ELIAS ROHRER

ON THE SCALABILITY, RESILIENCE, AND PRIVACY OF DECENTRALIZED BLOCKCHAIN NETWORKS

ON THE SCALABILITY, RESILIENCE, AND PRIVACY OF DECENTRALIZED BLOCKCHAIN NETWORKS

vorgelegt von
DIPL.-INF.
ELIAS ROHRER

an der Fakultät IV: Elektrotechnik und Informatik
der Technischen Universität Berlin

zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr. Stefan Tai
1. Gutachter:	Prof. Dr. Florian Tschorsch
2. Gutachter:	Prof. Dr. Stefan Schmid
3. Gutachter:	Prof. Dr. Hannes Hartenstein

Tag der wissenschaftlichen Aussprache: 8. Dezember 2021

Berlin, 2021

Ordo renascendi est crescere posse malis.

— Rutilius Namatianus

ABSTRACT

Cryptocurrencies such as Bitcoin or Ethereum promise to establish themselves as decentralized alternatives to financial infrastructures that so far have been reliant on centralized trust models. These decentralized blockchain networks are built around the foundational principles introduced with Bitcoin's consensus protocol, in which a peer-to-peer network manages a globally distributed ledger of transactions—the blockchain. While this new and rather unorthodox approach to achieve Byzantine agreement in open and decentralized networks offers a number of promising properties and features, it suffers from very limited throughput scalability. As distributed systems under the pressure to scale are often at risk of neglecting other essential qualities, improving scalability while considering resilience and decentralization poses a fundamental challenge for the research on open blockchain networks today.

This thesis is therefore dedicated to the study of blockchain scalability from a computer networking perspective. In this regard, we focus on the two main approaches towards blockchain scalability—on-chain and off-chain scaling—, study the currently deployed state-of-the-art protocols and architectures, and propose improvements that consider decentralization, security, and privacy first-class design goals.

As on-chain scalability has been previously shown to be highly dependent on the reliability and performance of the underlying networking layer, the first part of this thesis studies the peer-to-peer networks utilized for block and transaction propagation. To this end, we research the Bitcoin and Zcash networks through longitudinal measurement studies and enable their model-based evaluation through the introduction of a network-centric simulation framework. Furthermore, we present Kadcast, a new transport protocol based on a structured overlay network. We show that Kadcast enables faster and more efficient block and transaction propagation while maintaining decentralization.

In the second part, we direct our attention to the notion of payment channel networks, which promise to improve scalability by processing most transactions off-chain. To this end, we study the provisionings and characteristics of Bitcoin's Lightning Network and analyze its resilience to random failures and targeted attacks. Moreover, we study the feasibility of timing attacks on privacy and show how on-path adversaries benefiting from network centralization may compromise user privacy. Furthermore, we study possible attachment strategies according to which new nodes may join the network and how they impact the network's topology and efficiency long-term.

Lastly, we show how the foundational principles of blockchain networks may be applied beyond cryptocurrencies in order to create decentralized infrastructures with improved security properties and reduced trust requirements. To this end, we introduce Webchain, a decentralized system enabling reference verifiability for the World Wide Web.

ZUSAMMENFASSUNG

Kryptowährungen wie Bitcoin oder Ethereum versprechen sich als dezentrale Alternativen zu den bisher zentral organisierten Finanzinfrastrukturen zu etablieren. Diese dezentralen Blockchain Netzwerke basieren auf den fundamentalen Prinzipien von Bitcoins Konsensprotokoll, in welchem ein Peer-to-Peer Netzwerk ein dezentrales Kontenbuch aller Transaktionen, die Blockchain, verwaltet. Auch wenn dieser neue und tendenziell unorthodoxe Ansatz byzantinischen Konsens in offenen und dezentralen Netzwerken herzustellen einige vielversprechende Eigenschaften und Vorzüge bietet, so leidet er doch unter der begrenzten Transaktionsdurchsatzskalierbarkeit. Da verteilte Systeme unter Skalierungsdruck oft dazu neigen andere essentielle Qualitäten zu vernachlässigen, stellt derzeit die Frage wie man die Skalierbarkeit verbessern und zugleich die Resilienz und Dezentralität berücksichtigen kann eine fundamentale Aufgabe für die Forschung an offenen Blockchain Netzwerken dar.

Diese Dissertation is daher dem Studium der Skalierbarkeit von Blockchain-basierten Systemen aus der Perspektive der Computernetzwerke gewidmet. Dahingehend konzentrieren wir uns auf die zwei zentralen Ansätze für Blockchainskalierbarkeit (on-chain und off-chain Skalierung), untersuchen den Zustand der derzeitig verwendeten Protokolle und Architekturen, und machen Verbesserungsvorschläge, die Dezentralität, Sicherheit und Privatheit berücksichtigen.

Vorherige Arbeiten haben bereits die Abhängigkeit der on-chain Skalierbarkeit von der Leistungsfähigkeit und der Zuverlässigkeit der Netzwerkschicht gezeigt. Der erste Abschnitt der Dissertation befasst sich daher mit den Peer-to-Peer Netzwerken, die bei der Transaktions- und Blockverteilung Verwendung finden. In dieser Hinsicht erforschen wir Netzwerke von Bitcoin und Zcash durch langfristige Messstudien und ermöglichen ihre modellbasierte Untersuchung durch die Entwicklung einer netzwerkzentrierten Simulationssoftware. Darüberhinaus präsentieren wir Kadcast, ein neues Transportprotokoll welches auf einem strukturierten Overlaynetzwerk basiert. Wir zeigen, dass Kadcast schnellere Block- und Transaktionsverteilung ermöglicht und dennoch die Dezentralität des Netzwerkes gewährleistet.

Im zweiten Abschnitt richten wir unsere Aufmerksamkeit auf das Konzept der Payment Channel Networks, welche eine Verbesserung der Skalierbarkeit ermöglichen da sie die meisten Transaktionen off-chain verarbeiten. Dahingehend untersuchen wir die Charakteristiken des Bitcoin Lightning Netzwerks und analysieren seine Resilienz gegenüber zufälligen Ausfällen sowie zielgerichteten Angriffen. Außerdem untersuchen wir die Möglichkeit von zeitbasierten Angriffen auf die Pri-

vatheit und zeigen, dass ein auf dem Übertragungspfad positionierter Angreifer die Nutzerprivatheit kompromittieren kann. Darüberhinaus betrachten wir verschiedene Strategien gemäß denen sich neue Knoten zum Netzwerk verbinden können, sowie ihre langfristigen Auswirkungen auf die Eigenschaften der Netzwerktopologie.

Zuletzt zeigen wir, wie die fundamentalen Prinzipien von Blockchain Netzwerken auch über das Anwendungsfeld von Kryptowährungen zum Einsatz kommen können um dezentrale Infrastrukturen mit verbesserten Sicherheitseigenschaften und reduzierten Vertrauensanforderungen zu entwerfen. Dahingehend stellen wir Webchain vor, ein dezentrales System das Referenzverifizierbarkeit für das World Wide Web ermöglicht.

ACKNOWLEDGMENTS

This research adventure started in 2017, when Florian Tschorsch asked me if I could imagine doing a PhD with him at TU Berlin.

First and foremost, I therefore want to thank Florian for this opportunity, as well as for the ongoing trust and support. Not only did he always have an open ear, but also offered advise and guidance when needed. Moreover, he is is a great mentor and a very dedicated co-author, from which I had the chance to learn a lot. Without our invaluable discussions, this thesis would not have come to fruition. Thank you!

Secondly, I would like to thank my peers at the DSI research group: Christoph Döpmann, Erik Daniel, Robert Muth, and Saskia Nuñez von Voigt. Sharing an office and working together was always fun, even though in the last one and a half years we had to re-locate our shared space to the interwebs. I really enjoyed the open and collegial atmosphere at DSI, which encouraged the free exchange of ideas. So long, and thanks for all the cake!

This thesis is in part based on contributions that emerged from collaborative research efforts. I therefore want to thank my fellow co-authors: Erik Daniel, Julian Malliaris, Kimberly Lange, and Steffen Heidel. Thank you for these very productive and inspiring collaborations.

In my time at TU Berlin, I supervised 20 projects and theses at the bachelor's and master's level. For me, working with other students on a wide range of interesting topics was always fun and a great opportunity to learn. As I am convinced that my own research significantly benefited from the gained insights and the breadth of these topics, I have to express many thanks to the students I had the chance to work with: Alexander Wagner, Hammad Abdullah, Jan Preßler, Julian Malliaris, Kimberly Lange, Marcel Reppenhagen, Niklas Gögge, Peiran Sun, Sergej Eichwald, Steffen Heidel, Thomas Wirth, Tim Brachwitz, Wolfgang Landes, Vinothkumar Nagasayanan, and Yuriy Strela.

Finally, I want to thank my Nuria for all the love and support throughout the last four years.

CONTENTS

III Beyond Cryptocurrencies

1

INTRODUCTION

1.1 MOTIVATION

Ever since the introduction of the Bitcoin peer-to-peer electronic cash system by Satoshi Nakamoto in 2008 [Nak08], the notion of blockchain networks has spawned a movement towards a novel, steadily growing ecosystem of decentralized infrastructures. At the same time, the underlying protocol architecture could also establish itself as a viable solution to achieve state replication in open computer networks. In this *Nakamoto consensus protocol*, all network participants maintain a shared ledger of transactions, which is segmented into blocks—the *blockchain*. As new transactions are introduced to the network, they are collected by miners, secured through a cryptographic proof-of-work scheme, and the resulting blocks are disseminated in the network. Since all nodes upon receipt can independently verify the validity of these updates to the local blockchain state, the protocol enables verifiably secure transaction processing without the necessity to trust any central authority. While previously distributed systems based on classical consensus algorithms faced many limitations in open and Byzantine network settings [FR03], the seemingly unorthodox approach introduced by the Nakamoto consensus protocol was able to stir up the field of distributed computing by challenging a number of certainties that were deemed unshakable before.

In particular, ever since Fischer et al. [FLP85] gave the first proof that achieving binary consensus in the presence of failures is impossible in asynchronous message passing systems and Lamport et al. [LSP82] showed that agreement cannot be reached in networks with more than one third Byzantine nodes, the theoretical guarantees provided by distributed systems seemed severely limited. In order to overcome these theoretical boundaries, previous approaches such as Practical Byzantine Fault Tolerance (PBFT) [CL99] rely on strongly identified sets of validators and apply weakened liveness requirements in order to enable Byzantine consensus in real-world network settings. In contrast, the Nakamoto consensus protocol combines relaxed consistency assumptions with a proof-of-work leader election scheme based on economic incentives, enabling it to reach Byzantine consensus in open and decentralized network settings even without presupposed knowledge of strong participant identities.

Just as the Nakamoto consensus protocol in this regard can be considered a breakthrough in the field of distributed systems, the emerging decentralized blockchain networks promise to hold the potential to revolutionize the financial sector and other areas today reliant on centralized trust models for secure transaction processing. However, the actual utility of these new peer-to-peer networks is currently still severely restricted by bottlenecks inherent to their design: as the protocol requires that blocks are produced in roughly regular intervals, the maximum feasible block size is limited by the amount of data that can be disseminated to network participants before a consistency bound is reached. To this end, the Nakamoto protocol has been proven to achieve consistency and liveness in bounded-delay networks, i. e., only under the assumption of a partial-synchronous network model in which message delivery between participants is bounded by such an upper delay limit [GKL15; GKL20; KRS18; PSS17].

Since the size of a block also linearly corresponds with the number of included transactions, the possible transaction throughput of the Bitcoin system is severely limited [Cro+16]. As this renders block space a scarce resource that is heavily competed for during peak times, it may furthermore lead to high transaction fees [@Blo21b] and confirmation times [@Blo21a]. Similar limitations are shared by all popular cryptocurrencies based around the principles of Nakamoto-like proof-of-work consensus such as, for instance, the Ethereum network [@But14]. As of April 2021, Ethereum is regularly facing heavy congestion resulting from its inability to process transactions fast enough to keep up with user demands [@Eth21]. As a consequence, transaction confirmation may be delayed and fees spike, i. e., the network likewise exhibits temporarily degraded consistency properties, which lead to increased costs and an impaired user experience.

