdsourcing platforms only provide access to a subset of all available work and developers. Specifically, the key problem is that centralized market approaches for client-developer matching lead to *fragmentation* and *lock-in* effects [217, 233]. Many software developers have their portfolios fragmented across multiple platforms like TopCoder, GitHub and LinkedIn. Each platform only yields a partial impression on the capabilities and background of a developer, making it challenging for a client to make an educated decision on who to hire for a particular task. Additionally, data assets are usually locked to one platform and cannot easily be exported across different services, complicating this matchmaking process even more [235].

There currently is no independent platform for dApp crowdsourcing without fragmentation and lock-in effects. We argue that the availability of such a platform would increase efficiency and effectiveness when matching reputable developers looking for work and clients that are in need of talent. In other words, such a platform would reduce *search frictions*, which are impediments to finding matches between clients and developers. We address this deficiency in this work. As a first step, we design *DevID*, a unified portfolio for software developers. We believe that such a portfolio is a key step towards a more efficient crowdsourcing marketplace, since highlighting the expertise of a developer is instrumental during the client-developer matchmaking process. DevID portfolios enable developers to showcase their expertise, making it easier for prospective clients to get an overview of their capabilities. An impression of a DevID portfolio is given in Figure 6.1. DevID portfolios are powered by a scalable blockchain ledger, used for durable storage of records. Developers can add tamper-proof records to their portfolio, import existing data from external platforms, and add references to their prior activities, e.g., projects on GitHub. DevID records are stored in a peer-to-peer fashion without central authority.

We then build a decentralized crowdsourcing marketplace for the development of dApps, named *dAppCoder*. Our platform enables clients to post projects, and developers to find projects that match with their expertise. All payments are conducted directly between clients and developers, sidestepping the need a trusted intermediary for payment processing and avoiding commissions. We believe that our single, public, and open crowdsourcing market has the potential to be more efficient compared to a centralized solution

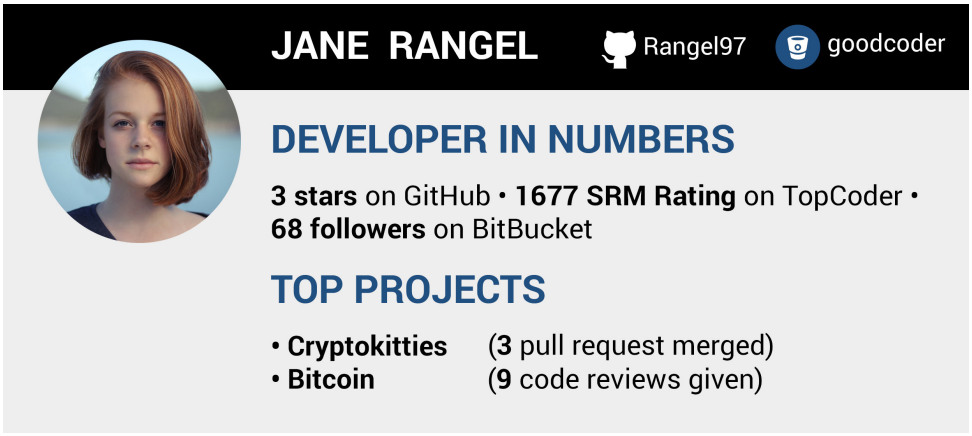


Figure 6.1: An impression of a DevID portfolio. These infographics can be automatically generated and customized based on records in a DevID portfolio.

with fragmentation and lock-in effects.

The main contribution of this work is three-fold:

1. We first present *DevID*, a unified portfolio for dApp developers (Section 6.3). DevID stores records associated with a developer, e.g., GitHub projects, on a scalable blockchain ledger with high scalability and low overhead.

2. We then present *dAppCoder*, a decentralized crowdsourcing market for the development of dApps (Section 6.4).

3. Finally, we present a *deployment trial* of DevID and dAppCoder with a small group of users, which demonstrates the practicality of our work (Section 6.5).

6.2 Problem Description

We identify technical challenges in two directions. First, we wish to create a developer portfolio that gives an accurate impression of their capabilities and expertise. Second, we aim to build a decentralized crowdsourcing marketplace for the development of dApps. We elaborate on the challenges associated with each contribution.

Developer portfolios. We list three main requirements for this portfolio. First, we require that developers are able to *import existing data* from other platforms into their portfolio. This streamlines the bootstrapping process of a portfolio with relevant records since a developer is likely to use multiple platforms to store and work on their projects (e.g., GitHub and LinkedIn). However, not all platforms have built-in tools to easily export personal data to another ecosystem. Additionally, it is key to ensure that data being imported actually belongs to the user importing it, preventing a malicious user from impersonating someone else. We consider the import process of personal data an essential requirement to ensure trustworthy portfolios.

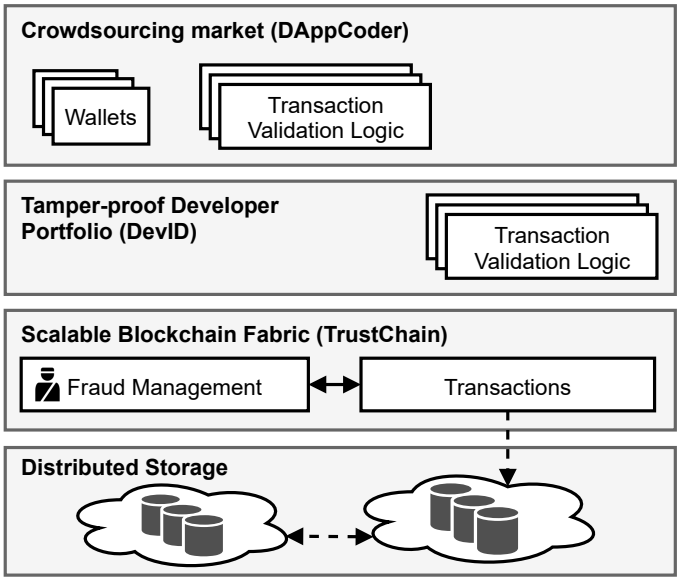


Figure 6.2: The architecture of DevID and dAppCoder.

Second, we require portfolio records to be tamper-proof. Specifically, we want to avoid the situation where a developer temporarily inflates its portfolio when applying for a particular project. Modifications to a DevID portfolio should be transparent and traceable.

Third, we require our developer portfolio to be *independent of any trusted intermediary*. Specifically, in the context of our work this means that the data itself is not managed by a trusted authority but instead by the operating user itself. Blockchain technology is increasingly being used as middleware for building decentralized applications without centralized authority. For example, platforms like Ethereum and EOS enable developers to write and deploy smart contracts, self-executing code that enforces agreements between two or more parties [236]. However, most blockchain fabrics are not suitable for large-scale storage of portfolio records due to their limited throughput and, for some platforms, prohibitively high transaction fees [237]. Therefore, we prefer an alternative, more lightweight solution to store portfolio records.

Crowdsourcing Market. Similar to the requirements of our developer portfolios, a key requirement for our crowdsourcing market is that there is no intermediary responsible for creating and managing projects, and for matching clients with developers. Instead, all information should be managed in a decentralized manner, namely by peers in the network themselves.

Given these requirements, the research question of this work is as follows: *How can we build a decentralized crowdsourcing platform for the development of dApps?*

6.3 DevID: Unified Portfolios for Software Developers

We now present our unified developer portfolio, named *DevID*. The architecture is given in Figure 6.2. This figure also includes the architecture of dAppCoder, our platform to crowdsource the development of decentralized applications. dAppCoder itself is discussed later in Section 6.4.

6.3.1 Record Types

DevID distinguishes between the following seven record types:

Personal Information. This record contains personal information associated with the developer, for example, a full name, a profile picture, an email address, or a GitHub username.

Statistics. Statistics are quantifiable and verifiable numbers that represent a specific developer metric. For example, these records could represent developer statistics like the number of years of programming experience, or the total number of code reviews given. A visualization of these records is shown in Figure 6.1 under the section "Developer in Numbers".

Projects. Developers can add projects to their DevID portfolio, for example, open-source software projects where the developer has contributed to. In Figure 6.1, this information is displayed under the section "Top Projects". Optionally, a reference to a project can be added to a DevID portfolio, such as a link to a GitHub repository or to the hash of a particular commit.

Skills. Developers can add skills to their DevID portfolio. We consider the ability to highlight proficiency in specific programming languages and familiarity with blockchain platforms an essential feature of a developer portfolio. It aids programmers in finding projects that match their expertise, and it enables clients to find developers that fit their projects best. For instance, applications that have access to DevID portfolios can filter available developers on one or multiple skills, therefore reducing search frictions.

Endorsements. Another record type we define is endorsements. Developers can endorse other developers (e.g., by writing a letter of recommendation) or endorse specific skills of others. Skills and endorsements can also be imported from other platforms like LinkedIn, which is discussed in the next section.

Import. Developers are able to import data from other sources into their DevID portfolio. When performing an import action, a record with this type is created, containing specifications on the import source and the imported data elements. Other records can reference the import record, for example, to signify that a particular skill endorsement has been imported from LinkedIn. We provide more technical details on records in Section 6.3.5.

Wallets. Finally, developers can add wallet information to their DevID portfolio, for example, the address of their Bitcoin wallet. We envision that DevID portfolios will be utilized when developers are looking for work, and embedding payment addresses directly in the portfolio speeds up the payout process by the client when a developer has work on a project. It also enables developers to quickly receive donations as compensation for community work.

We believe that the above record types are sufficient for developers to build a basic portfolio. The implementation of DevID is flexible and allows system designers to add

additional record types after deployment.

6.3.2 Unifying Developer Data

We now discuss how to import developer data from multiple platforms in a DevID portfolio and how to verify the correctness of the imported data.

Importing Developer Data. DevID allows developers to import relevant information from different platforms into their portfolio. For example, they can import data from LinkedIn (e.g., skills or past projects) or from GitHub (e.g., the number of followers or significant code contributions). Importing can be done by querying the public interfaces (APIs) of external platforms and requesting the relevant data, if the platform offers this functionality. To store the data in the portfolio, one can either add a reference to the (external) data or copy the data assets into the portfolio. To reduce dependency on third-party services, we choose to copy the data into a DevID portfolio and store it within records.