1.2 CHALLENGES AND OBJECTIVES

Given the increased interest that is driven by the advancing real-world adoption of cryptocurrency systems, improving transaction throughput is currently a central goal for research on blockchain networks. However, distributed systems under the pressure to scale are often at risk to resort to centralized trust models in order to account for increased performance requirements [Tro+17]. Therefore, improving scalability while maintaining or even increasing resilience and decentralization poses a fundamental challenge for open blockchain networks. Moreover, as decentralization of the trust models underlying distributed systems does not guarantee—and in some cases may even be detrimental to—the privacy of network participants [Tro+17], the need for private transaction processing has to be considered an additional challenge when designing scalability solutions for decentralized blockchain networks.

There are two main approaches to increase the scalability of decentralized blockchain networks: *on-chain* and *off-chain* scaling. The on-chain approach aims to alleviate inefficiencies in the blockchain protocol stack in order to push the performance of these systems as close as possible towards their theoretic boundaries, so that more transactions per second can be processed. To this end, proposed improvements on the consensus protocol for instance include a decoupling of the leader election process from transaction serialization [Eya+16] or applying alternate chain selection rules [SZ15]. However, previous work has also shown the securely achievable throughput of the blockchain protocol to be heavily dependant on properties of the networking layer [DW13; ES14; Ger+16]. The first part of this thesis is therefore dedicated to the study of currently deployed blockchain peer-to-peer networks, how they can be adequately measured and modelled, and how the information dissemination process in these networks can be implemented in a more efficient manner without relinquishing decentralization.

On the other hand, the off-chain scaling approach follows the idea that—by introducing another level of indirection—not all transactions have to be immediately processed by the consensus layer. Instead, transactions may be securely offloaded to a second-layer protocol that operates on top of the consensus layer and relies on it for conflict resolution. While a number of such protocols have been proposed in recent years [Gud+20], the notion of payment channel networks (PCNs) is among the most promising solutions to scale decentralized blockchain networks through a second layer. In PCNs such as Bitcoin's Lightning Network [PD16], payment channels are established between network nodes, which then can be used by the involved parties to process most transactions locally rather than requiring global agreement on a shared ledger state. While this approach generally allows for much higher transaction throughput, a novel set of challenges arises from this system model: as PCNs form dedicated overlay networks over which payments are routed in a multi-hop fashion, it is unclear which impact the emerging network structures and the utilized network algorithms have on the payment properties PCNs can provide to their user base. The objective of the second part of this thesis is therefore to study how the employed algorithms and topological properties influence payment efficiency and success, as well as how network centralization influences the resilience and privacy guarantees of payment channel networks.

While blockchain technology in recent years established itself as a dedicated area of research, it stands on the shoulder of many giants from the fields of distributed computing and cryptography. Even though to date the primary use cases for blockchain technology are financial applications such as cryptocurrency networks, some of the novel ideas coming from blockchain research may be applied to improve decentralization and security of distributed systems' design overall. However, adopting such concepts without necessarily also carrying over some of the underlying assumptions on economic incentives can prove to be quite challenging. To this end, the third part of this thesis explores how some of the foundational principles and building blocks of blockchains can be reasonably adopted to build resilient and decentralized distributed systems for use cases beyond cryptocurrencies.

In summary, the objective of the present thesis is to study and improve the challenge of blockchain scalability from a networking perspective, with a particular focus on decentralization, resilience, and privacy.

1.3 CONTRIBUTIONS

The main contributions of this thesis can be summarized as follows:

- We conduct a longitudinal measurement study on the Bitcoin peer-to-peer network and extract essential parameters to create a geographically clustered model of its overlay topology. On the basis of this model, we implement `bns`, a new network-centric simulation framework that enables the empirical analysis of large-scale blockchain network scenarios. We validate the simulation against related work and real-world measurements, and show that the utilized protocols as well as miners' geographic location have a significant impact on the performance and security of blockchain networks (Chapter 3).

- We expose the Zcash network through the first longitudinal measurement study on its peer-to-peer networking layer. Our study captures key metrics spanning multiple protocol update cycles. Moreover, we present an inference method based on the timing of block arrivals that allows to determine the interconnections of nodes. We furthermore evaluate and verify this method through simulations and real-world experiments (Chapter 4).

- We present Kadcast, a new transport protocol for the propagation of blocks and transactions in blockchain networks. Kadcast is based on a structured overlay topology that enables faster and more efficient broadcast operations. We analyze Kadcast's reliability, security, and privacy properties and evaluate its performance, efficiency, and security through extensive network simulations utilizing the `bns` framework. Furthermore, we confirm its superior performance through the comparative deployment of a QUIC-based prototype implementation in a large-scale cloud-based testbed (Chapter 5).

- We analyze the topology of Bitcoin's Lightning Network through graph-theoretic measures and show that it can be classified as a small-world and scale-free network. Moreover, we investigate its resilience to random failures and targeted attacks. In this regard, we introduce the notions of channel exhaustion and node isolation and show that the network is susceptible to these kinds of attacks. In particular, we show that the network's tendency towards centralization may enable network partitioning attacks run by a strategic adversary (Chapter 7).

- We explore what payment success could be gained if PCNs would follow a multi-path route selection approach. To this end, we model payments as network flows and identify that routing multiple concurrent payments is reducible to the multi-commodity flow problem. Moreover, we discuss the possibility of a distributed routing algorithm that takes multiple routing demands into account (Chapter 8).

- We show that the privacy guarantees of the Lightning Network may be subverted by an on-path adversary conducting timing attacks on the HTLC state negotiation messages. To this end, we provide estimators that enable an adversary to reduce the anonymity set and infer the likeliest payment endpoints. We show attack feasibility through a proof-of-concept implementation and evaluate the adversarial success in model-based network simulations. We find that controlling a small number of central malicious nodes is sufficient to observe a large share of all payments and that adversaries of different magnitudes could employ timing-based attacks to deanonymize payment endpoints with high precision and recall (Chapter 9).

- We provide an empirical study on the impact of various attachment strategies for payment channel networks. To this end, we introduce candidate strategies from the field of graph theory and analyze them with respect to their computational complexity as well as their repercussions for end users and service providers. Moreover, we evaluate their long-term impact on the decentralization and performance of the PCN topology (Chapter 10).

- We introduce Webchain, a new distributed system enabling referential and citation provenance for the World Wide Web. The Webchain architecture adapts essential concepts from blockchain technologies to facilitate the verifiably of online sources. To this end, we integrate a distributed timestamping scheme and analyze Webchain's security properties in the face of forging attacks. Moreover, we provide a working prototype implementation of the Webchain system and utilize it to evaluate Webchain's fault tolerance and attestation delay in emulated network scenarios (Chapter 11).

In addition to these main contributions, we discuss the state of PCN research in Chapter 6 and now turn the attention to the state of research on blockchain networks in the following Chapter 2.

Part I

NETWORKING IN BLOCKCHAIN SYSTEMS

2

STATE OF BLOCKCHAIN NETWORKS

In 2008, Satoshi Nakamoto introduced the Bitcoin peer-to-peer electronic cash system [Nak08]. While transaction systems that facilitate the electronic and private transfer of money have been explored in literature at least since the 1980's [Cha85; CFN88; Dai98; Fin04; Sza05], approaches that predate the introduction of Bitcoin were deemed impractical at the time or relied on the existence of trusted intermediaries. The central novel concept introduced by Bitcoin is the *blockchain*, a chain of cryptographically linked blocks that implement a shared ledger in which an account of all conducted transactions is kept. This shared ledger is replicated to all network participants and functions as a common and independently verifiable source of truth.

In the years since then, a variety of blockchain architectures and consensus protocols have been proposed and implemented, most of which are however descendant from the ideas introduced by Bitcoin. In this chapter, we therefore elucidate the basic functionality of blockchain networks by reference to the foundational principles of the Nakamoto consensus protocol. Moreover, we present a taxonomy on the parts composing the peer-to-peer network layer underlying blockchain systems, and introduce specifics and advancements of the networking stacks found as part of the Bitcoin implementation. Finally, we discuss previous work from literature related to our contributions towards the blockchain networking layer.

2.1 THE NAKAMOTO CONSENSUS PROTOCOL

One of the major issues arising in digital currencies is the *double-spending* problem: as data can be easily copied, any naïve form of electronic cash faces the issue that digital "coins" representing a certain value may be duplicated and spent more than once. This kind of counterfeiting leads inevitably to the inflation and devaluation of the currency. The primary function of the Nakamoto consensus protocol is therefore to enable the network participants to eventually agree on a shared and verifiable view of a global ledger that records the ownership and transfer of funds, thereby mitigating the possibility of counterfeit.

2.1.1 *Addresses and Transactions*

In this regard, the Nakamoto consensus protocol introduces the notion of *addresses* that can be used to transfer funds by means of *transactions* between them. A user can be in control of many Bitcoin addresses and is able to send transactions secured by the use of a digital signature scheme. In particular, Bitcoin addresses are derived from the public part of an ECDSA `secp256k1` [Bro10] key pair, and spending any funds belonging to an address requires a corresponding signature.

As Bitcoin follows the so-called Unspent Transaction Output (UTXO) model, transactions have a number of *inputs* that reference previously unspent (i. e., *locked*) funds and *outputs* that allow anyone with the required secret key material to *unlock* and spend these funds by supplying a matching digital signature. To this end, Bitcoin implements an imperative programming language that enables the processing of simple stack-based scripts allowing for signature verification. As a consequence of the UTXO model, funds may only be merged by referencing them in the same transaction and divided by reallocating them to different outputs. When the transactions are processed by the Bitcoin system, the consensus rules enforce that the sum of the funds spent in output scripts has to be less or equal to the sum of the unlocked funds referenced by the transaction's inputs. As furthermore any difference of input and output funds is regarded as a transaction fee, additional outputs that redirect the change towards addresses controlled by the user have to be provided, if not the entirety of input funds should be spent.

Since the transaction ledger is public data, sending transactions over and over from the same address would make it trivial for anyone to monitor the entire transaction history of a user. It is therefore considered best practice to keep user addresses as private as possible and always use freshly generated addresses for change. However, as the usage of change addresses is a standardized transaction pattern, this does not provide a high level of obfuscation. In fact, given that the Bitcoin transaction scheme in its current state can only provide pseudonymity rather than true anonymity to their users [PK00], literature has shown that user entities can successfully be re-identified based on the patterns of the public transaction graph [Mei+13; RH13].

2.1.2 *Blocks, Blockchain, and Mining*

In order to increase network synchronicity, new transactions are not individually included in the global ledger state, but are first collected in *blocks*. These blocks are regularly appended to the *blockchain* data structure representing the ledger, which essentially implements a distributed timestamping scheme [HS91; MAQ99]. As shown in Figure 2.1, blocks are composed of two parts: the header and the body. The transactions are stored in the block body and are secured against manipulation by the use of cryptographic hash pointers. In the case of Bitcoin, each transaction is hashed twice with SHA-256 [FIP15] and inserted into a Merkle tree [Mer87] structure that yields a single root hash securing the entirety of the block's transactions. The root hash is included in the block header which also encompasses among other vital fields a

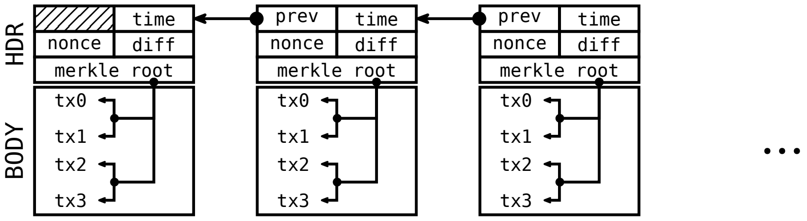


Figure 2.1: New blocks are appended to the blockchain.

hash pointer reference to the previous block in the blockchain and a verifiable solution to a cryptographic puzzle.

In this regard, Bitcoin applies a proof-of-work scheme similar to the Hashcash algorithm [Bac02], the purpose of which is to ascertain that only entities that invested a certain amount of computational resources gain the right to append to the blockchain. This part of the consensus protocol therefore on the one hand functions as a distributed leader election algorithm, but at the same time is an effective and decentralized mechanism for mitigating Sybil attacks [Dou02] in open networks without access control based on strong identities. To this end, the consensus algorithm dictates that the `nonce` given as part of the block header solves a hash puzzle according to a certain *difficulty* target (cf. Figure 2.1), i. e., it fulfills the equation $H(\texttt{HDR}) < \texttt{diff}$. The difficulty is a network-wide consensus parameter that every participant client re-adjusts based on historic data every 2016 blocks, so that on average every 10 minutes one solution is found. Due to the avalanche effect [Fei73] of cryptographic hash functions, the *miners* searching for the next block solution have no other option than to hash the block header with many different nonce values, and test whether the result conforms to the consensus requirement. This brute-force mining process is therefore completely random and memoryless [Ros11].

Once a solution is found, a new block is created and propagated in the network by the miner. As other peers receive this block, they verify its validity according to the consensus rules and append it to their local state of the blockchain. Since blocks may be mined simultaneously, two blocks may end up referencing the same predecessor in the chain. This introduces the notion of a *blockchain fork* which is resolved eventually by the algorithm through application of the *longest chain rule*: whichever of the competing branches is extended first wins, while all other branches are discarded. Previous work has shown the longest chain rule to make reverting a previously included block exponentially harder, the more blocks are appended in its succession. As a consequence, double spending attacks become highly infeasible, as long as no single party controls more than 50% of the network's hash rate [Ros12].

2.1.3 *Consensus Security and the Network Layer*

The central idea that an honest majority of peers individually validating local state transitions facilitates a global agreement is however based on the assumption that peers receive new blocks soon after their creation. In this regard, recent analytical works focusing on the consensus layer proved that the consistency guarantees of the Nakamoto consensus protocol only hold when blocks are delivered faster than a certain delay limit [GKL15; GKL20; KRS18; PSS17]. Otherwise occurring inconsistencies are undesirable, because they facilitate attacks such as double spending [Ger+16; SZ15].

In an early study, Decker and Wattenhofer highlighted the importance of the block propagation delay for the security of the Bitcoin system [DW13]. They showed that block propagation in Bitcoin's unstructured overlay network follows a long-tailed distribution in which at the time 5% of the peers had to wait more than 40 seconds to receive newly mined blocks. The authors also showed that the propgation delay increases the probability of blockchain forks.

Eyal and Sirer discovered the feasibility of *selfish mining* attacks on the Bitcoin protocol [ES14]. They show that a malicious miner could gain an advantage by withholding mined blocks. The main idea is to let other miners "waste" their computational power on an old block, while the selfish miner (secretly) mines a new block. When other miners find and propagate a block solution, the selfish miner quickly broadcasts its secret block and therefore still has a chance to "win the race" for the longest chain. The authors show that with this strategy, a selfish miner can increase its block reward, relative to the actual fraction of mining power it contributes to the system. The success of this attack, however, heavily depends on the attacker's mining power and the share of the network it can reach with its secretly mined block first. Improving the performance of the networking layer therefore reduces the attack surface for selfish mining attacks, as the time window for the successful publication of selfishly mined blocks shrinks.

More recently, the impact of block propagation on Bitcoin's resilience towards selfish mining and double spending [KAC12] attacks was further investigated by Gervais et al. [Ger+16]. The authors show that the occurrence of stale blocks, i. e., blocks that do not get included in the final longest chain and therefore do not contribute to its security, can have severe negative consequences for the performance and the security of proof-of-work-based blockchain networks. Their results suggest that the stale block rate is influenced by the block propagation delay, and that it heavily depends on the employed network-layer protocols.

We therefore conclude that the Nakamoto consensus protocol ensures eventual consistency and liveness properties, if blocks and transactions are propagated promptly to all network participants. This emphasizes the integral role of the different properties and aspects of blockchain networking layer, to which we turn our attention in the following.

2.2 NETWORKING LAYER TAXONOMY

In terms of a localization in an overall model of computing architecture, blockchains are located on the application layer. The networking stack of blockchain systems is therefore situated at the lower end of the application layer with some overlap to the layers beneath. As everything below the networking layer may have a significant impact on its performance and behavior, we in the following ascend the stack and discuss how patterns, technologies, and design choices are relevant to blockchain networking in general and our work in particular.

MEMBERSHIP MANAGEMENT Blockchain networks may be differentiated based on their approach to membership management. While *open* networks allow everyone willing to participate, *closed* networks are access-controlled and only allow a limited set of nodes. Moreover, different systems may employ different protocols for *neighbor discovery* and *neighbor selection*. While some systems randomly choose neighbors uniformly from all available peer addresses, others do so based on variety of different rules, e. g., limiting the share of chosen nodes from each autonomous system (AS). This functions as a basic form of *Sybil prevention* [Dou02], a field of research with much relevance to open blockchain networks [Bor06; VCS03].

NETWORK CONDITIONS Each blockchain network operates based on different pre-determined conditions regarding the underlying network infrastructure. The networks deviate in the *geographical distribution* of their nodes around the globe, and hence exhibit a number of *regionally dependent properties*, such as inter-peer latencies, provisioned bandwidths, and routing particularities. Moreover, the *reliability* or average uptime of peers, i. e., the existence of a core infrastructure with a low churn rate, has a tremendous impact on network stability [Imt+19; SR06]. In our work, we study the empirical network conditions of Bitcoin (Chapter 3) and Zcash (Chapter 4).

OVERLAY NETWORK TOPOLOGY The network topology emerging from the chosen peer-to-peer protocol has significant consequences regarding the efficiency and performance of the blockchain networking layer. While Bitcoin introduced the prevalent paradigm in which nodes form an *unstructured* overlay topology by establishing connections to randomly drawn neighbor nodes, other interconnection models, such as for example the introduction of supernodes, have been studied in peer-to-peer literature [YG03]. Similarly, currently deployed block distribution networks, such as Bitcoin FIBRE [@FIB21] resort to *federated* network structures and recent works discuss the possibility of location-based node clustering [BYZ18; SOA17a]. Moreover, our work presented in Chapter 5 studies the benefits of a *structured* overlay topology.

TRANSPORT PROTOCOL How the data is transmitted between nodes is up to the transport protocol. While the *connection-based* TCP is currently the default way of ensuring reliable transmission over the unreliable internet, some protocols [@FIB21; Kla+] opt for the more lightweight *datagram-based* UDP protocol in combination with forward error correction (FEC). The QUIC transport protocol [Lan+17] is also a promising candidate for fast an reliable communications in blockchain networks, a notion which we explore further in Chapter 5.

DISSEMINATION SCHEME While in blockchain networks generally most data items have to be delivered to all nodes in the network, the specifics are determined by the respective dissemination protocol. For example, while in the case of Bitcoin transactions are propagated using the diffusion spreading protocol [FV17], block announcements are currently simply flooded in its unstructured overlay network. More recent works in literature have however proposed dissemination schemes that improve transaction privacy [Fan+18; VFV17], and we explore a more efficient block dissemination protocol in Chapter 5.

MESSAGING PATTERN Communication between peer-to-peer network nodes may be classified according to the applied patterns, such as *unsolicited* message propagation, *request/response*, or *publish/subscribe* schemes. Messaging patterns specific to the domain of blockchain networks entail improved relaying schemes based on probabilistic set reconciliation [@Cor16; Nau+19; Ozi+19]. In this regard, our work studies how messaging patters can impact the network utilization and propagation delay (Chapter 3 and Chapter 5).

Based on this general taxonomy of blockchain networks, we now turn our attention to the concrete implementation of these concepts in the currently implemented networking stack implemented by Bitcoin.

2.3 THE BITCOIN NETWORKING PROTOCOL

The backbone of the Bitcoin network is comprised of nodes that form an open and unstructured peer-to-peer overlay network: everyone who wants to participate in the network can setup a so-called *full node*, i. e., run a software which implements the Bitcoin peer-to-peer protocol and replicates the entire blockchain. While there are a number of projects that implement the Bitcoin protocol to some degree, we base our account of the protocol on the behavior of the reference client, Bitcoin Core [@Bit19].

The Bitcoin peer protocol is an unencrypted TCP-based network protocol in which nodes pick their neighbors in a randomized fashion: every node by default establishes 8 outgoing connections and, if reachable and configured, accepts up to 117 incoming connections, resulting in a maximum connection count of 125. By convention, nodes that accept incoming connections from other peers are called *servers*, and *clients*, if they only establish outgoing connections. The Bitcoin software keeps a local database of known peer addresses from which it randomly draws candidates for outgoing connections. If this database is empty, e. g., when the software is started for the first time, it is bootstrapped by querying a number of community-run DNS servers whose addresses are hard-coded in the client software. They return peer IP addresses as the contents of `SRV` resource records. Peer addresses are also gossiped to network neighbors and can likewise be requested by and from each network participant.

After a TCP connection is established, the nodes exchange `version` and `verack` messages which transport crucial peer data and additionally serve as basic handshake messages. Moreover, the exchange of peer address data, transaction forwarding, and block propagation is initiated. Every network participant may insert new transactions to the network, which are then propagated via Bitcoin's gossip protocol and are validated by every full node along the way. In particular, new transactions are first announced to neighboring nodes through `inv` (read: *inventory*) messages, which in this case are only sent after an exponentially distributed delay for privacy reasons. As mentioned before, this propagation scheme is known as *diffusion spreading* [FV17].

Transactions are validated, collected, and bundled into blocks by miner nodes which start mining the block, i. e., start calculating the solution to a cryptographic puzzle whose difficulty parameter is set based on network consensus. Once a solution is found, the new block is disseminated in the peer-to-peer network. In earlier versions of the Bitcoin protocol, blocks were announced via immediately forwarded `inv` messages, and receiving nodes would request blocks and headers independently with `getheaders` and `getdata` requests. These would then again get answered by corresponding `headers` and `block` messages.

However, since protocol version 70012 was introduced with Bitcoin Core v0.12.0, the default block propagation scheme changed [@Daf15]: blocks are announced directly by sending `headers` messages, which reduce the propagation delay. Yet, when more than one block has to be announced, the client falls back to the `inv`-based announcement scheme. Nodes enable this new protocol by sending a `sendheaders` message after the initial handshake. In addition, a scheme for *compact block relay* was introduced [@Cor16], which allows to send block announcements in a more bandwidth efficient way. In particular, it allows Bitcoin nodes to only retrieve the transaction data they are missing from an announced block, which can severely reduce the bandwidth overhead of block propagation, but is prone to induce an additional latency overhead.

While other blockchain networks may exhibit their own idiosyncrasies, they often coincide with Bitcoin's general networking paradigm, i. e., block and transaction propagation through broadcast in an unstructured overlay topology. For example, even though Ethereum's [@But14] peer discovery is based on a Kademlia [MM02] overlay construction, the actual `eth` propagation protocol follows a schematism similar to that of Bitcoin [@Eth19a; Hen+19]. However, at the time of writing, it chooses to forgo the request-response scheme over unsolicited block propagation and also does not implement discussed protocol extensions, such as a compact block relay mechanism [Kim+18]. Likewise, as the reference client of Zcash [@Ele21] was previously forked from Bitcoin Core, its network currently still operates on what is essentially an older version of the Bitcoin networking stack.[1]

2.4 RELATED WORK

In the following, we discuss work related to our contributions towards the field of blockchain networking.