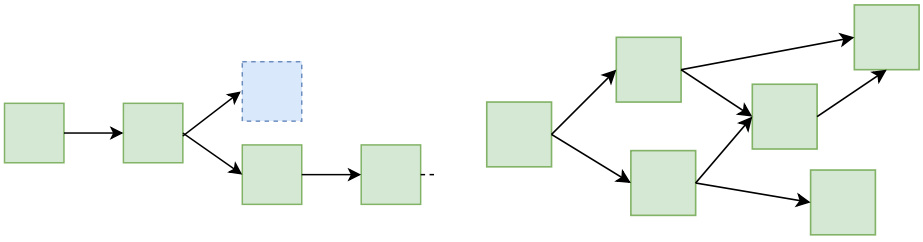
Verifying Developer Data. As discussed in Section 6.2, it is essential to ensure that imported data actually belongs to the developer importing it. We propose two solutions to achieve trustworthy importation of data: *challenges* and *TLS auditing*.

The first solution is to pose a challenge that has the developer importing the data prove that they have control over this data. For example, when importing data from GitHub, we can require a public identifier (e.g., a public key) of the developer to be part of the "bio" profile field on a GitHub profile. This information can then be verified for correctness by other users who query the public GitHub API. While this is a basic mechanism to ensure the accuracy of imported data, it heavily depends on the availability of a public API.

The second solution is TLS auditing [238]. The key idea is to proxy a TLS connection through a random witness, which then verifies and signs the data after the TLS connection terminates. When the TLS session finishes, the client gives the witness the private key used to decrypt HTTPS responses from the web service. Note that this way the witness is not able to decrypt the request made to the web service, which likely includes credentials or access tokens. The role of a witness can either be fulfilled by other entities in the network, or by a trusted notary service. Depending on the significance of data being imported, multiple witnesses can be used for this. Compared to challenges, TLS auditing works when access to a public API is absent but is more advanced. Our lab has implemented an advanced TLS auditing mechanism, which is currently under a security audit.

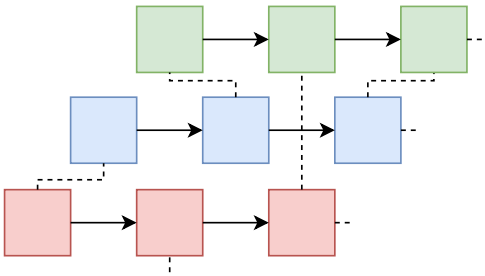
6.3.3 Verified Identities

In DevID, users are identified by a self-generated public key and digitally sign portfolio records with their private key. This allows for the Sybil Attack, the situation where a real-world persona can operate many DevID portfolios [101]. To address this situation, developers can verify their digital identity. A verified identity is uniquely linked to a real-world entity. Software built on DevID can give preferential treatment to developers that have verified their identity. For example, an application can ignore endorsements that are given by developers with an unverified identity, and clients can only consider developers with a verified identity for a particular project. The user interface of dAppCoder highlights verified users with a special badge. Identity verification can be done with an attestation provided by a trusted third party like the government or a notary. This attestation is then



(a) Linear ledger (Ethereum).

(b) DAG ledger (IOTA).



(c) Pairwise ledger (Nano).

Figure 6.3: Three different structures of distributed blockchain ledgers. Each arrow points to the subsequent block in the chain. The dotted block in (a) indicates a fork.

included in the DevID portfolio with a separate record. Strong, long-lived identities in DevID is comparable with account validation that many centralized platforms use (e.g., the verification of a phone number). Since misbehaviour can be traced back to the real-world persona, it also raises the barrier for users to collude with each other, e.g., tit-for-tat behaviour when creating endorsements.

6.3.4 Efficient Blockchain Storage

DevID requires a data structure that can store tamper-proof data records. Blockchain technology is gaining traction as platform to store verified transactions without central authority. We now discuss three common blockchain organizations, visualized in Figure 6.3, and analyse their trade-offs in the context of our work.

Linear Ledger. Figure 6.3a shows a linear blockchain ledger which is used by platforms such as Bitcoin [1] and Ethereum [2]. This ledger consists of multiple blocks and each block contains transactions. Every block, except for the first one, points back to the prior block in the ledger through a hash pointer. Usually, this type of ledger is safeguarded by a consensus mechanism where at least a majority of users continuously reach agreement on the exact sequence of transactions. A network-wide consensus mechanism like Proof-of-Work or Proof-of-Stake prevents the situation where a malicious user intentionally creates a fork of their chain to override prior transactions [111]. While linear ledgers provide a relatively high level of security, the transaction throughput of these ledgers is often not sufficient to facilitate record creation and modification by millions of users. This motivates us to consider different types of blockchain structures for portfolio storage.

DAG Ledger. Another blockchain structure is the Directed Acyclic Graph (DAG) ledger where each block can be referenced by multiple other blocks. This ledger structure, shown in Figure 6.3b, is adopted by blockchain platforms like IOTA [239] and Dagcoin [240]. IOTA is optimized for micro-payments within Internet-of-Things and Dagcoin advertises itself as data storage for arbitrary data (e.g., documents or ownership records). Since these ledgers allow for different consensus mechanisms, transaction throughput is often superior compared to that of linear ledgers. However, they usually have differing security guarantees. While these ledger structures are more suitable for data storage compared to linear ledgers with network-wide consensus, we consider current implementations unfit for developer portfolios. The reason is that they either rely on a centralized coordinator (IOTA) or a fixed group of witness nodes (Dagcoin). Instead, our goal is to devise a portfolio infrastructure without any authority with leveraged permission.

Pairwise Ledger. A third blockchain structure we consider is the pairwise distributed ledger. The key property of this ledger, given in Figure 6.3c, is that each user maintains and grows their individual chain with transactions. Each block holds exactly one transaction and optionally contains a (hash) pointer to a transaction in the individual chain of another user. Blockchain fabrics like R3 Corda [32], Nano [241], and TrustChain [98] use pairwise ledgers as their underlying data structure. These platforms address the double-spending attack either by a trusted notary (Corda), a weighted voting system (Nano) or by guaranteed eventual consistency (TrustChain). In general, they can provide superior scalability compared to linear ledgers as used by Bitcoin and Ethereum.

We believe that the pairwise distributed ledger is a suitable data structure to store portfolio records as transactions. Compared to linear and DAG ledgers, all data associated with a portfolio owner is stored in their own personal ledger and maintained by owners themselves. Specifically, we choose to build DevID, and subsequently dAppCoder (see Section 6.4), using the TrustChain data structure.[1] TrustChain, first introduced by Otte et al. [98], is a lightweight distributed ledger where each user maintains a grow-only personal ledger. To detect malicious behaviour, in particularly the forking of a personal ledger, users continuously request random transactions from other users and share their transactions with other users upon request. The consistency of incoming transactions is checked against known transactions, and illegitimate modifications of the personal ledger can quickly be revealed by the collective effort of users in the network.

TrustChain has four particular advantages that align with developer portfolios. First, TrustChain enables selective queries of data stored on the chains of other members, without the need for full data replication across the network. Second, TrustChain is particularly designed for the tamper-proof accounting of generic data elements. Third, while TrustChain is primarily built around bilateral transactions, this ledger architecture also supports unilateral transactions. Unilateral transactions are particularly helpful when storing content that is not related to other users, e.g., when a developer includes a project in its portfolio. Finally, TrustChain is already used as transaction fabric within a self-sovereign identity system, described in the work of Stokkink et al. [105]. Availability of a self-sovereign identity system aligns with our requirement for strong, long-lived identities (see Section 6.3.3). TrustChain, however, does provide less consistency guarantees.

[1]The work describing TrustChain considers a personal ledger as a collection of *records*. To avoid semantic overload, we use the term transaction to refer to a TrustChain record in this chapter.

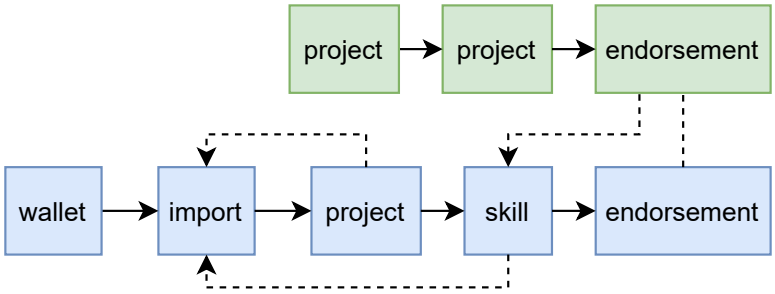


Figure 6.4: Example of DevID records and links between them on two TrustChain personal ledgers (coloured differently). Solid arrows indicate hash references between transactions whereas dashed arrows indicate application-specific references included in the transaction payload.

At the same time, we believe that this does not pose a barrier for DevID (depending on the system parameters and network size, malicious behaviour such as removing a transaction from a personal ledger can usually be detected within seconds).

6.3.5 DevID records and TrustChain

TrustChain enables users to issue transactions with an application-specific payload. These transactions are appended to the personal ledger of the operating peer and shared with others. TrustChain also enables system designers to specify validity rules for different transaction types. A TrustChain transaction has a `type` field, which we fix as the record types discussed in Section 6.3.1 with `devid_` as prefix. The transaction type is used by participating users to distinguish between different applications and to conduct application-specific validation of incoming transactions (since different applications operate on the same TrustChain infrastructure). The transactions containing records associated with projects, skills, import actions, and wallet information are unilateral. Endorsements are implemented as bilateral transactions since they involve an interaction between two users. However, endorsements do not strictly require a counter-transaction from the user being endorsed. An endorsement contains a reference to the transactions containing the skills being endorsed.

Figure 6.4 shows how DevID records are mapped on the TrustChain data structure. The figure shows (a part of) the personal ledger and TrustChain transactions of two different users a and b, in green and blue colours, respectively. User a added two projects to its portfolio and provided an endorsement to user b. User b added information about a particular wallet to its portfolio and subsequently imported data from an external source. This import action added a single project and skill to the portfolio of user b. The transactions associated with the project and skill reference the transaction containing the import details.

Users joining the network continuously request TrustChain transactions from other users and consequently build up knowledge of the DevID portfolios of others. Upon receiving TrustChain transactions, users will invoke an application-specific validation process that depends on the transaction type. This involves a check whether the transaction payload is well-formatted and contains all expected fields. Some transaction types require more extensive validation, for example, validating `import` transactions. Since there are no

guarantees on the order in which TrustChain records arrive, the validation of transactions that are linked to an import action (e.g., projects) is delayed until the `import` transaction has been received and verified.

6.3.6 Storing Large Data Assets

While pairwise distributed ledgers are suitable for storing small portfolio records, they are not suitable for storing arbitrary large data assets. Such data assets can include source code, documentation, videos, and reviews. To overcome this, we leverage off-chain distributed storage solution that offers data immutability and scalability. Figure 6.2 includes this distributed storage, which comprises the lowest layer in our architecture.

We wish to avoid data storage by a trusted operator. Suitable distributed storage solutions for our work are a Distributed Hash Table (DHT) like Kademlia, a BitTorrent swarm or the InterPlanetary File System (IPFS) [37, 39, 242]. These solutions enable users to store large data assets, without involvement of a trusted third party. Large data is inserted in the distributed storage back-end, and a reference to the data (i.e., a content hash) is included in the on-chain transaction. DevID (and dAppCoder) users can store their data at different *providers*. DevID portfolio records with external data assets attached include the provider identifier (e.g., "`IPFS`") and a pointer to the data (e.g., an IPFS content hash).

6.4 dAppCoder: Crowdsourcing the Development of dApps

By extending the DevID portfolio architecture, we build a crowdsourcing marketplace, named *dAppCoder*, for the development of dApps. Running completely without centralized servers, dAppCoder enables clients to propose projects and to find developers qualified to work on their projects. The process of finding developers is streamlined by direct integration of DevID portfolios, therefore reducing search frictions. dApp developers can choose to work on projects that match their skill set. dAppCoder also provides primitives for project management and financial compensation for developers. The architecture of dAppCoder is presented in the upper layer of Figure 6.2. We now elaborate on the main functionalities of dAppCoder.

6.4.1 Creating and Managing Projects

Clients that want their idea realized (e.g., the implementation of a particular smart contract) can offer a new project in dAppCoder. Creating a new project requires the client to specify various fields, as exemplified in Listing 6.1. Each project includes a title, a description, a list with technical requirements for the final deliverable, a deadline (optional), a list of required skills needed to successfully complete the project, the height of the financial compensation (if any), and an optional list of assets provided by the client. Since the project description, requirements, and assets can become large, these assets are stored off-chain and a reference to these assets is included in the transaction. This reference contains a `provider` and `uri` field. To publish a project, a TrustChain transaction with type `dappcoder_project` containing all project information is constructed, appended to the personal ledger of the project creator, and disseminated in the network. Clients can publish projects that concern both open-source deliverables (e.g., a publicly deployed Ethereum smart contract) and closed-source deliverables (e.g., a smart contract deployed in a private

Example 6.1: A project offered in dAppCoder (in JSON format).

```
{
        "title": "An Ethereum-based Art Marketplace",
        "description": {
                "provider": "IPFS",
                "uri": "..."
        },
        "requirements": {
                "provider": "IPFS",
                "uri": "..."
        },
        "deadline": "20-09-2021",
        "required_skills": ["Solidity", "Ethereum"],
        "compensation": {
                "height": "...",
                "deposit": "...",
        },
        "assets": [{
                "description": "example art objects",
                "provider": "IPFS",
                "uri": "..."
        }]
}
```


blockchain). dAppCoder supports direct payouts between clients and developers using cryptocurrency. This avoids the need for trusted intermediaries for payment processing.

As shown in Figure 6.5, a dAppCoder project can be in one of the five following stages: *created*, *implementation*, *testing*, *finished*, or *cancelled*. After a project has been created, developers can indicate their interest to work on a preferred project by creating and sharing a unilateral `dappcoder_project_interest` transaction. A client is still able to cancel projects that have not advanced to the implementation stage. If many developers have indicated their interest to work on a particular project, the client can filter developers, for example, based on the information in their DevID portfolios. A project is updated using a `dappcoder_project_update` transaction created by the project creator that transitions the project to a different stage. The dAppCoder software has built-in validation rules that assess whether a project transaction is valid. For example, a project cannot transition from the *cancelled* to *created* stage and transactions containing such a transition are ignored.

During the implementation stage, assigned developers work on implementing the project deliverables. When the implementation is complete, the project enters the *testing* stage. Testing is a crucial requirement for the development of secure decentralized applications that often involve value management or business-critical operations. A single bug has the potential to bring down an ecosystem that manages billions worths of assets, as demonstrated by the DAO hack in 2016 [243]. This testing might be performed by the developers that worked on the project deliverables during the implementation stage, or by other developers. If the testing stage reveals that more work is needed on the project deliverable, the project can transition to the *implementation* stage again. This is ultimately determined by the client. When the deliverable is satisfactory, the client indicates that the

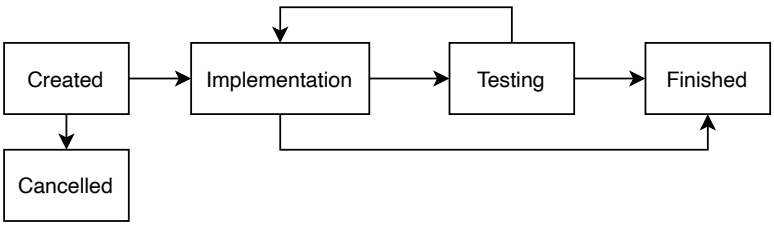


Figure 6.5: The different stages of a project in dAppCoder, and their transitions.

project is finished and conducts payments to the involved developers if there is a financial compensation involved.

6.4.2 Finding Projects and Publishing Deliverables

Developers looking for work can browse through the list of open projects and filter them based on the skills they have added to their DevID portfolio. When a developer has found an interesting project, they indicate their interest to work on the project. This work starts when the client assigned the interested developer to the project and when the project transitions to the implementation stage. If the source code of the project can be made public, a developer may create a `dappcoder_deliverable` transaction that includes a pointer to the work done. This pointer, for example, can refer to a GitHub repository. Such a transaction also links to the project associated with the submission.

6.4.3 Paying Out Developers

dAppCoder has built-in tools to directly payout developers that worked on a paid project. The time at which these payouts take place should be decided between the developer and client using an out-of-band communication channel. For example, a developer can request to be partially compensated up-front. We envision that all payouts in dAppCoder proceed using cryptocurrencies and are public. A developer can signify a payout by creating a `dappcoder_payout` transaction, containing a reference to the cryptocurrency transaction associated with the payout (e.g., the hash of a Bitcoin transaction). By verifying the source and destination addresses of the transaction with the wallet information included in DevID portfolios, other users can determine whether a client correctly compensated a developer. Unreliable clients that have no valid payout associated with a finished and paid project will be flagged in the application and can be avoided by developers.

6.5 Implementation and Deployment Trial

Next, we elaborate on the implementation of both the DevID portfolios and the dAppCoder application. We also discuss our deployment trial and present preliminary results.

6.5.1 Implementation

We have implemented both DevID and dAppCoder in the Python programming language. Our implementation consists of all components shown in Figure 6.2. The graphical user interface of dAppCoder is visualized in Figure 6.6 and is implemented with the Qt5 library.

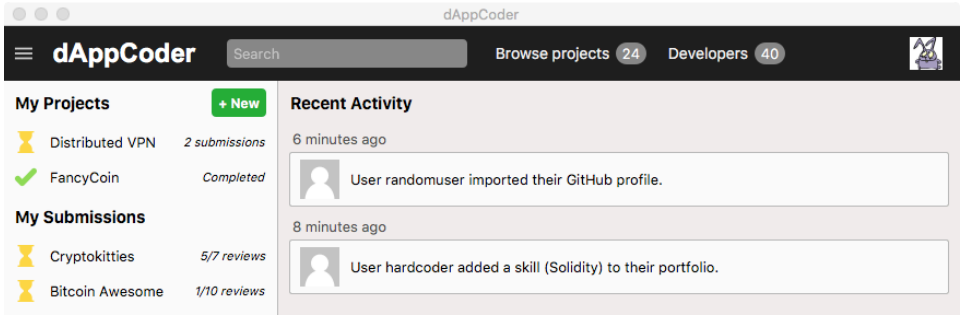


Figure 6.6: The user interface of dAppCoder, our application to crowdsource the development of decentralized applications.

It communicates with the back-end over a RESTful API. The open source implementations of both DevID and dAppCoder are available online.[2]

We build DevID, and by extension dAppCoder, on the TrustChain ledger introduced by Otte et al. [98]. This ledger is built on our existing networking library that provides support for building decentralized overlay networks [63]. We use the InterPlanetary File System (IPFS) to store large data assets like project specifications, submissions, and code reviews. Users can import statistics from their GitHub profile using the challenge mechanism described in Section 6.3.2.

6.5.2 Deployment Trial

To assess the feasibility of dAppCoder and to get insight into the efficiency of the TrustChain ledger, we conduct a deployment trial. We present the trial setup and results.

Setup. For our trial, we recruited 15 participants among local staff and students of our faculty. Each participant interacts with the dAppCoder application for around 15 minutes and during this time, participants were free to use the application as they see fit. To bootstrap the application, we initiated dAppCoder with five unpaid projects ourselves. Two of these projects asked developers to resolve one or more bugs in a piece of Python code. The other three projects asked the developer to implement a small application. Since only a fraction of our users is familiar with the development of decentralized applications, we accepted submissions in other programming languages during our trial, like Java. We collected data and observed the growth of the distributed ledger over a period of five working days.

Results. Figure 6.7 shows the growth of the TrustChain ledger, in terms of transactions, when more participants join the trial. Each entry on the horizontal axis represents the state of the ledger after a participant was introduced, and the vertical axis shows the transaction count for the six different types of transactions. When more participants join, the distribution of transaction types on the ledger changes slightly. We observe that the growth of projects over time decreases, and users focus more on creating submissions and reviews. We remark that the version of dAppCoder subjected to the user trial oriented around competition-based crowdsourcing and participants are able to review the submis-

[2]See `https://github.com/tribler/dappcoder`

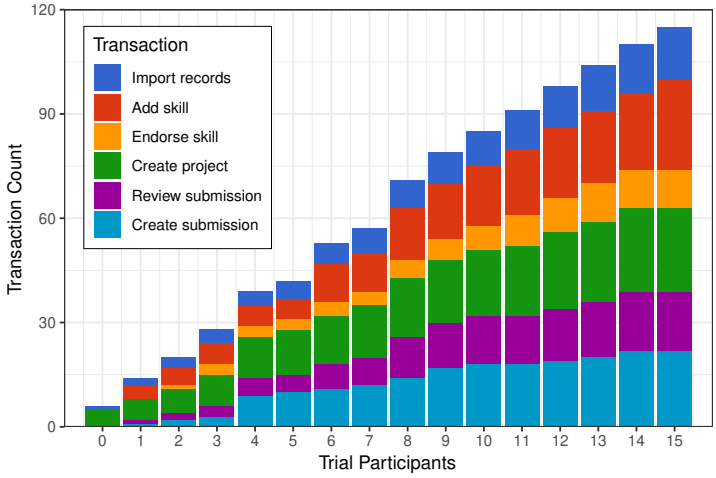


Figure 6.7: Results from our deployment trial.

sions of others. Another observation is that the number of skills added by each developer grows rapidly, but the growth of endorsements stays behind.