2.4.1 *Network Measurement and Inference*

A large body of literature has previously studied various individual properties of real-world blockchain networks. While early entries were mainly concerned with the network topology and block propagation behavior [DW13; DPH14], more recent contributions measured the latency and bandwidth provisionings [Gen+18; Neu19], the mining power distributions [Ger+14; WCY19; WL15], as well as the node churn [Imt+19; Neu19] and transaction propagation behavior [Pap+18] of the Bitcoin network. Moreover, the peer-to-peer networks of Ethereum [Gen+18; Kim+18; Sil+20] and Monero [Cao+20] have been explored in literature. However, most entries do not entail recent measurement results that consider the current state of blockchain network provisioning. To this end, our work introduced in Chapter 3 provides a comprehensive model of the Bitcoin network based on an updated data set.

Beyond these blockchain networks, the Zcash [Ben+14a; @Ele21] cryptocurrency exhibits similar properties to Bitcoin, but provides additional privacy guarantees by utilizing zero-knowledge proofs [Ben+14b] to enable anonymous transaction processing. While most prior work on Zcash focuses on the anonymity aspects [Kap+18; Que17], our work presented in Chapter 4 introduces the first measurement study of the Zcash peer-to-peer network and explores methods to discover its topol-

---

[1] Notably however, the roadmap for a new implementation and networking stack has recently been announced by the Zcash foundation [@Val20].

ogy. Previous entries in literature studied the possibility of topology inference of blockchain networks based protocol details such as peer announcements [Mil+15] or transaction propagation mechanics [GNH18]. Notably, Neudecker et al. [NAH16] introduced a method to infer the topology of the Bitcoin network by analyzing the timing behavior of transaction propagation. Our work is fundamentally based on this concept, but extends this approach by translating it to the Zcash network setting and utilizing a more robust and finer-grained model based on the block propagation timing behavior.

2.4.2 *Models and Simulators*

In order to analyze their behavior, blockchain networks may be studied based on broad spectrum of models, tools, and simulations, reaching from testbeds with actually deployed prototypes to highly-abstracted simulated processes. Previously, Miller and Jansen [JH12; MJ15] proposed a simulator based on the actual Bitcoin Core source code, while abstracting lower-level network behavior. While this approach promises to closely model application behavior, its high complexity requirements tend to limit the size of the simulated scenarios. Contrastingly, the BlockSim simulator [AM18] highly abstracts from the application and network behavior, promising a lightweight simulation based on average delay values configurable by the end-user. The contributions most closely related to the simulation framework introduced in our work, are the Bitcoin network simulator by Gervais et al. [Ger+16], as well as the SimBlock [Aok+19; BS19; NBS20] simulator. While similarly to our approach the former makes use of the discrete-event network simulation provided by the ns-3 [@ns-21] framework, it relies on a simplified model of the network topology, such as establishing network links based on a random graph model congruent to the Bitcoin node's TCP connections. It thereby tends to yield idealized results. While SimBlock on the other hand adopts a more realistic geographical distribution of nodes and implements more recent protocol updates, such as compact blocks, it abstracts from the lower-level network protocols. In this regard, both entries do not allow to consider more complex network effects, such as congestion with resulting queuing delays and packet losses. In contrast, the `bns` simulation framework we introduce in Chapter 3 builds upon the ns-3 simulator and models the link, network, transport, and application layers of blockchain nodes independently, allowing to study a larger variety of (often interdependent) effects.

2.4.3 *Network-Layer Improvements*

In recent years, many contributions proposed changes to the network-layer protocols for transaction and block propagation in order to improve the resulting performance and privacy aspects.

For instance, schemes for improved transaction and block relaying have been discussed repeatedly in literature. Ozisik et al. [Ozi+19] proposed a more efficient block transmission protocol based on Bloom filters and Invertible Bloom Lookup Tables (IBLTs). This scheme augments the concept of compact block relaying [@Cor16] to enable higher bandwidth utilization by only retrieving transactions missing from the transaction memory pool, which however accepts the possibility of an increased worst-case block propagation delay. Similarly, multiple entries are concerned with enabling a more bandwidth-efficient transaction dissemination [Han+20; Nau+19], as well as utilizing forward error correction (FEC) in order to reduce bandwidth requirements of block propagation [Cha+19; ZWL21]. While improving on some aspects—such as the messaging overhead—these protocols do not fundamentally change the prevalent network and propagation models. In particular, they do not address issues that are inherent to unstructured overlay networks. In contrast, the Kadcast protocol we introduce in Chapter 5 tackles these issues by fundamentally rethinking the overlay structure as well as the dissemination protocol.

While furthermore a number of entries suggest to optimize neighbor selection with regard to propagation latency [Mao+20] or geographic proximity [SOA16; SOA17a; SOA17b], such approaches could lead to more centralized network topologies with unknown consequences for fairness and privacy. Third-party relay networks, such as the FIBRE [@FIB21] or bloXroute [Kla+] networks, are supposed to improve the distribution of blocks and transactions. While the emergence of these proposals clearly show the urgency of the problem, we deem them orthogonal to the goal of improving the peer-to-peer networks themselves. Moreover, first results from previous works suggest [Ger+16] that a separate relay network has a negligible effect over switching to a faster block propagation scheme. Since such federated relay networks also require central and manual coordination, they likewise do not necessarily align with the aspired design goals regarding decentralization.

While initially most research on the privacy of cryptocurrency transactions focused on the consensus layer [Mei+13; RS13], more recent work discussed deanonymization attacks on the peer-to-peer layer [BKP14; KKM14] and analyzed what privacy properties it provides [FV17]. Very recently, Tramèr et al. have shown that timing-based side-channel information and traffic analysis may be used to attack the privacy guarantees of Zcash and Monero cryptocurrency transactions [TBP20].

In order to mitigate privacy threats arising from the networking layer, a number of contributions deal with private transaction relaying. For example, Venkatakrishnan et al. and Fanti et al. propose protocol redesigns that improve anonymity of transaction propagation in the Bitcoin network [Fan+18; VFV17]. These designs utilize a new propagation method called "diffusion-by-proxy", which divides transaction propagation in two phases: first, in the "anonymity phase", transactions carry out a random walk in the network. Then, in the "spreading phase", the endpoint of the random walk starts performing the usual broadcast utilizing Bitcoin's diffusion spreading technique. While privacy aspects are not the main focus of the Kadcast protocol introduced in Chapter 5, techniques such as diffusion-by-proxy spreading may be easily integrated in its design in order to improve transaction privacy.

3

MEASURING AND SIMULATING THE BITCOIN NETWORK

This chapter is based on the publication: Elias Rohrer and Florian Tschorsch. "Blockchain Layer Zero: Characterizing the Bitcoin Network through Measurements, Models, and Simulations." In: LCN '21: Proceedings of the 46th IEEE International Conference on Local Computer Networks. 2021 [*RT21a*]

Given the complex effects and behaviors emerging in large-scale peer-to-peer networks, network-layer properties are often studied best in controlled environments such as network simulations or experimental testbeds. Establishing adequate simulation models however requires access to comprehensive empirical datasets on the grounds of which reasonable assumptions and input parameters can be determined. As prior studies on the Bitcoin network often focused on a number of individual aspects of the network, we in the following present an updated, comprehensive measurement study capturing a wide array of parameters. From the captured data set, we extract a network model that builds the foundation for realistic network simulations. We furthermore present `bns`, a modular simulation framework for blockchain networks that implements this model and allows for experimentation based on configurable networking stacks. Lastly, we validate the proposed simulation model by reference to third-party network measurements, and by comparison to related work.

3.1 BITCOIN MEASUREMENT STUDY

In order to characterize the network conditions currently found in the Bitcoin network, we conducted a network measurement study that allows us to categorize the network peers along the lines of seven regional clusters: North America (NA), South America (SA), Europe (EU), Oceania (OC), Asia (AS), Africa (AF), and China (CN).[1] In the following, we present and discuss the applied methodology and the measurement results.

3.1.1 *Data Rates*

In order to be able to characterize how fast Bitcoin network peers are able to propagate blocks, starting from April 1, 2020, we conducted a longitudinal measurement study recording the bandwidth distribution in the Bitcoin peer-to-peer network. In particular, since it is typically the limiting factor of internet access, we are interested in the upload bandwidth of Bitcoin nodes. For this, we developed a measurement utility that was deployed on seven nodes as close as possible to the geographical center of the seven regional clusters. In particular,

---

[1] The existence of a separate cluster for China is justified on the basis of earlier research that highlights its special role for blockchain networks [DRT19; Ger+14; KJL18].

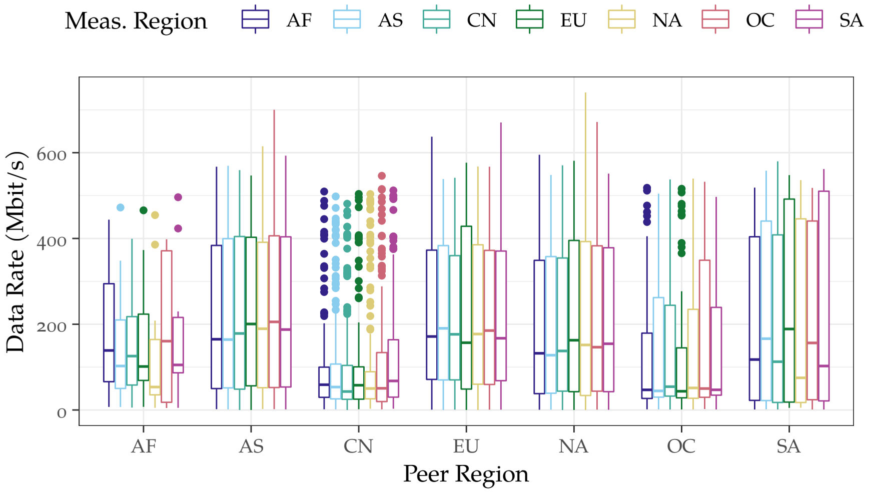


Figure 3.1: Measured Data Rates per Region

we deployed nodes at the following Amazon AWS regions: `us-west-1` (NA), `sa-east-1` (SA), `eu-central-1` (EU), `ap-southeast-2` (OC), `ap-south-1` (AS), `me-south-1` (AF), and `ap-east-1` (CN).

Once started, the measurement tool connects to one Bitcoin peer at a time and requests as many blocks as possible in a given time frame. In order to ensure bandwidth saturation, we configured the client to establish three concurrent connections to each of the node addresses publicly available from the Bitnodes [@Bit21b] database. After an initial offset to account for connection establishment, the download traffic was recorded and analyzed for five minutes utilizing `libpcap` [@Gro20], before moving on to the next address. During the overall runtime of around one month, each of the seven measurement nodes processed the node list in a randomized order to ensure we operate as close to line speed as possible as well as to minimize the risk of our study interfering with regular network operation. In order to approximate the upload speed of each peer, we processed the incoming data in intervals of 30 ms and recorded peak data rates. Moreover, the addresses were clustered based on the GeoLite2 [@Max21] geolocation database.