At the end of our deployment trial, the average transaction size in serialized form is 0.6 kB. The total size of all transactions stored on the distributed ledger is 65.4 kB. Each personal ledger stores on average 7.2 transactions, with an average size of 4.1 kB. In comparison, when using a linear ledger like Bitcoin, each user is required to store the entire global ledger or parts of it. The time required to append new transactions to the TrustChain ledger is in the range of milliseconds and not of influence on the user experience. The initial results of the trial look promising, and we are ready for further evaluation of dAppCoder and DevID portfolios.

6.6 Related Work

We are the first to build a tamper-proof and unified developer portfolio, to the best of our knowledge. Already in 1995, research has been conducted, that explores the advantages of online electronic portfolios over traditional resumes, particularly within an educational environment [244, 245]. The emergence of the open source software paradigm enabled developers to use code contributions as proof of verifiable technical expertise and to build an online reputation [246]. The work of Cai et al. explores how this data can be used to construct a theoretical reputation model, and what metrics would be best suited for this [247]. Other work is focused on visualization tools to highlight contributions of the individual developer on platforms like GitHub or StackOverflow [248–250]. Their research is primarily focused on the design and evaluation of models to represent the technical skills, based on data from open source projects. The focus of this work is on combining records from different platforms and presenting them in a unified portfolio.

The evolution of crowdsourcing and the benefits are well-studied topics with an extensive literature corpus [234]. TopCoder Inc. is an example of a crowdsourcing platform

where clients can outsource software contributions to developers in a competition-based environment [251]. In 2017, Li et al. introduced CrowdBC, a decentralized blockchain-based framework for crowdsourcing [252]. CrowdBC is a platform to crowdsource generic micro-tasks and is not suitable to crowdsource development of decentralized applications. Lu et al. devised a privacy-preserving crowdsourcing mechanism on top of an open blockchain [253]. Zou et al. present a consensus protocol that is suitable for crowdsourcing tasks [254]. The protocol selects transaction validators and addresses unfaithful behaviour when participating in the system. Buccafurri et al. introduce TweetChain and show how to build a crowdsourcing application which stores all information on Twitter timelines [255]. TweetChain is comparable with personal ledgers in TrustChain but depends on a central authority for dissemination and storage of data (Twitter). In comparison to most of the research performed on blockchain-based crowdsourcing, this work focuses on a specific use-case, namely crowdsourcing the development of business-critical applications.

6.7 Conclusions

We have presented dAppCoder, a decentralized crowdsourcing platform for the development of decentralized applications. As a first step, we built DevID, a unified portfolio for software developers. DevID addresses the fragmentation and lock-in of developer data across different platforms with a mechanism to import data from third-party services. By building upon an existing scalable ledger, DevID is capable of storing tamper-proof records and does not depend on any trusted party. Portfolio records are fully managed by developers themselves. Our decentralized crowdsourcing marketplace, dAppCoder, enables clients to create projects and developers to find work that matches with their expertises. With a deployment trial, we have demonstrated that both DevID and dAppCoder are efficient at storing data.

Future work is focused on a large-scale deployment of dAppCoder and better support for specific bug bounties. We envision the integration of multiple cryptocurrencies and external data providers. Using our TLS auditing mechanism, we plan to expand DevID with the ability to import data elements from other platforms, in particular, LinkedIn and StackOverflow. Finally, we aim to explore the use of DevID within other domains besides crowdsourcing.

7

Conclusions

In this thesis we have introduced five novel mechanisms to decentralize and disintermediate all aspects of blockchain-based marketplaces. Using our universal ConTrib accounting mechanism, market information can securely be stored in a decentralized manner by peers themselves. With our decentralized MATCH middleware, participants do not have to rely on a centralized matchmaker to match their orders in peer-to-peer markets. Our XChange trading mechanism enables asset trading between permissioned blockchains without any requirement for a trusted third party that performs settlement. The decentralized Internet-of-Money overlay enables fast and international money transfers without settlement by a central bank. Finally, software developers using DevID can build self-hosted, durable portfolios without their data being managed by a central party can and showcase these portfolios on the decentralized crowdsourcing platform dAppCoder.

7.1 Conclusions

The main conclusions of this thesis are as follows:

1. In Chapter 2 we have built ConTrib, a universal accounting mechanism. With a two-year deployment trial of ConTrib in our peer-to-peer application Tribler, we have successfully addressed free-riding behaviour in our Tor-like overlay. Our ConTrib mechanism is highly suitable for accounting data within different application domains that can tolerate fraud to remain undetected for a short period. In this thesis we have leveraged the accounting capabilities of ConTrib to store data elements in the XChange (Chapter 4), Internet-of-Money (Chapter 5) and dAppCoder/DevID (Chapter 6) mechanisms.

2. In Chapter 3 we have presented MATCH, decentralized middleware that is highly resilient against manipulation during order matchmaking. This manipulation is a significant concern in peer-to-peer markets under central ownership. MATCH performs high-quality matchmaking and does so with bandwidth, latency, and memory overhead orders of magnitude lower compared to matchmaking on a blockchain. We are the first to experiment with a fair and decentralized alternative to the Uber ride-hailing market.

3. In Chapter 4 we have presented a novel approach for asset trading between permissioned blockchains. Compared to existing trading approaches that either require third party intervention or modifications to deployed blockchain logic, our approach is fully decentralized and is compatible with all permissioned blockchains. Peers record all trading activity in a distributed log, and users will not trade with suspected fraudsters until an identified dispute is resolved. This approach significantly reduces the economic damage that adversaries can cause in the system.

4. In Chapter 5 we have presented how we reduce the settlement duration of inter-bank payments from days to mere seconds. Our decentralized overlay, Internet-of-Money, circumvents slow settlement by a central bank by breaking up an inter-bank payment into multiple intra-bank payments and by routing funds through the bank account of intermediate money routers. By accounting all money transfers between users and money routers in a distributed log, we can detect if a money router seized incoming funds. Money routers that have committed such fraud are blacklisted by users. Our Internet-of-Money mechanism does not require changes to existing banking infrastructure.

5. In Chapter 6 we have introduced dAppCoder, a decentralized crowdsourcing marketplace for the development of dApps. A key component of this platform is DevID, unified, blockchain-based portfolios. DevID solves the problem that the portfolio of developers is usually scattered across centralized platforms, and vendor locked-in, making it hard to get an accurate impression of the developers' skills. Our fully decentralized software crowdsourcing marketplace leverages DevID portfolios to match clients with developers, reduces search frictions and avoids trusted intermediaries for information management and client-developer payouts.

The following three conclusions transcend single chapters:

6. *Pair-wise accounting* is an efficient and effective approach to devise market mechanisms without central authority or trusted intermediaries. We have used the ConTrib mechanism to implement this approach in the XChange (Chapter 4), Internet-of-Money (Chapter 5) and dAppCoder (Chapter 6) mechanisms to detect fraudulent behaviour and to store information generated by peers.

7. *Detecting* fraud, instead of preventing it, is an efficient and often overlooked approach that can improve the performance of blockchain-based marketplaces. In Chapter 2 we have demonstrated that fraud targeted at the ConTrib data structure can be detected within seconds. In Chapter 4 we detect fraud and violate the liveness of malicious peers, preventing them from causing further harm. Finally, in Chapter 5 we leverage fraud detection to identify malicious money routers and show that the economic gains by adversaries are manageable.

8. *Incremental settlement*, the act of breaking up an individual payment into multiple smaller ones, is an effective risk mitigation strategy. We have successfully applied this strategy to reduce value-at-stake in our XChange trading mechanism (Chapter 4) and our Internet-of-Money overlay (Chapter 5).

7.2 Future Directions

Many opportunities remain to decentralize and disintermediate the aspects of blockchain-based marketplaces. We end this thesis by outlining, per chapter, directions for further research.

1. In Chapter 2 we have introduced the universal accounting mechanism ConTrib. As we also point out in Chapter 2, ConTrib would benefit from privacy-preserving enhancements that reduce the amount of sensitive information one can extract with transaction analysis. Other efforts could focus on improving the probability of fraud detection further when sharing records. We believe that more sophisticated dissemination techniques can further improve our mechanism. For example, the dissemination of records can take the record payload into consideration. Applications then share "important" records amongst more peers, increasing their availability.

2. In Chapter 3 we have presented MATCH, our decentralized middleware for fair matchmaking in peer-to-peer markets. Even though we show that MATCH is highly resistant against malicious matchmakers, we considered the identification of such matchmakers outside the scope of our work. A natural extension of MATCH would be to leverage ConTrib and record full specifications of order dissemination and proposed matches. By replaying the matching events in ones personal ledger, a user can detect a deviation from a particular matching policy. This would, however, incur additional resource usage. We also suggest exploring statistical approaches for the detection of malicious behaviour. Specifically, our random dissemination model results in a particular distribution of the order book entries over peers in the network. By inspecting incoming match proposals, long-term deviation from a matching policy can be detected.

3. In Chapter 4 we have introduced a universal asset trading mechanism between permissioned blockchains. An important question that remains is how our mechanism can be used to trade assets between public blockchains, for example, between the Ethereum and Bitcoin blockchains. The critical problem when deploying our mechanism in a public setting is the ability to quickly generate a new identity after committing fraud, i.e., the Sybil Attack. We envision that the use of collateral deposits can help to alleviate this threat.

4. In Chapter 5 we have presented our international money transfer mechanism, named Internet-of-Money. A shortcoming of our approach is that the magnitude of an intra-bank payment is limited by the balance constraints of money routers in the circuit. Once a money router has depleted its balance in one connected bank account, this router might be unable to route further payments. Rebalancing the router requires a conventional payment which can be slow. A potential research avenue is to rebalance money routers using Internet-of-Money functionality itself.

5. In Chapter 6 we have introduced a decentralized crowdsourcing marketplace which includes unified portfolios for software developers. We believe that there are opportunities to research new processes for securely linking third-party assets with DevID portfolios, for example, using verifiable claims in conjunction with self-sovereign

identities. Other research efforts can focus on a large-scale deployment of dApp-Coder and DevID, and the integration of these tools in platforms like GitHub.

Bibliography

References

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

[2] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[3] Paul Syverson, Roger Dingledine, and Nick Mathewson. Tor: The second generation onion router. In *Usenix Security*, pages 303–320, 2004.

[4] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, 2017(4):404–426, 2017.

[5] Tomaso Aste, Paolo Tasca, and Tiziana Di Matteo. Blockchain technologies: The foreseeable impact on society and industry. *computer*, 50(9):18–28, 2017.

[6] Kai Sedgwick. No, visa doesn't handle 24,000 tps and neither does your pet blockchain. `https://news.bitcoin.com/no-visa-doesnt-handle-24000-tps-and-neither-does-your-pet-blockchain/`, 2018.

[7] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with bitcoins. In *IEEE P2P 2013 Proceedings*, pages 1–5. IEEE, 2013.

[8] Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer. The carbon footprint of bitcoin. *Joule*, 3(7):1647–1661, 2019.

[9] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.

[10] Ghassan Karame. On the security and scalability of bitcoin's blockchain. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1861–1862, 2016.

[11] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19:1, 2012.

[12] Daniel Larimer. Delegated proof-of-stake. *Bitshares whitepaper*, 2014.

[13] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 18(2), 1996.

[14] Friedhelm Victor and Bianca Katharina Lüders. Measuring ethereum-based erc20 token networks. In *International Conference on Financial Cryptography and Data Security*, pages 113–129. Springer, 2019.

[15] Muhammad Izhar Mehar, Charles Louis Shier, Alana Giambattista, Elgar Gong, Gabrielle Fletcher, Ryan Sanayhie, Henry M Kim, and Marek Laskowski. Understanding a revolutionary and flawed grand experiment in blockchain: the dao attack. *Journal of Cases on Information Technology (JCIT)*, 21(1):19–32, 2019.

[16] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th USENIX Security Symposium (USENIX Security)*, pages 1335–1352, 2018.

[17] Lexi Brent, Neville Grech, Sifis Lagouvardos, Bernhard Scholz, and Yannis Smaragdakis. Ethainter: a smart contract security analyzer for composite vulnerabilities. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 454–469, 2020.

[18] Lawrence J Trautman. Virtual currencies; bitcoin & what now after liberty reserve, silk road, and mt. gox? *Richmond Journal of Law and Technology*, 20(4), 2014.

[19] Lindsay X Lin, Eric Budish, Lin William Cong, Zhiguo He, Jonatan H Bergquist, Mohit Singh Panesir, Jack Kelly, Michelle Lauer, Ryan Prinster, Stephenie Zhang, et al. Deconstructing decentralized exchanges. *Stanford Journal of Blockchain Law & Policy*, 2, 2019.

[20] Yan Chen and Cristiano Bellavitis. Blockchain disruption and decentralized finance: The rise of decentralized business models. *Journal of Business Venturing Insights*, 13, 2020.

[21] Lewis Gudgeon, Daniel Perez, Dominik Harz, Benjamin Livshits, and Arthur Gervais. The decentralized financial crisis. In *2020 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pages 1–15. IEEE, 2020.

[22] Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. *arXiv preprint arXiv:2003.03810*, 2020.

[23] Yannis Bakos. The emerging role of electronic marketplaces on the internet. *Communications of the ACM*, 41(8):35–42, 1998.

[24] Theodore H Clark and Ho Geun Lee. Electronic intermediaries: Trust building and market differentiation. In *Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers*, pages 10–pp. IEEE, 1999.

[25] PayPal. `https://www.paypal.com`, 2007.

[26] Ye Guo and Chen Liang. Blockchain application and outlook in the banking industry. *Financial Innovation*, 2(1):24, 2016.

[27] Rolf T Wigand. Whatever happened to disintermediation? *Electronic Markets*, pages 1–9, 2020.

[28] George M Giaglis, Stefan Klein, and Robert M O'Keefe. Disintermediation, reintermediation, or cybermediation? the future of intermediaries in electronic marketplaces. In *Global Networked Organizations, Proceedings of the 12th Electronic Commerce Conference*. Citeseer, 1999.

[29] Stefano Lande and Roberto Zunino. Sok: unraveling bitcoin smart contracts. *Principles of Security and Trust LNCS 10804*, page 217, 2018.

[30] Efpraxia D Zamani and George M Giaglis. With a little help from the miners: distributed ledger technology and market disintermediation. *Industrial Management & Data Systems*, 2018.

[31] Frederik Armknecht, Ghassan O Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. Ripple: Overview and outlook. In *International Conference on Trust and Trustworthy Computing*, pages 163–180. Springer, 2015.

[32] Richard Gendal Brown. The corda platform: An introduction. Technical report, R3, 2018.

[33] Openbazaar. `https://www.openbazaar.org`, 2018.

[34] Benny Rachlevsky-Reich, Israel Ben-Shaul, Nicholas Tung Chan, Andrew W Lo, and Tomaso Poggio. Gem: A global electronic market system. *Information Systems*, 24(6):495–518, 1999.

[35] Aurora Labs. Aurora: A decentralized financial institution utilizing distributed computing and the ethereum network. Technical report, Aurora Labs, 2018.

[36] Etherdelta. `https://etherdelta.com`, 2018.

[37] 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.

[38] David Hausheer. *PeerMart: Secure decentralized pricing and accounting for peer-to-peer systems*. ETH Zurich, 2006.

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

[40] Juan Benet. libp2p specification. `https://github.com/libp2p/specs`, 2018.

[41] Juan Benet and Nicola Greco. Filecoin: A decentralized storage network. Technical report, Protocol Labs, 2018.

[42] Daniel J Veit. *Matchmaking in electronic markets: An agent-based approach towards matchmaking in electronic negotiations*, volume 2882 of *Lecture Notes in Artificial Intelligence*. Springer, 2003.

[43] Yue Wu, Kaifu Zhang, and V Padmanabhan. The matchmaker's dilemma. Technical report, INSEAD, 2015.

[44] Vasilios Mavroudis and Hayden Melton. Libra: Fair order-matching for electronic financial exchanges. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies (AFT)*, pages 156–168, 2019.

[45] K Sigdel, S Li, B Pourebrahimi, K Bertels, and S Vassiliadis. Centralized matchmaking-an empirical study. In *16th Annual Workshop on Circuits, Systems and Signal Processing*, pages 438–444. Published, 2005.

[46] Will Warren and Amir Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. Technical report, 0x Project, 2017.

[47] Michael Oved and Don Mosites. Swap: A peer-to-peer for trading ethereum tokens, 2017.

[48] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. Fast and secure global payments with stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pages 80–96, 2019.

[49] Fabian Schuh and Daniel Larimer. Bitshares 2.0: Financial smart contract platform, 2015.

[50] Daniel Wang, Jay Zhou, Alex Wang, and Matthew Finestone. Loopring: A decentralized token exchange protocol. Technical report, Loopring, 2018.

[51] Taiyang Zhang and Loong Wang. Republic protocol, 2017.

[52] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[53] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, pages 245–254. ACM, 2018.

[54] TierNolan. Atomic swaps using cut and choose. `https://bitcointalk.org/index.php?topic=1364951.0`, 2016.

[55] Henning Pagnia, Holger Vogt, and Felix C Gärtner. Fair exchange. *The Computer Journal*, 46(1):55–75, 2003.

[56] Stefan Dziembowski, Lisa Eckey, and Sebastian Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984, 2018.

[57] Nadarajah Asokan, Matthias Schunter, and Michael Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM conference on Computer and communications security*, pages 7–17, 1997.

[58] Jian Liu, Wenting Li, Ghassan O Karame, and N Asokan. Toward fairness of cryptocurrency payments. *IEEE Security & Privacy*, 16(3):81–89, 2018.

[59] Stefan Thomas and Evan Schwartz. A protocol for interledger payments. *URL https://interledger. org/interledger. pdf*, 2015.

[60] Bisq. Bisq - the p2p exchange network. `https://bisq.network`, 2018.

[61] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 193–210. IEEE, 2019.

[62] Hemang Subramanian. Decentralized blockchain-based electronic marketplaces. *Communications of the ACM*, 61(1):78–84, 2017.

[63] Tribler team. Ipv8 networking library. `https://github.com/tribler/py-ipv8`, 2021.

[64] Henri Bal, Dick Epema, Cees de Laat, Rob van Nieuwpoort, John Romein, Frank Seinstra, Cees Snoek, and Harry Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.

[65] Tribler team. Gumby experiment framework. `https://github.com/tribler/gumby`, 2021.

[66] Niels Zeilemaker, Mihai Capotă, Arno Bakker, and Johan Pouwelse. Tribler: P2p media search and sharing. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 739–742, 2011.

[67] Jon Frost, Leonardo Gambacorta, Yi Huang, Hyun Song Shin, and Pablo Zbinden. Bigtech and the changing structure of financial intermediation. *Economic Policy*, 2019.

[68] Michael L Best. The internet that facebook built. *Communications of the ACM*, 57(12):21–23, 2014.

[69] Juliet Schor. Debating the sharing economy. *Journal of Self-Governance and Management Economics*, 4(3):7–22, 2016.

[70] Koen Frenken and Juliet Schor. Putting the sharing economy into perspective. In *A Research Agenda for Sustainable Consumption Governance*. Edward Elgar Publishing, 2019.

[71] Eszter Bokányi and Anikó Hannák. Understanding Inequalities in Ride-Hailing Services Through Simulations. *Scientific Reports*, 10(1):6500, December 2020.

[72] Bapu Kotapati, Simon Mutungi, Melissa Newham, Jeff Schroeder, Shili Shao, and Melody Wang. The antitrust case against apple. *Available at SSRN*, 2020.

[73] Imran Bashir. *Mastering Blockchain: Distributed ledger technology, decentralization, and smart contracts explained.* Packt Publishing Ltd, 2018.

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

[75] Thomas Locher, Patrick Moore, Stefan Schmid, and Roger Wattenhofer. Free riding in bittorrent is cheap. In *5th Workshop on Hot Topics in Networks (HotNets)*, 2006.

[76] Michael Sirivianos, Jong Han Park, Rex Chen, and Xiaowei Yang. Free-riding in bittorrent networks with the large view exploit. In *IPTPS*, 2007.