Every measurement node initiated connections to an average of 4,155 peers that could be successfully reached (from a total of 7,111 known peers). The resulting data rates are shown in Figure 3.1 for each peer region and in dependence of the measurement region. We observe that generally the data rates follow a wide spread of up to 740 Mbit/s as well as down to close to 0 Mbit/s. Moreover, the measured network bandwidth in NA, SA, EU, and AS regions are highest and similarly distributed, as they exhibit average peak rates of around 200 Mbit/s, 222 Mbit/s, 218 Mbit/s, and 224 Mbit/s, respectively. This suggests that peers in these regions supply the core infrastructure of the Bitcoin peer-to-peer network. The peers in the AF and OC regions are fewer and

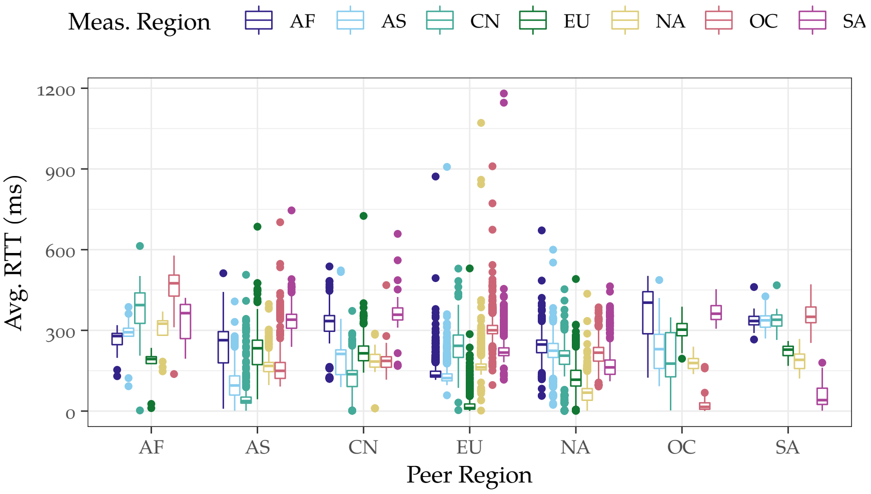


Figure 3.2: Measured Inter-regional Latencies

not as well connected, featuring average peak rates of 161 Mbit/s and 144 Mbit/s. Interestingly, the measured rates of peers located in the CN region are lowest with a mean peak rate of only around 106 Mbit/s. This observation is particularly notable as the regional distribution of blockchain networks have been discussed in literature for quite a while [DRT19; Ger+14] and these measurement results are in line with prior research that suggests that the so-called "Great Firewall" of China may pose a significant bandwidth bottleneck, which the authors link to detrimental miner behavior, such as creating empty blocks [KJL18].

3.1.2 *Latencies*

Besides bandwidth, inter-peer latencies have a major impact on the characteristics of message propagation in peer-to-peer networks. Therefore, we conducted an extensive measurement study with the goal of capturing latency distributions between the different geographical regions of the Bitcoin network. In April 2020, we therefore deployed seven additional measurement nodes in the NA, SA, EU, AF, AS, CN, and OC regions. After deployment, each measurement node ran a script sending 100 ICMP `ping` requests to each of the publicly available IP addresses of Bitcoin nodes and recorded the average encountered round-trip time (RTT) value. Afterwards, we collected the results and grouped them according to their source-destination regions i. e., from where the measurement was conducted and where the measured nodes were located.

From the 7,121 queried IP addresses, 2,519 did not respond to the ping requests on average, be it because they were not online anymore, or were blocking ICMP requests. This leaves us with measurement results for 4,602 Bitcoin peers. The measured average RTT values are

Table 3.1: Regional Peer and Mining Distribution

Region	EU	NA	AS	OC	SA	CN	AF
Peer Share (%)	49.1	41.4	4.7	1.8	1.3	1.0	0.6
Mining Share (%)	85.2	10.1	0.8	0.0	0.0	3.9	0.0

shown in Figure 3.2: we observed a mean RTT of 180 ms, which however exhibits a large standard deviation of 92 ms, as values range from less than 1 ms to the maximum outliers well surpassing 1,000 ms. We attribute measurements close to 0 ms to nodes in physical proximity (maybe even in the same data center). Moreover, peers located in the EU and NA regions are reachable the quickest, both featuring an overall mean RTT of 176 ms, while AF peers are the slowest to respond as they do so within 304 ms on average. In contrast to the bandwidth measurements, the latency of peers located in the CN region are not exhibiting significantly sub-par performance. However, our observation that ICMP packets seem not to experience similar effects actually further supports the hypothesis that the low data rates shown before are caused by the performance deficits induced by the Great Firewall's deep packet inspection.

3.1.3 *Peer and Mining Distribution*

In order to investigate the current regional distributions of network peers and mining power, a Bitcoin node was deployed in the network of TU Berlin. It was configured to run a modified version of the Bitcoin Core software that allowed for an unbound number of incoming and outgoing node connections. We let this node run for several days to acquire a high number of connected neighbor nodes, which eventually fluctuated around 2,500 connections, thus covering a large share of the Bitcoin network. From September 1, 2020 to September 30, 2020, we recorded incoming block announcements from all neighbor nodes. Given the good connectivity of our measurement node, we assume the node receives new block announcements from the miner directly or from a source close to the miner. Hence, we attribute the first observed announcement of a new block to the geographic region associated with the IP address we received it from, which corresponds to a *first-spy estimator* [VFV17].

As can be seen in Table 3.1, the distribution of network peers over the regional clusters is heavily skewed, as the highest share of peers is located in the EU region with 49.1 %, while the lowest share is located in the AF region with only 0.6 % of peers. The result also reveal a mining power distribution which is even more skewed than the peer distribution. Notably, the EU region provides 85.2 % of observed blocks, while

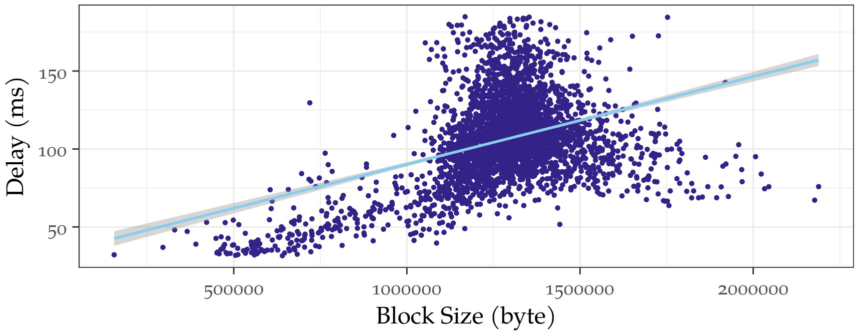


Figure 3.3: Observed Block Sizes and Validation Delays

only 49.1 % are located in this region. Moreover, the CN region exhibits an overproportional share of mining power of 3.9%, while providing only 1.0% of network nodes. The current picture is however a rather big change from a pilot study following the same methodology we conducted in the end of 2019, in which we observed an even more drastic inequality in mining power distribution: in November 2019, peers from the CN region still provided 77.6% of mined blocks, while only accounting for 3.9% of the network. While we observe this significant change, we currently have no clear indication on what exactly led to the shift of mining power distribution.

3.1.4 *Block Sizes and Validation Delay*

In order to get an understanding on how much data the network needs to process during block propagation, starting September 1, 2020, we utilized our measurement node to record newly published blocks over a period of one month. During this time, 4,087 blocks were published, for which the sizes can be seen in Figure 3.3. While the range of block sizes spanned everything from 200 bytes to 2.09 MB, the mean observed block size was 1.17 MB, which shows that the network is currently not constantly hitting its capacity limit.

Moreover, as each node only forwards new blocks after it validated its transactions and the proof-of-work, we recorded the time our measurement node took for validation. As seen in Figure 3.3, we observed a linear correlation between block size and validation delay. We furthermore validated this using Pearson's product-moment correlation test, which yielded a correlation coefficient $r = 0.41$ and $p < 2.2e - 16 \ll 0.05$, which allows us to reject the null hypothesis, i. e., suggests that there is indeed a significant correlation between block size and validation delay.

3.1.5 *Data Set and Measurement Ethics*

We make the source code of our measurement tools, the measurement data, and the inferred regionally clustered model accessible to the public.[2] Note that some of the recorded data, such IP addresses, are most likely also available from other public sources such as Bitnodes [@Bit21b]. However, for the sake of measurement ethics and in accordance with the Menlo report [Bai+12], we try to minimize our interference with the live network and hence treat such potentially identifying information as sensitive data. We therefore refrain from publishing the raw data set and instead publish data only in pre-processed form.

In summary, our measurements allow us to infer a comprehensive model of the Bitcoin network behavior, which we discuss in the following.

3.2 MODELLING AND SIMULATING THE BITCOIN NETWORK

In the following, we methodically model blockchain networks based on our measurements of the Bitcoin network.

3.2.1 *The `bns` Simulation Framework*

In order to capture the complex networking effects and behaviors of blockchain networks, we implemented `bns`, a new blockchain network simulation framework whose architecture is able to incorporate interchangeable networking modules and network topology models. The simulation is based on the ns-3 network simulator [@ns-21; Hen+08], whose discrete-event based architecture enables realistic and time-independent simulations of large computer networks. While the `bns` simulator follows a modular approach that allows for expandability and customizability of the networking stack, the default implementation is oriented towards the paradigmatic Bitcoin node logic and, as a baseline, currently implements a TCP-based networking stack that resembles Bitcoin's unstructured peer-to-peer overlay for block propagation. The `bns` codebase is open source and has been used by numerous researchers since its publication.[3]

ARCHITECTURE The main component of `bns` is implemented as a C++ program that creates different network scenarios using the ns-3 simulator. To this end, it spawns a configurable number of `ns3::Node` objects and configures network links between them. On each node, a blockchain-specific `ns3::Application` is installed, which is then run during the simulation process. As the different layers closely resemble

---

[2] See the companion repository: `https://git.tu-berlin.de/rohrer/blz-data`

[3] The source code is accessible under: `https://git.tu-berlin.de/rohrer/bns-public`

the real internet architecture, and all nodes and applications behave as independent actors, this simulation method captures detailed network effects and dynamics. In particular, and in contrast to previous works, TCP streams flow over shared links, thereby inducing queueing delays, and possibly even network congestion leading to dropped packets and retransmissions.

CONFIGURABILITY AND PARAMETRIZATION The simulator can be parametrized to reflect different blockchain systems and in order to experiment with different parameter sets. We chose the default parameter set of `bns` in reference to the Bitcoin network which therefore is able to mimic its wire protocol, i. e., it implements the inventory and header-based announcement schemes, as well as a stochastic model for block propagation based on compact blocks.

In order to provide a unified and controlled simulation environment, we base the parametrization of the simulator on the network model derived from our measurement study presented in Section 3.1. Specifically, simulated block sizes are sampled from our measurements and are optionally multiplied with a *block size factor* in order to simulate different block sizes. Likewise, blocks are only forwarded after a validation delay Δ_v calculated based on their size B_s corresponding to the linear equation

$$\Delta_v = \alpha + \beta B_s,$$

where α and β are taken from the measured validation delays, as discussed in Section 3.1.4.

Similarly, each miner schedules block generation events based on her hash power, overall resulting in a Poisson process spawning a block every ten minutes on average. Again, this may be modified by applying a *block interval factor*, e. g., a factor of 0.025 would result in Ethereum's 15 second target interval. As default, we simulate 10 mining pools that together provide nearly the entire hash power of the Bitcoin network [@Blo19]. To this end, `bns` currently supports the simulation of block propagation in two different network topologies, which we describe in the following.

3.2.2 *Topology Models*

HUB & SPOKE TOPOLOGY MODEL As a baseline model for protocol evaluation, `bns` implements a Hub & Spoke topology where all nodes are connected to one central node representing "the Internet". In this model, our assumption is that the internet is not a bottleneck and therefore we set the hub's default link capacities to 100 Gbps and client bandwidth to 50 Mbps. Moreover, we sample link latencies from the publicly available data set of measured end-to-end median latencies in the Bitcoin network [@Tec20]. The Hub & Spoke model is a typical

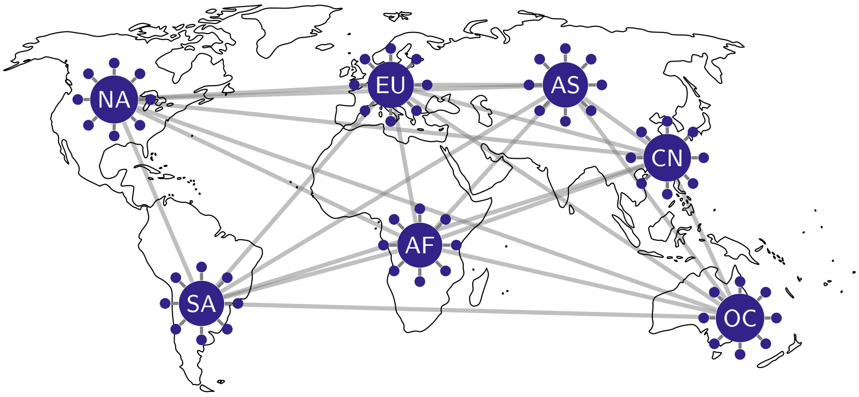


Figure 3.4: Geographic topology model consisting of seven regional hubs and associated peers.

setup for the assessment of peer-to-peer overlays, and while it captures rudimentary network effects, it does not rely on additional assumptions about the underlying topology. This creates an idealized simulation scenario that gives us the capability to assess the networking stacks based on a neutral, common ground.