[77] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Detection and removal of malicious peers in gossip-based protocols. Technical report, University of Bologna, 2004.

[78] 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.

[79] Martijn de Vos and Johan Pouwelse. A blockchain-based micro-economy of bandwidth tokens. *CompSys 2018*, 2018.

[80] Richard TB Ma, Sam CM Lee, John CS Lui, and David KY Yau. An incentive mechanism for p2p networks. In *24th International Conference on Distributed Computing Systems*, pages 516–523. IEEE, 2004.

[81] Giancarlo Ruffo and Rossano Schifanella. Fairpeers: Efficient profit sharing in fair peer-to-peer market places. *Journal of Network and Systems Management*, 15(3):355–382, 2007.

[82] David P Anderson. Boinc: A platform for volunteer computing. *Journal of Grid Computing*, pages 1–24, 2019.

[83] David Easley, Maureen O'Hara, and Soumya Basu. From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial Economics*, 134(1):91–109, 2019.

[84] Mainak Ghosh, Miles Richardson, Bryan Ford, and Rob Jansen. A torpath to torcoin: Proof-of-bandwidth altcoins for compensating relays. Technical report, Naval Research, 2014.

[85] Thomas Hummel, O Stramme, and Ryan M La Salle. Earning a living among peers-the quest for viable p2p revenue models. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 10–pp. IEEE, 2003.

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

[87] Murat Karakaya, Ibrahim Korpeoglu, and Özgür Ulusoy. Free riding in peer-to-peer networks. *IEEE Internet computing*, 13(2):92–98, 2009.

[88] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. Reputation systems: A survey and taxonomy. *Journal of Parallel and Distributed Computing*, 75:184–197, 2015.

[89] Emanuele Bellini, Youssef Iraqi, and Ernesto Damiani. Blockchain-based distributed trust and reputation management systems: A survey. *IEEE Access*, 8:21127–21151, 2020.

[90] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72. Berkeley, CA, USA, 2003.

[91] Tsuen-Wan Ngan, Dan S Wallach, and Peter Druschel. Enforcing fair sharing of peer-to-peer resources. In *International Workshop on Peer-to-Peer Systems*, pages 149–159. Springer, 2003.

[92] Ivan Osipkov, Peng Wang, and Nicholas Hopper. Robust accounting in decentralized p2p storage systems. In *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pages 14–14. IEEE, 2006.

[93] Rachid Guerraoui, Kévin Huguenin, Anne-Marie Kermarrec, Maxime Monod, and Swagatika Prusty. Lifting: lightweight freerider-tracking in gossip. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 313–333. Springer, 2010.

[94] Sonia Ben Mokhtar, Jérémie Decouchant, and Vivien Quéma. Acting: Accurate freerider tracking in gossip. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 291–300. IEEE, 2014.

[95] Sven Seuken and David C Parkes. Sybil-proof accounting mechanisms with transitive trust. *Proceedings of the International Foundation for Autonomous Agents and Multiagent Systems*, 2014.

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

[97] Amadou Diarra, Sonia Ben Mokhtar, Pierre-Louis Aublin, and Vivien Quéma. Fullreview: Practical accountability in presence of selfish nodes. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 271–280. IEEE, 2014.

[98] Pim Otte, Martijn de Vos, and Johan Pouwelse. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 107:770–780, 2020.

[99] Scott A Crosby and Dan S Wallach. Efficient data structures for tamper-evident logging. In *USENIX Security*, pages 317–334, 2009.

[100] David Hausheer and Burkhard Stiller. Peermint: Decentralized and secure accounting for peer-to-peer applications. In *International Conference on Research in Networking*, pages 40–52. Springer, 2005.

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

[102] 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.

[103] Rahim Delaviz, Nazareno Andrade, Johan Pouwelse, and Dick Epema. Sybilres: A sybil-resilient flow-based decentralized reputation mechanism. In *2012 IEEE 32nd International Conference on Distributed Computing Systems*, pages 203–213. IEEE, 2012.

[104] Haifeng Yu, Phillip B 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.

[105] 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.

[106] 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.

[107] Yuhong Liu, Yafei Yang, and Yan Lindsay Sun. Detection of collusion behaviors in online reputation systems. In *2008 42nd Asilomar Conference on Signals, Systems and Computers*, pages 1368–1372. IEEE, 2008.

[108] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 315–334. IEEE, 2018.

[109] Andrew Poelstra, Adam Back, Mark Friedenbach, Gregory Maxwell, and Pieter Wuille. Confidential assets. In *International Conference on Financial Cryptography and Data Security*, pages 43–63. Springer, 2018.

[110] Ramayya Krishnan, Michael Smith, Zhulei Tang, and Rahul Telang. The virtual commons: Why free-riding can be tolerated in file sharing networks. *ICIS 2002 Proceedings*, page 82, 2002.

[111] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *iNetSec*, pages 112–125. Springer, 2015.

[112] Ivan Osipkov, Eugene Y Vasserman, Nicholas Hopper, and Yongdae Kim. Combating double-spending using cooperative p2p systems. In *27th International Conference on Distributed Computing Systems (ICDCS'07)*, pages 41–41. IEEE, 2007.

[113] Martijn de Vos, Can Umut Ileri, and Johan Pouwelse. Xchange: A universal mechanism for asset exchange between permissioned blockchains. *World Wide Web*, pages 1–38, 2021.

[114] Martijn de Vos, Mitchell Olsthoorn, and Johan Pouwelse. Devid: Blockchain-based portfolios for software developers. In *2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON)*, pages 158–163. IEEE, 2019.

[115] Martijn de Vos and Johan Pouwelse. Real-time money routing by trusting strangers with your funds. In *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 1–9. IEEE, 2018.

[116] Norm Matloff. Introduction to discrete-event simulation and the simpy language. *Davis, CA. Dept of Computer Science. University of California at Davis. Retrieved on August*, 2(2009):1–33, 2008.

[117] Paolo Palmieri and Johan Pouwelse. Paying the guard: an entry-guard-based payment system for tor. In *International Conference on Financial Cryptography and Data Security*, pages 437–444. Springer, 2015.

[118] Rob Jansen, Matthew Traudt, John Geddes, Chris Wacek, Micah Sherr, and Paul Syverson. Kist: Kernel-informed socket transport for tor. *ACM Transactions on Privacy and Security (TOPS)*, 22(1):1–37, 2018.

[119] Arvind Malhotra and Marshall Van Alstyne. The dark side of the sharing economy… and how to lighten it. *Communications of the ACM*, 57(11):24–27, 2014.

[120] PWC. Consumer intelligence series: The sharing economy. Technical report, PWC, 2015.

[121] OECD. *An Introduction to Online Platforms and Their Role in the Digital Transformation*. OECD, 2019.

[122] Pepper D. Culpepper and Kathleen Thelen. Are we all amazon primed? consumers and the politics of platform power. *Comparative Political Studies*, 53(2):288–318, 2020.

[123] Eduardo M. Azevedo and E. Glen Weyl. Matching markets in the digital age. *Science*, 352(6289):1056–1057, 2016.

[124] David Trastour, Claudio Bartolini, and Chris Preist. Semantic web support for the business-to-business e-commerce lifecycle. In *Proceedings of the 11th international conference on World Wide Web*, pages 89–98, 2002.

[125] Daniel J Veit, Christof Weinhardt, and Jorg P Muller. Multi-dimensional matchmaking for electronic markets. *Applied Artificial Intelligence*, 16(9-10):853–869, 2002.

[126] Aniko Hannak, Gary Soeller, David Lazer, Alan Mislove, and Christo Wilson. Measuring price discrimination and steering on e-commerce web sites. In *ICM*, page 305–318, 2014.

[127] Ryan Calo and Alex Rosenblat. The taking economy: Uber, information, and power. *Colum. L. Rev.*, 117:1623, 2017.

[128] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 254–269, 2016.

[129] 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.

[130] Aashish Kolluri, Ivica Nikolic, Ilya Sergey, Aquinas Hobor, and Prateek Saxena. Exploiting the laws of order in smart contracts. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 363–373, 2019.

[131] Aljosha Judmayer, Nicholas Stifter, Alexei Zamyatin, Itay Tsabary, Ittay Eyal, Peter Gazi, Sarah Meiklejohn, and Edgar R Weippl. Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies. *IACR*, 2019:775, 2019.

[132] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Aris Juels. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. *arXiv e-prints*, 2019.

[133] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[134] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, pages 407–418, 2003.

[135] Ernesto Damiani, S De Capitani Di Vimercati, and Pierangela Samarati. Managing multiple and dependable identities. *IEEE Internet Computing*, 7(6):29–37, 2003.

[136] 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.

[137] Quinten Stokkink, Dick Epema, and Johan Pouwelse. A truly self-sovereign identity system. *arXiv preprint arXiv:2007.00415*, 2020.

[138] Zvi Schreiber. k-root-n: An efficient algorithm for avoiding short term double-spending alongside distributed ledger technologies such as blockchain. *Information*, 11(2):90, 2020.

[139] Anh Pham, Italo Dacosta, Guillaume Endignoux, Juan Ramon Troncoso Pastoriza, Kévin Huguenin, and Jean-Pierre Hubaux. Oride: A privacy-preserving yet accountable ride-hailing service. In *USENIX Security*, pages 1235–1252, 2017.

[140] Rui Zhu, Bang Liu, Di Niu, Zongpeng Li, and Hong Vicky Zhao. Network latency estimation for personal devices: A matrix completion approach. *Transactions on Networking*, 25(2):724–737, 2016.

[141] NYC Taxi and Limousine Commission. Trip record data. `https://www1.nyc.gov/site/tlc/about/tlc-trip-record-data.page`, 2017.

[142] Iddo Bentov, Yan Ji, Fan Zhang, Lorenz Breidenbach, Philip Daian, and Ari Juels. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1521–1538, 2019.

[143] Fabian Schuh and Daniel Larimer. Bitshares 2.0: general overview, 2017.

[144] Antonio Carzaniga, David S Rosenblum, and Alexander L Wolf. Design and evaluation of a wide-area event notification service. *Transactions on Computer Systems (TOCS)*, 19(3):332–383, 2001.

[145] Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E Strom, and Daniel C Sturman. An efficient multicast protocol for content-based publish-subscribe systems. In *Proceedings. 19th International Conference on Distributed Computing Systems*, pages 262–272. IEEE, 1999.

[146] Antonio Carzaniga, Matthew J Rutherford, and Alexander L Wolf. A routing scheme for content-based networking. In *INFOCOM*, volume 2, pages 918–928. IEEE, 2004.

[147] Karl Czajkowski, Ian Foster, Nick Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A resource management architecture for metacomputing systems. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer, 1998.

[148] Simone A Ludwig and Thomas Schoene. Matchmaking in multi-attribute auctions using a genetic algorithm and a particle swarm approach. In *New Trends in Agent-Based Complex Automated Negotiations*, pages 81–98. Springer, 2012.

[149] Jacek Gomoluch and Michael Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Middleware*, pages 211–218, 2003.

[150] Marian Mihailescu and Yong Meng Teo. A distributed market framework for large-scale resource sharing. In *European Conference on Parallel Processing*, pages 418–430. Springer, 2010.

[151] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14:1507–1542, 2002.

[152] Rajesh Raman, Miron Livny, and Marvin Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceedings of The 7th International Symposium on High Performance Distributed Computing*, pages 140–146. IEEE, 1998.

[153] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[154] B Pour Ebrahimi, K Bertels, S Vassiliadis, and K Sigdel. Matchmaking within multiagent systems. *Proceeding of ProRisc-2004*, 2004.

[155] Abdulrahman A Azab and Hisham A Kholidy. An adaptive decentralized scheduling mechanism for peer-to-peer desktop grids. In *International Conference on Computer Engineering & Systems*, pages 364–371. IEEE, 2008.

[156] Janick Edinger, Dominik Schäfer, and Christian Becker. Decentralized scheduling for tasklets. In *Proceedings of the Posters and Demos Session of the 17th International Middleware Conference*, pages 7–8, 2016.

[157] Tariq Abdullah, Koen Bertels, Luc Onana Alima, and Zubair Nawaz. Effect of the degree of neighborhood on resource discovery in ad hoc grids. In *International Conference on Architecture of Computing Systems*, pages 174–186. Springer, 2010.

[158] K Sigdel, K Bertels, B Pourebrahimi, S Vassiliadis, and L Shuai. A framework for adaptive matchmaking in distributed computing. In *Proceedings of GRID Workshop*, volume 224, 2005.

[159] Victor Shafran, Gal Kaminka, Sarit Kraus, and Claudia V Goldman. Towards bidirectional distributed matchmaking. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 3*, pages 1437–1440. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[160] Oved Michael and Mosites Don. Swap: A peer-to-peer protocol for trading ethereum tokens. Technical report, AirSwap, 2017.

[161] D Haussheer and Burkhard Stiller. Decentralized auction-based pricing with peermart. In *2005 9th IFIP/IEEE International Symposium on Integrated Network Management, 2005. IM 2005.*, pages 381–394. IEEE, 2005.

[162] Hisham S Galal and Amr M Youssef. Verifiable sealed-bid auction on the ethereum blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 265–278. Springer, 2018.

[163] Hisham S Galal and Amr M Youssef. Succinctly verifiable sealed-bid auction smart contract. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 3–19. Springer, 2018.

[164] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, and Yacov Manevich. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference*, pages 1–15, 2018.

[165] Stefano De Angelis, Leonardo Aniello, Roberto Baldoni, Federico Lombardi, Andrea Margheri, and Vladimiro Sassone. Pbft vs proof-of-authority: applying the cap theorem to permissioned blockchain. In *Italian Conference on Cyber Security*, January 2018.

[166] Marko Vukolić. Rethinking permissioned blockchains. In *Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts*, pages 3–7, 2017.

[167] J.P. Morgan. Quorum. `https://github.com/ConsenSys/quorum`, 2018.

[168] Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, and Alberto Granzotto. Bigchaindb: a scalable blockchain database. *white paper, BigChainDB*, 2016.

[169] Nadia Hewett, Wolfgang Lehmacher, and Yingli Wang. Inclusive deployment of blockchain for supply chains. Technical report, World Economic Forum, 2019.

[170] Michael Borkowski, Daniel McDonald, Christoph Ritzer, and Stefan Schulte. Towards atomic cross-chain token transfers: State of the art and open questions within tast. *Distributed Systems Group TU Wien (Technische Universit at Wien), Report*, 2018.

[171] Michael del Castillo. Blockchain 50: Billion dollar babies. `https://www.forbes.com/sites/michaeldelcastillo/2019/04/16/blockchain-50-billion-dollar-babies`, 2019.

[172] Stefan Schulte, Marten Sigwart, Philipp Frauenthaler, and Michael Borkowski. Towards blockchain interoperability. In *International Conference on Business Process Management*, pages 3–10. Springer, 2019.

[173] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. Where is current research on blockchain technology?—a systematic review. *PloS one*, 11(10):e0163477, 2016.

[174] Pekka Nikander, Juuso Autiosalo, and Santeri Paavolainen. Interledger for the industrial internet of things. In *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*, volume 1, pages 908–915. IEEE, 2019.

[175] Hoang Tam Vo, Ziyuan Wang, Dileban Karunamoorthy, John Wagner, Ermyas Abebe, and Mukesh Mohania. Internet of blockchains: techniques and challenges ahead. 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 1574–1581. IEEE, 2018.

[176] Emmanuelle Ganne. *Can Blockchain revolutionize international trade?* World Trade Organization Geneva, 2018.

[177] Tommy Koens and Erik Poll. Assessing interoperability solutions for distributed ledgers. *Pervasive and Mobile Computing*, 59:101079, 2019.

[178] Gareth W Peters and Efstathios Panayi. Understanding modern banking ledgers through blockchain technologies: Future of transaction processing and smart contracts on the internet of money. In *Banking beyond banks and money*, pages 239–278. Springer, 2016.

[179] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on selected areas in communications*, 24(5):1010–1019, 2006.

[180] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. Technical report, IACR Cryptology ePrint Archive, 2019: 1128, 2019.

[181] Vitalik Buterin. Chain interoperability. *R3 Research Paper*, 2016.

[182] Rami Khalil, Arthur Gervais, and Guillaume Felley. Tex-a securely scalable trustless exchange. *IACR Cryptol. ePrint Arch.*, 2019:265, 2019.

[183] Ethan Heilman, Sebastien Lipmann, and Sharon Goldberg. The arwen trading protocols. In *International Conference on Financial Cryptography and Data Security*, pages 156–173. Springer, 2020.

[184] Runchao Han, Haoyu Lin, and Jiangshan Yu. On the optionality and fairness of atomic swaps. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 62–75. ACM, 2019.

[185] Dawei Li, Jianwei Liu, Zongxun Tang, Qianhong Wu, and Zhenyu Guan. Agentchain: A decentralized cross-chain exchange system. In *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pages 491–498. IEEE, 2019.

[186] Arlyn Culwick and Dan Metcalf. The blocknet design specification, 2019.

[187] ARK Team. Ark ecosystem whitepaper, 2019.

[188] POA Team. Poa network whitepaper, 2018.

[189] Amritraj Singh, Kelly Click, Reza M Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications*, 149:102471, 2020.

[190] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 72, 2014.

[191] Johnny Dilley, Andrew Poelstra, Jonathan Wilkins, Marta Piekarska, Ben Gorlick, and Mark Friedenbach. Strong federations: An interoperable blockchain solution to centralized third-party risks. *arXiv preprint arXiv:1612.05491*, 2016.

[192] jae Kwon and Ethan Buchman. Cosmos - a network of distributed ledgers. Technical report, Tendermint, 2018.

[193] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.

[194] Jae Kwon. Tendermint: Consensus without mining. Technical report, Tendermint, 2014.

[195] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42, 2016.

[196] Interledger Project. Interledger protocol v4. `https://interledger.org/rfcs/0027-interledger-protocol-4/`.

[197] Hyperledger. Hyperledger quilt. `https://github.com/hyperledger/quilt`, 2020.

[198] Lisa Eckey, Sebastian Faust, and Benjamin Schlosser. Optiswap: Fast optimistic fair exchange. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security*, pages 543–557, 2020.

[199] Sergi Delgado-Segura, Cristina Pérez-Solà, Guillermo Navarro-Arribas, and Jordi Herrera-Joancomartí. A fair protocol for data trading based on bitcoin transactions. *Future Generation Computer Systems*, 107:832–840, 2020.

[200] Indrajit Ray and Indrakshi Ray. Fair exchange in e-commerce. *ACM SIGecom Exchanges*, 3(2):9–17, 2002.

[201] Rajashekar Kailar. Accountability in electronic commerce protocols. *IEEE Transactions on software engineering*, 22(5):313–328, 1996.

[202] Stephen M. Specht and Ruby B. Lee. Distributed denial of service: Taxonomies of attacks, tools, and countermeasures. In *Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, September 15-17, 2004, The Canterbury Hotel, San Francisco, California, USA*, pages 543–550, 2004.

[203] Martijn de Vos, Georgy Ishmaev, and Johan Pouwelse. Match: A decentralized middleware for fair matchmaking in peer-to-peer markets. In *Proceedings of the 21th International Middleware Conference*, 2020.

[204] Alasdair Gilchrist. *Industry 4.0: the industrial internet of things*. Springer, 2016.

[205] Paul Resnick and Richard Zeckhauser. Trust among strangers in internet transactions: Empirical analysis of ebay's reputation system. *The Economics of the Internet and E-commerce*, 11(2):23–25, 2002.

[206] Tom Kokkola. *The payment system: Payments, securities and derivatives, and the role of the Eurosystem*. European Central Bank, 2011.

[207] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *IEEE Access*, 8:16440–16455, 2020.

[208] Robert McMillan. The inside story of mt. gox, bitcoins 460 million dollar disaster. *Wired. March*, 3, 2014.

[209] Julia Carrie Wong. Uber concealed massive hack that exposed data of 57m users and drivers. `https://www.theguardian.com/technology/2017/nov/21/uber-data-hack-cyber-attack`, 2017.

[210] Zheng Yan and Silke Holtmanns. Trust modeling and management: from social trust to digital trust. *IGI Global*, pages 290–323, 2008.

[211] David M Kreps, Paul Milgrom, John Roberts, and Robert Wilson. Rational cooperation in the finitely repeated prisoners' dilemma. *Journal of Economic theory*, 27(2):245–252, 1982.

[212] Bert-Jaap Koops. The trouble with european data protection law. *International Data Privacy Law*, 4(4):250–261, 2014.

[213] 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. ACM, 2003.