GEOGRAPHIC TOPOLOGY MODEL In order to capture the protocol behavior in complex and more realistic settings, we employ a regionally clustered network model derived from the measured data set, i. e., recreate a network topology in the ns-3 simulator that relies on the seven regional node clusters. As can be seen in Figure 3.4, each of the clusters follows a Hub & Spoke model in which nodes are arranged around one of the region hubs, which in turn are fully interconnected.

The regional distribution of peers and miners is conducted according to our previously discussed findings. Moreover, data rates of nodes are drawn from a piecewise-linear distribution created based on our measurements, while inter-hub links are not assumed to be bottlenecks, i. e., are provisioned with really high data rates.

However, since our network measurements were conducted end-to-end, they already capture the inter- as well as intra-regional parts of the latencies. In order to be able to parametrize all segments of a peer-to-peer path, we therefore first create individual piecewise-linear distributions for each regional combination. We then establish intra-regional links in each region r and estimate individual link latencies $\hat{l}_{r,i}$ by sampling from the intra-regional distribution, divided by four. In the next step, we then need to create a model for each of the inter-regional links between all regions r_0 and r_1. For this, we first calculate the means of the provisioned intra-regional latencies $\overline{L_r}$, as well as the

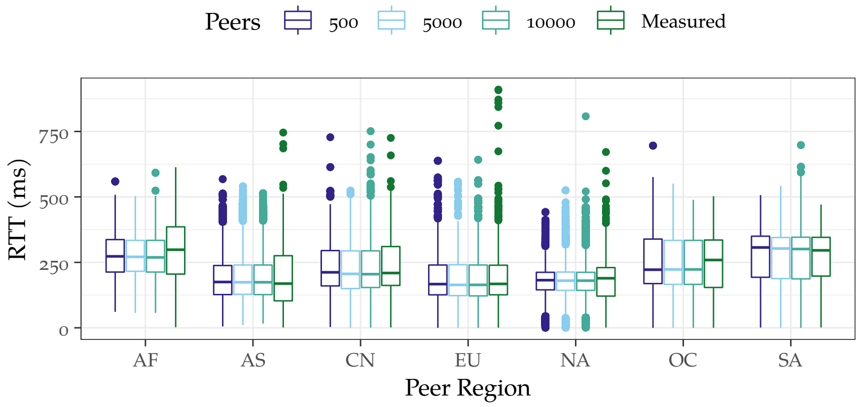


Figure 3.5: Simulated latencies for different network sizes in comparison to measured values.

mean measured peer-to-peer inter-regional latencies $\overline{L_{r_0,r_1}}$. We then calculate the estimated inter-regional link latency as

$$\hat{l}_{r_0,r_1} = \frac{\overline{L_{r_0,r_1}}}{2} - \overline{L_{r_0}} - \overline{L_{r_1}},$$

which is finally assigned to the corresponding edge.

3.3 NETWORK EXPERIMENTS

In the following we present empirical experiments showing the validity of the introduced network model as well as the capabilities of the `bns` simulation framework.

3.3.1 *Latency Model Validation*

In order to investigate whether the results gathered from simulated network scenarios of different magnitudes still fit the real-world conditions, we validate the latency model of the `bns` simulation framework. To this end, we implemented a `ping` application that was deployed on random nodes in each of the seven regions of the geographic topology model. Each instance of the application was configured to retrieve latency measurements to random nodes located in all seven regions, a process which was repeated fifty times for each regional combination and fifty times overall in order to ensure the statistical significance of the results. The simulations were furthermore run in scenarios of different magnitudes, i. e., in networks with 500, 5,000, and 10,000 peers.

The simulation results are shown in Figure 3.5 in comparison to the real-world latencies we retrieved as part of our measurement study. We observe that in all cases the latency distribution is very stable and independent of the number of network peers. This indicates that the `bns` simulation framework is able to yield expressive results, even when

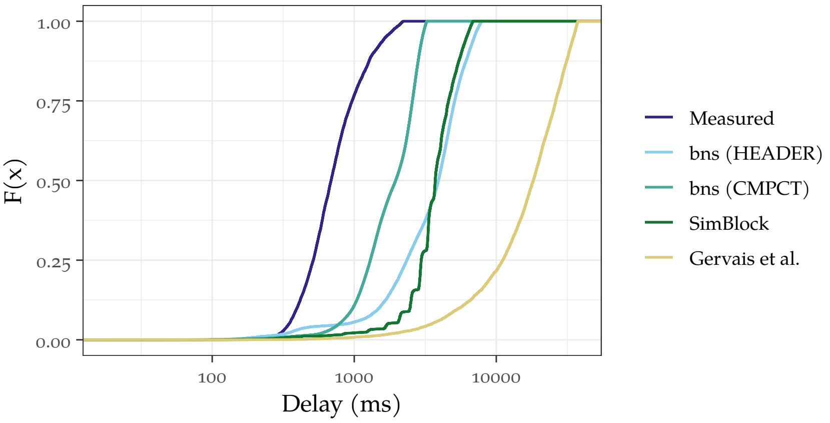


Figure 3.6: Block propagation times as simulated by `bns` in comparison to real-world measurements and results by other blockchain simulators.

only smaller scenarios are considered. Moreover, the simulated latency distribution fits the measured real-world distribution very well, with means diverging only about 1-6 ms in most cases.

3.3.2 *Propagation Model Validation*

To further investigate whether the simulated geographically clustered network topology is able to produce valid results that closely resemble the properties of real-world blockchain networks, we compare block propagation times of `bns` with related work as well as independent real-world data. To this end, we simulated network scenarios with 500 nodes utilizing `bns`, the SimBlock [Aok+19] simulator, as well as the simulator introduced by Gervais et al. [Ger+16]. We moreover retrieved block propagation data collected by Neudecker [Neu19; @Tec20] as ground truth.

Figure 3.6 shows the simulation results in logarithmic scale as cumulative distribution function, which allows a visual comparison of the data. In order to quantify how well the simulated data sets approximate the measured data, we additionally calculated the *root mean squared error* (*RMSE*). We observe that SimBlock (RMSE: 3,345 ms) yields results very similar to `bns` (RMSE: 3,590 ms) when the header-based propagation method is used. However, when compact block relaying is enabled, `bns` resembles the measured real-world data most closely, resulting in an RMSE of 1,415 ms. This does not come as a surprise, since Bitcoin's introduction of compact blocks [@Cor16] indeed reduced the average block propagation delay. The characteristics of the data retrieved from Gervais et al.'s simulator diverge from the real-world data set the most (RMSE: 20,066 ms). We conclude that the network model underlying the `bns` simulator enables valid simulations of blockchain network behavior.

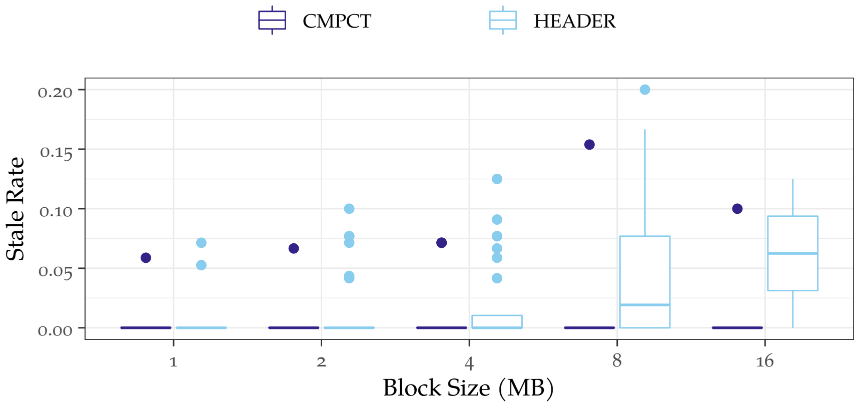


Figure 3.7: Rate of stale blocks in dependence of block size and propagation method.

3.3.3 *Impact of Network Utilization*

We furthermore investigate what impact different block sizes and propagation schemes have on the network utilization and the resulting block propagation process. To this end, we simulated the compact block and header-based propagation of 1, 2, 4, 8, and 16 MB blocks in scenarios with 500 nodes and analyzed the block propagation delay.

As to be expected, the average block propagation delay increases with the size of transferred blocks and is significantly decreased when compact block relaying is enabled. Starting from 4 MB, we observe a high network utilization for header-based relaying that increasingly leads to network congestion and in turn induces further packet losses and retransmits. Especially in edge cases and for lower-bandwidth peers, this leads to higher delays until blocks are received, which is reflected by increased standard deviations of the average block propagation delays. This behavior is of particular relevance, because when miners receive new blocks too late, they waste their mining power on producing stale blocks, which has been shown to negatively impact the security of the consensus layer [Ger+16]. Figure 3.7 therefore shows the rates of stale blocks, i. e., blocks that are finally not included in the blockchain, in dependence of the block size and propagation method. While the stale rate for header-based propagation remains negligible for 1 and 2 MB blocks, it rises for block sizes of 4 MB and above. Stemming from the high network utilization observed for 16 MB blocks, the average stale rate even surpasses 6%.

While these results show that the Bitcoin network currently could not handle header-based propagation of larger blocks without incurring security penalties, they also highlight that compact block propagation significantly improves the block propagation delay and network utilization. As shown in Figure 3.7, the simulation results indicate that Stale

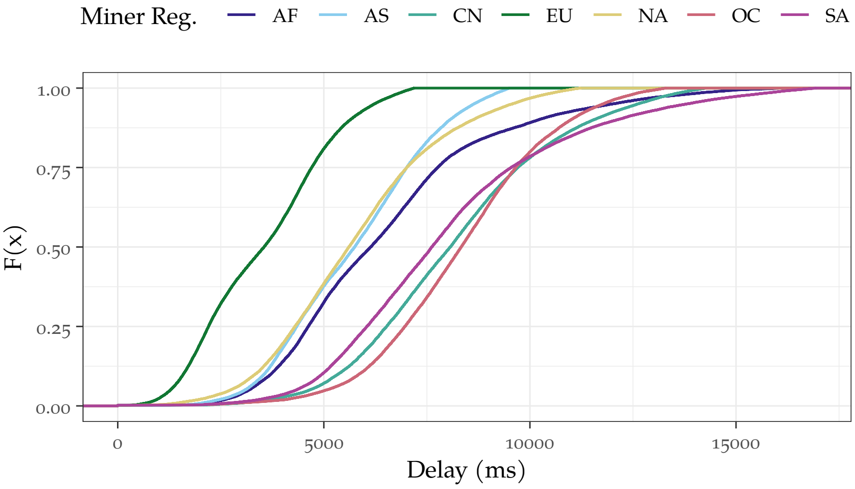


Figure 3.8: Propagation delays in dependence of miner's geographic location.

rates can in fact kept negligible for sizes of up to 16 MB and beyond, when these blocks are propagated through the compact block scheme.

3.3.4 *Regional Influence on Block Propagation*

In order to evaluate the regional influence, we configured `bns` with parameters in accordance to Bitcoin, i. e., with block size factor set to one, and simulated scenarios in which single miners were deployed in each of the regional hubs. These simulations were run in network scenarios with 500 peers and were repeated 150 times to ensure statistical significance.

Figure 3.8 shows the incurred delay in order to disseminate a new block to 90% of network peers in dependence of the miner's geographical location. Not surprisingly, miners in the EU, NA, and AS regions are able to propagate their blocks the quickest, exhibiting an average delay of 3.5 s, 5.7 s, and 5.7 s, respectively. In comparison, miners located in the AF, SA, and CN regions take more than 84% longer than EU miners to propagate their blocks in the network. Miners located in the OC region take the longest to propagate their blocks, resulting in an average block propagation delay of 8.5 s, 133% longer than miners in the EU region.