[214] Mudhakar Srivatsa, Li Xiong, and Ling Liu. Trustguard: countering vulnerabilities in reputation management for decentralized overlay networks. In *Proceedings of the 14th international conference on World Wide Web*, pages 422–431. ACM, 2005.

[215] Swift payment system. `https://www.swift.com`.

[216] McKinsey. *Global Payments 2016*, 2016 (accessed November 27, 2017).

[217] Johan Pouwelse, André de Kok, Joost Fleuren, Peter Hoogendoorn, Raynor Vliegendhart, and Martijn de Vos. Laws for creating trust in the blockchain age. *European Property Law Journal*, 6(3):321–356, 2017.

[218] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[219] Ansley Post, Vijit Shah, and Alan Mislove. Bazaar: Strengthening user reputations in online marketplaces. In *Proceedings of NSDI*, volume 11, page 183, 2011.

[220] Mounaim Cortet, Tom Rijks, and Shikko Nijland. Psd2: The digital transformation accelerator for banks. *Journal of Payments Strategy & Systems*, 10(1):13–27, 2016.

[221] John Doe and Johan Pouwelse. A vulnerability analyis of smartphone banking applications. Unpublished research, 2015.

[222] J.P. Awesome and Johan Pouwelse. A vulnerability analysis of mobile banking applications. Unpublished research, 2015.

[223] NGDATA. *NGDATA 2014 Consumer Banking Survey*, 2014 (accessed November 20, 2017).

[224] Rami Khalil and Arthur Gervais. Revive: Rebalancing off-blockchain payment networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 439–453, 2017.

[225] Niels Zeilemaker, Boudewijn Schoon, and Johan Pouwelse. Dispersy bundle synchronization. Technical report, Delft University of Technology, 2013.

[226] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.

[227] Patrick M Jost and Harjit Singh Sandhu. *The hawala alternative remittance system and its role in money laundering*. Interpol, 2003.

[228] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Lightning Network, 2016.

[229] David Schwartz, Noah Youngs, and Arthur Britto. The ripple protocol consensus algorithm. Technical report, Ripple Labs, 2014.

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

[231] Dirk A Zetzsche, Douglas W Arner, and Ross P Buckley. Decentralized finance. *Journal of Financial Regulation*, 6(2):172–203, 2020.

[232] Nasdaq. The blockchain developer shortage: Emerging trends and perspectives. https://www.nasdaq.com/article/the-blockchain-developer-shortage-emerging-trends-and-perspectives-cm701294, 2016.

[233] Yiwei Gong, Sélinde van Engelenburg, and Marijn Janssen. A reference architecture for blockchain-based crowdsourcing platforms. *Journal of Theoretical and Applied Electronic Commerce Research*, 16(4):937–958, 2021.

[234] Thomas D LaToza and Andre van der Hoek. Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE software*, 33(1):74–80, 2016.

[235] Maik Hesse and Timm Teubner. Reputation portability–quo vadis? *Electronic Markets*, pages 1–19, 2019.

[236] Nick Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[237] Jacob Eberhardt and Stefan Tai. On or off the blockchain? insights on off-chaining computation and data. In *European Conference on Service-Oriented and Cloud Computing*, pages 3–15. Springer, 2017.

[238] Unknown Authors. Tlsnotary - a mechanism for independently audited https sessions. Technical report, 2014.

[239] Sergei Popov. The tangle. Technical report, IOTA foundation, 2018.

[240] Sergio Demian Lerner. Dagcoin: a cryptocurrency without blocks, 2015.

[241] Colin LeMahieu. Nano: A feeless distributed cryptocurrency network. Technical report, Nano Foundation, 2017.

[242] Bram Cohen. The bittorrent protocol specification, 2008.

[243] Vikram Dhillon, David Metcalf, and Max Hooper. The dao hacked. In *Blockchain Enabled Applications*, pages 67–78. Springer, 2017.

[244] Dutchie Riggsby, Virginia Jewell, and Arthur Justice. Electronic portfolio: Assessment, resume, or marketing tool? Technical report, Columbus College, 1995.

[245] Helen Barrett. Electronic teaching portfolios. In *Society for Information Technology & Teacher Education International Conference*, pages 1029–1034. Association for the Advancement of Computing in Education (AACE), 1999.

[246] Dirk Riehle. How open source is changing the software developer's career. *IEEE Computer*, 48(5):51–57, 2015.

[247] Yuanfeng Cai and Dan Zhu. Reputation in an open source software community: Antecedents and impacts. *Decision Support Systems*, 91:103–112, 2016.

[248] Thunyathon Jaruchotrattanasakul, Xin Yang, Erina Makihara, Kenji Fujiwara, and Hajimu Iida. Open source resume (osr): A visualization tool for presenting oss biographies of developers. In *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 57–62. IEEE, 2016.

[249] Rohit Saxena and Niranjan Pedanekar. I know what you coded last summer: Mining candidate expertise from github repositories. In *CSCW*, pages 299–302. ACM, 2017.

[250] Xiaofan Chen and Anita Sarma. Supporting comparison of developer profiles across online communities. Technical report, Oregon State University, 2016.

[251] Karim R Lakhani, David A Garvin, and Eric Lonstein. Topcoder (a): Developing software through crowdsourcing. *Harvard Business School General Management Unit Case*, 610-032, 2010.

[252] Ming Li, Jian Weng, Anjia Yang, Wei Lu, Yue Zhang, Lin Hou, Jia-Nan Liu, Yang Xiang, and Robert H Deng. Crowdbc: A blockchain-based decentralized framework for crowdsourcing. *IEEE Transactions on Parallel and Distributed Systems*, 30(6):1251–1266, 2018.

[253] Yuan Lu, Qiang Tang, and Guiling Wang. Zebralancer: Private and anonymous crowdsourcing system atop open blockchain. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 853–865. IEEE, 2018.

[254] Jun Zou, Bin Ye, Lie Qu, Yan Wang, Mehmet A Orgun, and Lei Li. A proof-of-trust consensus protocol for enhancing accountability in crowdsourcing services. *IEEE Transactions on Services Computing*, 12(3):429–445, 2018.

[255] Francesco Buccafurri, Gianluca Lax, Serena Nicolazzo, and Antonino Nocera. Tweetchain: An alternative to blockchain for crowd-based applications. In *International Conference on Web Engineering*, pages 386–393. Springer, 2017.

Curriculum Vitæ

Marinus Abraham de Vos

09-03-1993 Date of birth in Sint-Maartensdijk, The Netherlands

Education

2011-2014 BSc Computer Science
Delft University of Technology, Delft, The Netherlands

2013-2014 Minor Education
Delft University of Technology, Delft, The Netherlands

2014-2016 MSc Computer Science
Thesis title: *Identifying and Managing Technical Debt in Complex Distributed Systems*
Distributed Systems
Delft University of Technology, Delft, The Netherlands

2016-2021 PhD Candidate
Dissertation title: *Decentralization and Disintermediation in Blockchain-based Marketplaces*
Distributed Systems
Delft University of Technology, Delft, The Netherlands

Work Experience

2013-2014 Internship Minor Education
Mollerlyceum, Bergen op Zoom, The Netherlands

2014-present Business Owner
CodeUp
Delft, The Netherlands

2016-2021 PhD Candidate
Distributed Systems
Delft University of Technology, Delft, The Netherlands

2021-present	Postdoctoral Researcher Distributed Systems Delft University of Technology, Delft, The Netherlands

List of Publications

1. **Martijn de Vos**, Georgy Ishmaev, and Johan Pouwelse, Decentralizing Components of Electronic Markets to Prevent Gatekeeping and Manipulation
   Under review.

2. Joost Verbraeken, **Martijn de Vos**, and Johan Pouwelse, Bristle: Decentralized Federated Learning in Byzantine, Non-i.i.d. Environments
   Under review.

3. **Martijn de Vos** and Johan Pouwelse, ConTrib: Maintaining Fairness in Decentralized Big Tech Alternatives by Accounting Work
   In *Computer Networks*, 2021, Elsevier.

4. **Martijn de Vos** and Johan Pouwelse, ConTrib: Universal and Decentralized Accounting in Shared-Resource Systems
   In *Proceedings of the 1st International Workshop on Distributed Infrastructure for Common Good (DICG'20)*, Delft, The Netherlands, 2020.

5. Ayman Esmat, **Martijn de Vos**, Yashar Ghiassi-Farrokhfal, Peter Palensky, and Dick Epema, A Novel Decentralized Platform for Peer-to-Peer Energy Trading Market with Blockchain Technology
   In *Applied Energy*, 2020, Elsevier.

6. **Martijn de Vos**, Georgy Ishmaev, and Johan Pouwelse, MATCH: A Decentralized Middleware for Fair Matchmaking In Peer-to-Peer Markets
   In *Proceedings of the 20th International Middleware Conference (Middleware'20)*, Delft, The Netherlands, 2020. Acceptance Rate 25% (30/119).

7. **Martijn de Vos**, Can Umut Ileri, and Johan Pouwelse, XChange: A Universal Mechanism for Asset Exchange between Permissioned Blockchains
   World Wide Web journal, Special Issue on Emerging Blockchain Applications and Technology, 2021, Springer.

8. **Martijn de Vos** and Johan Pouwelse, XChange: A Decentralized, Blockchain-based Mechanism for Generic Trade at Scale
   Presented and discussed at the *ninth Erasmus Liquidity Conference*, Rotterdam, The Netherlands, 2019 (no proceedings). Acceptance Rate **9.4%** (11/117).

9. **Martijn de Vos**, Mitchell Olsthoorn, and Johan Pouwelse, DevID: Blockchain-based Portfolios for Software Developers
   In *Proceedings of the 1st IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON'19)*, San Fransisco, United States of America, 2019.

10. **Martijn de Vos** and Johan Pouwelse, Real-time Money Routing by Trusting Strangers with your Funds
In *Proceedings of the 17th International IFIP TC6 Networking Conference*, Zurich, Switzerland, 2018. Acceptance Rate 24% (55/225).

11. Pim Otte, **Martijn de Vos** and Johan Pouwelse, TrustChain: A Sybil-resistant Scalable Blockchain
In *Future Generation Computer Systems, Special Issue on Cryptocurrency and Blockchain Technology*, 2017, Elsevier.

12. Johan Pouwelse, André de Kok, Joost Fleuren, Peter Hoogendoorn, Raynor Vliegendhart, and **Martijn de Vos**, Laws for Creating Trust in the Blockchain Age
In *European Property Law Journal*, 2017, De Gruyter.

Included in this thesis.