These results clearly show that the provisioned bandwidth and geographical location have a significant impact on miners' block propagation times. As such specific network characteristics may only be simulated based on a fine-grained network model, this scenario highlights the capabilities of the `bns` framework.

4

MAPPING THE ZCASH PEER-TO-PEER NETWORK

This chapter is based on the publication: Erik Daniel, Elias Rohrer, and Florian Tschorsch. "Map-Z: Exposing the Zcash Network in Times of Transition." In: LCN '19: Proceedings of the 44th IEEE International Conference on Local Computer Networks. Oct. 2019 [DRT19]. While this was a collaborative effort, the implementation and execution of the measurement study and topology inference should be attributed to fellow co-authors.

The Zcash [@Ele21] project is a privacy-focused cryptocurrency that implements the Zerocash [Ben+14a] protocol, which enables transaction anonymity on the consensus layer through the application of zero-knowledge proofs [Ben+14b]. While the network is online since October 2016, no prior work explored the conditions found in the Zcash peer-to-peer network. To this end, we present in the following the first longitunidal measurement study on the peer-to-peer networking layer of the Zcash cryptocurrency. Moreover, we study whether the topology of the network can be inferred based on a passive timing analysis of network message exchanges.

4.1 ZCASH MEASUREMENT STUDY

The goal of this investigation is to illuminate network characteristics on a global scale. To this end, the empirical data covers two main categories: data about individual nodes and about information propagation between nodes. In particular, we show the worldwide interest in Zcash and present insights on network size and stability, geographic node distribution, software deployment and lifecycles, origin of blocks, block propagation time, as well as mining centralization.

4.1.1 *Measurement Setup*

In order to characterize the Zcash network, we deployed a Zcash reference client for data collection as an observation point. The client was run on a virtual machine in our university network (TU Berlin, IPv4). However, minor client modifications were necessary to ensure a reliable measurement setup that avoids unwanted side-effects and yields representative results. For example, in order to get a comprehensive overview, it is necessary that the observation point is connected to as many peers as possible. We therefore increased the maximum number of outbound connections and allowed for 873 simultaneous connections, which is the upper limit provided by the reference client on a Linux machine. Moreover, we disabled the sending of block inventory messages that announce new blocks to peers. We however recorded arrival times of block inventory messages from neighboring peers to get an overview of the block propagation times in the network. Additionally, we recorded information about connected peers and the mining difficulty. In our measurements, we omitted all data points recorded during the initial blockchain download, because this is a one-time phase and therefore

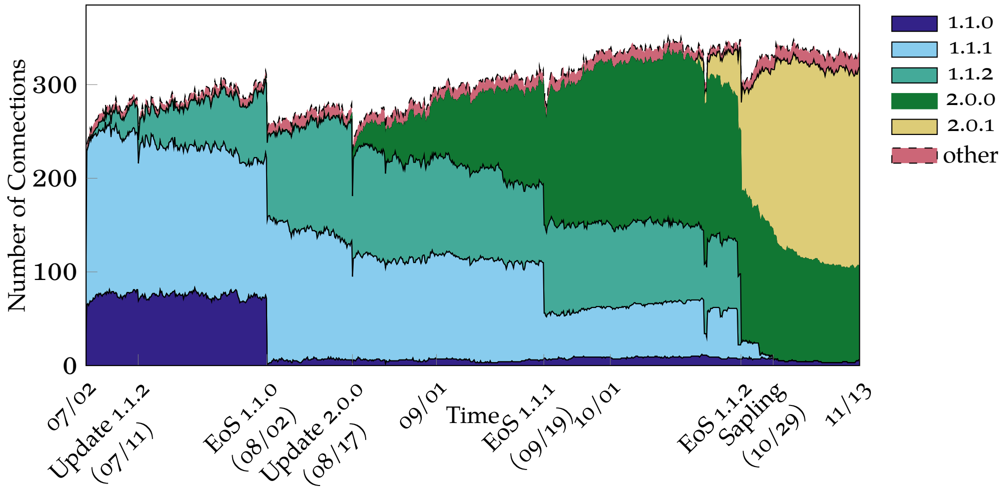


Figure 4.1: Distribution of announced version strings in the Zcash peer-to-peer network.

does not represent normal operation. The described methodology is on par with other measurement studies [Neu19].

We recorded data snapshots about peers and blocks every five minutes and two minutes, respectively. While a time interval of two minutes does not guarantee to record every single block, it should capture the vast majority of blocks[1]. In addition to the application layer data, we sent ICMP ping messages to the connected peers, which serve as a baseline and help to reveal networking problems during the measurements.

Our measurement period spans from July 2, 2018 until November 12, 2018. It thus captures the time after Zcash's first network upgrade called "Overwinter" [@Ele19]. During the measurements, we upgraded the client twice to include new functionalities and to ensure compatibility after Zcash's second network upgrade called "Sapling" [@Ele19]. Accordingly, client versions MagicBean 1.1.1, 1.1.2, and 2.0.0 were utilized.

4.1.2 *Measuring Network Size and Client Versions*

To begin with, we are interested in the network size and the number of nodes running a specific software version. In order to estimate these numbers, we count the number of network peers that were connected to our measurement node over time.

The number of simultaneous connections held by our vantage point and the distribution of client versions are shown in Figure 4.1. As the number of observed simultaneous connections never exceeds this range, we estimate the size of the network to be around 300 to 350 nodes. Note however that we observed 4,208 distinct IP addresses which established a connection to our vantage point at least once. In total, the address

---

[1] At the time of writing, Zcash's block interval time is 2.5 minutes

manager of our node learned from exchanged address information about around 25,000 IP addresses. We assume that this high number of (stale) IP addresses stems from a small number of network peers that constantly change their IP address.

As a side note, on November 11, we observed 117 simultaneous connections coming from the same IP address. These 117 connections were open for approximately 30 minutes but did not send any data. The version name of this client was "xbadprobe", which suggests that this may have been an attempt to occupy the node's inbound connections.

The reference client provided by the Zcash developer team uses the string "MagicBean" followed by a version number as the client name. While it is possible to modify the client software without changing the announced version string, we consider this a negligible side-effect and use the strings to account for different client versions. Likewise, the data shows that only a small fraction of clients use custom version strings or minor versions. We therefore only consider major client versions and categorize minor versions (e. g., 2.0.1-rc1) and custom version strings as "other clients" in the following.

Since client version 1.0.9, each reference client comes with an End-of-Support (EoS) period, which causes the client software to halt after the EoS period has expired. At the time of writing, the period is set to be 64,512 blocks or approximately 16 weeks. Visually, we can clearly identify the EoS periods in Figure 4.1. For example, we captured the complete 16 week lifecycle for client version 1.1.2, showing the adoption of and transition to other versions. Moreover, client versions older than 1.1.0 are no longer supported after the second network upgrade. Connections to these outdated clients are therefore dropped after the version handshake. Nevertheless, our data reveals that some clients were still active with older client versions, even though they no longer supported the consensus. However, the client versions 1.1.0 and 1.1.1 included a configuration option which disabled the automatic halt after the EoS period, which explains why the observed drop in numbers for those versions is not as entire as for version 1.1.2. The introduction of EoS periods is an interesting and clearly effective approach to ensure that all users update their client software and obsolete versions leave the network in a timely manner. This reduces the threat that bugs persist even after they were already fixed, which in turn bears the danger to undermine the trust in the currency.

4.1.3 *Measuring Geographical and Mining Distribution*

In Figure 4.2, we show the country distribution of observed IP addresses and block origins during our measurement period, which provides insights on Zcash's global adoption and mining power distribution. We used GeoIP [@Max21] to map the IP addresses from our data set to countries. If a country had less than 30 IP addresses it is grouped as "other". We further assume that the host which first announces the block is also the block's miner.

We generally can see that the Zcash network spreads over all northern continents. From a country perspective, the clients are mainly present in Russia and the United States. From a continent perspective, however, the network is evenly distributed over Europe, America, and Asia. Only a small number of nodes are located in the southern continents.

In contrast, the mining power distribution clearly shows a skewed distribution, suggesting an immense centralization of mining power. We observe that around 51 % of the blocks are created by 16 miners, out of which ten are located in China. Overall, 53 % of all observed blocks originate from China, 10 % from France, 9 % from the United States, 5 % from Germany, and 4 % from the Netherlands. The remaining 19 % are from different countries. Notably, while we observed only seven IP addresses from Ireland, the country contributes 3% of all blocks. It should also be mentioned that we found some nodes that did not announce any blocks.

We also observed a quadruplication in the difficulty during our measurements, resulting from an increased mining power, which effectively raises the bar to start mining blocks in the Zcash network. This increase in mining power hence makes it unlikely that the country distribution of miners will change in the near future. As new Equihash-capable ASIC mining hardware was introduced in mid 2018, the increase in mining power is likely a result of a gradual change from GPU mining to ASIC mining.

In general, geographical centralization of mining power in one jurisdictional area creates the risk of interference by a state actor. The centralization of mining power in China we observed is in line with results from other cryptocurrencies, e. g., Bitcoin [KJL18].

4.1.4 *Measuring Block Propagation*

In order to determine the time it takes for a block to propagate in the network, we measured the time it takes until a block is announced by all neighbors of our vantage point. Figure 4.3 accordingly shows the time differences as a mass function of the first and all following observed block announcements (i. e., arrival of block inventory messages). In order to estimate when the neighbor peers learned about the blocks, the shown times are adjusted by subtracting half RTT retrieved from Zcash's keep-alive messages. The RTT is estimated using the exponential weighted moving average (EWMA) approach known from TCP (cf. [Pax+11]).

We note that block propagation follows a long-tailed distribution, where a considerable number of inventory messages take significantly longer. After 690 ms, 50 % of all block inventory messages have arrived. And after two seconds, 90 % of all nodes know the block. Furthermore,

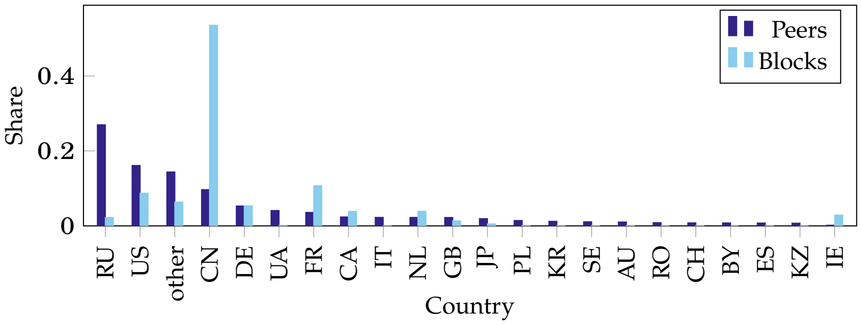


Figure 4.2: Distribution of observed addresses and block origins.

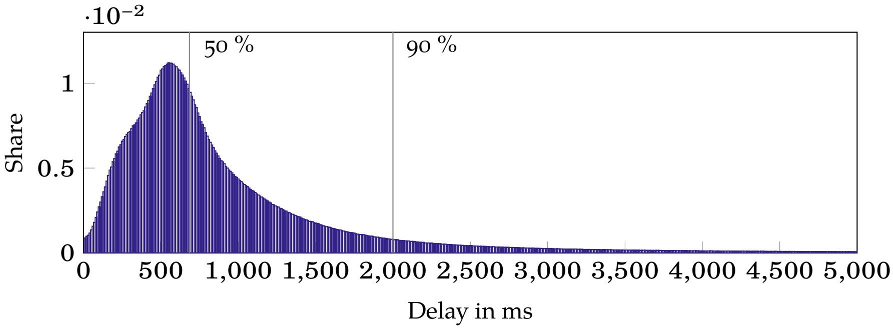


Figure 4.3: Block inventory arrival time differences.

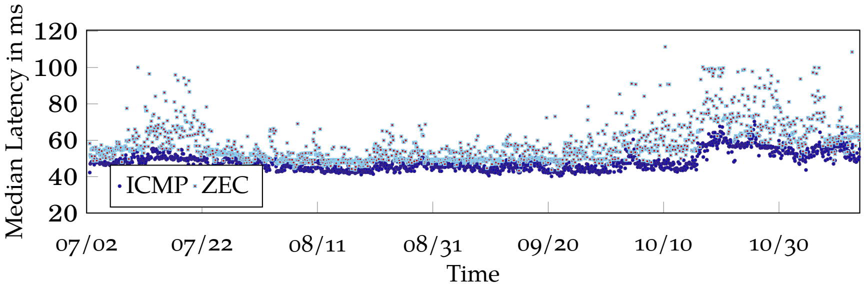


Figure 4.4: Latencies of the connected clients.

it takes a small number of nodes a really long time to retrieve and propagate new blocks. Interestingly, this is in accordance with similar observations ($\approx 2\,s$ for 90 % coverage) made in [Neu19] for the Bitcoin network, even though the latter is considerably larger (at the time of this paper: ≈ 350 vs. ≈ 10.000 nodes).

In general, we can say the network needs around 700 ms for a block to be known by most peers and roughly two seconds to spread the information in the whole network. However, this delay does not seem to have a significant negative impact on consensus. From the 57,365 observed block hashes, only 297 are not included in the Zcash blockchain, yielding a stale block rate of 0.337 %. This measurement is comparable to the stale rates found in other cryptocurrency networks like Bitcoin or Litecoin, which according to [Ger+16] exhibited around 0.41 % and 0.273 % stale blocks, respectively.

4.1.5 *Measuring Network Stability*

Lastly, we are interested in the network load and stability. Figure 4.4 shows the median latencies to all connected nodes over time. As a baseline, we additionally measured ICMP ping times and compare them to the Zcash ping measurements. The Zcash ping is a keep-alive message, scheduled every two minutes, which is send to all neighbors via the respective TCP connection and is usually processed with all other exchanged messages. This leads to head-of-line blocking during transaction and block relay, whereby the Zcash ping messages are delayed. In comparison to the more reliable ICMP ping, it therefore rather serves as an indicator of a node's activity.

The ICMP ping's median values vary between 45 ms and 55 ms and are lower and less fluctuating than Zcash pings. From October 18, the median ICMP ping and Zcash ping latencies increased. This behavior could be a reaction to the second upgrade ("Sapling"). In this upgrade, the performance of the so-called shielded transactions was improved, which might have made them more attractive and could have lead to increased usage. The increase of the ICMP ping could also be due to network interferences in the University network.

As in any other open peer-to-peer network, nodes in the Zcash network can join and leave at any time, which results in an ever-changing topology. In order to assess the network stability, we analyzed the lifetime of connections. In order to circumvent artificial spikes in our data set stemming from our own client updates, we only consider the time after we updated our measurement node to client version 2.0.0.

We generally observed that around 50 % of connections remained active for at least 50 minutes. Moreover, around 20 % of connections lasted longer than a day, while 10 % were active for more than four days, and 1 % have even remained active since the beginning of our study. We also observed around 24 % of short-lived connections, which were active less than five minutes. In summary, we could see that a good part of the network consists of long-lived stable peers. This *core network* can certainly become a reasonable target for topology inference.

4.2 TOPOLOGY INFERENCE

In the following, we aim to infer the interconnectivity of Zcash network peers by conducting a passive timing analysis, which allows us to incrementally uncover the topology of the peer-to-peer network. To this end, we build upon and extend the inference model developed by Neudecker et al. [NAH16; NH19]. We however replace the employed active transaction measurements with passive block measurements, which, as we will see, has a number of advantages. In a nutshell, the goal is to infer the connection between two peers by monitoring when they emit inventory messages announcing new blocks. The model concentrates on detecting connections between peers adjacent to mining nodes, which can be used to reveal security-critical information of the network.

4.2.1 *Inference Model*

Let us assume a source node S, which we consider the origin of a block, and a relay node R. Moreover, assume that node S and R are both interconnected and also connected to a measurement node M.

A schematic of this three-node scenario is shown in Figure 4.5. While numbered arrows indicate the order and direction in which messages are sent, dashed arrows indicate further protocol messages that are however only shown for completeness, but do not influence the inference model. Hence, a block relay in Zcash consists of a three-way message exchange (announce, request, response): An `inventory` message announces a block, a `getdata` message requests the block, and a `block` message transmits the actual block data.

From the perspective of M, we have knowledge about the time node S and R announced a specific block to node M. We also know the round-trip times RTT_{MS} and RTT_{MR} between nodes M and S, as well as nodes M and R, respectively. The measured arrival times consist of at least a delay introduced by the latency between S and R and the processing delay of R. Since the measurements are conducted on M, the measured time includes $RTT_{MS}/2$ as well as $RTT_{MR}/2$, which we subtract once prior to the following calculations. Note that if the latency difference is “small enough”, S and R are directly connected with a high probability.

The message exchange for four nodes (i. e., two hop inference) can be seen in Figure 4.6. In this case, $RTT_{MR_2}/2$ is subtracted from the measured time (instead of $RTT_{MR_1}/2$ only). Figure 4.7 shows inference scenarios for distances of up to three hops from the source S. While

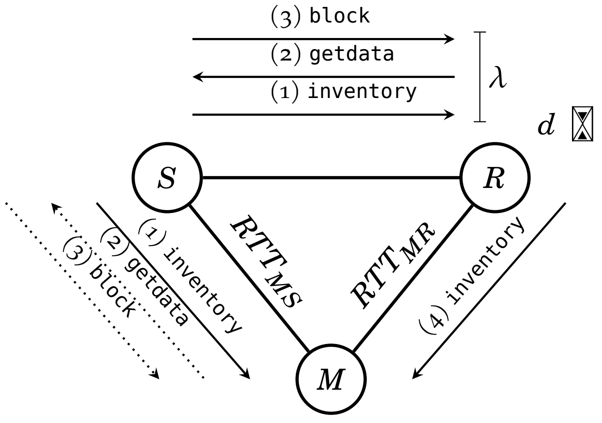


Figure 4.5: Block propagation with three nodes.

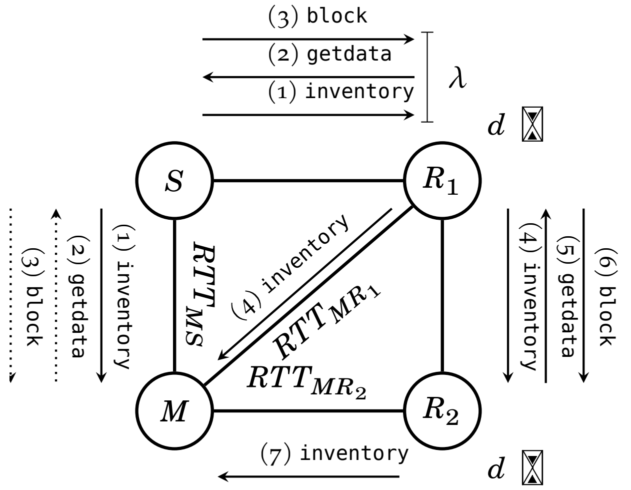


Figure 4.6: Block propagation with four nodes.

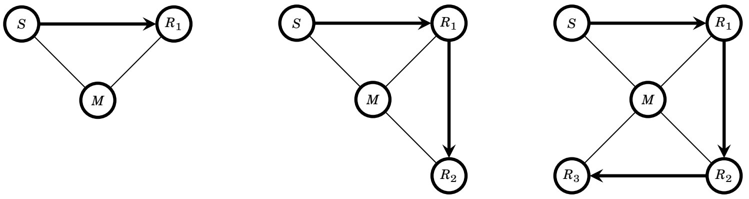


Figure 4.7: Topology inference for different hops.

not shown in the figure, all nodes can have connections to other nodes. In particular, the masurement node M would be connected to many more nodes. We take this as the basis for our timing analysis. In the following, we will derive our model in detail.

When we assume that the link latency between any two peers follows the same distribution λ and a node's processing delay can be described as d, then the probability of a time difference t with h edges in between the two reference nodes is given by

$$P(\Delta = t|H = h) = (\lambda^{*h} * d^{*h})(t), \tag{4.1}$$

which is adopted from [NH19]. Please note that the $*$-operator denotes a convolution; accordingly, λ^{*h} and d^{*h} denote the h convolution power. To infer the topology, we have to calculate the probability of h edges assuming a time difference t. This is possible using Bayes' Theorem,

$$P(H = h|\Delta = t) = \frac{P(\Delta = t|H = h) \cdot P(H = h)}{P(\Delta = t)}. \tag{4.2}$$

The probability that an inventory message arrives after time t, $P(\Delta = t)$, can be calculated according to the law of total probability. The probability $P(\Delta = t|H = h)$ is given by Equation (4.1). The probability of h edges between the two reference nodes, $P(H = h)$, can be calculated by assuming an Erdős-Rényi random graph model, where the probability of an edge is calculated based on the mean degree. Hence, given a mean degree deg and N nodes, the probability $P(H = h)$ can be derived as

$$P(H = h) = \left[1 - \left(\frac{deg}{N-1}\right)\right]^{h-1} \cdot \left(\frac{deg}{N-1}\right), \tag{4.3}$$

where we take the probability of not being connected to the power of $h - 1$ multiplied by the probability of being connected.

For our model, we assume that the link latency distribution λ is a result of the three-way message exchange to relay a block. Furthermore, we assume that this latency distribution follows a normal distribution, i. e., $\lambda(x) = \mathcal{N}(x; \mu_\lambda, \sigma_\lambda^2)$. The expected propagation time is presumed to depend on the geolocation of S and R. While this is a very simplifying assumption as latencies may depend on many factors, including AS and peering relationships, country-based measurements are already available and/or easier to obtain in decent quality. The mean μ_λ and variance σ_λ^2 values of the normal distribution can accordingly be estimated by using RTT measurements for the respective geolocations, multiplied by a factor of 1.5 to mind three-way message exchange.

We define a node's processing delay d as the sum of transmission delay, queuing delay, and block verification time. While the transmission delay can have a significant impact for very large block sizes, we assume the block verification time to be the dominating factor. We therefore assume the network-based delays to be adequately captured by the link latency and leave the development of an advanced transmission model as an open question for future research. Furthermore, we generally assume a linear correlation between the time it takes to validate a new block and the number of included transactions, i. e., the block size s_b. However, since the network peers run on equipment of varying power, the appropriate validation factor is not necessarily a global constant. To account for variations, we model the processing delay d as a normal distribution, i. e., $d(x) = \mathcal{N}(x; \mu_d, \sigma_d^2)$. The resulting function is

$$P(t - \epsilon \leq t \leq t + \epsilon|H = h) = \int_{t-\epsilon}^{t+\epsilon} \mathcal{N}(t; \mu, \sigma^2)dt \tag{4.4}$$

with $\mu = h \cdot (\mu_\lambda + \mu_d)$, $\sigma^2 = h \cdot (\sigma_\lambda^2 + \sigma_d^2)$. Here ϵ is a tolerance variable adjusting for the possibility of measurement errors. Given the complexity of this model, the selection of reasonable mean and variance values is important to reach an adequate degree of accuracy.

4.2.2 *Parametrization*

The inference model requires a number of parameters which have an impact on the accuracy of the resulting estimations. These parameters include the normal distributions for the latency λ and the processing delay d, as well as the value for the tolerance variable ϵ.

For the latency distribution λ, different data sources are possible. We consider the iPlane dataset [@iPl19], which provides publicly available data of global latency measurements, as a viable data source. Admittedly, the data is somewhat outdated but still fits the purpose. It consist of pairs of globally distributed IP addresses and a corresponding RTT value measured on a specific point in time. As an alternative data source, we suggest to conduct ICMP ping measurements from the observation points or to directly utilize the Zcash ping measurements (cf. Figure 4.4).

Values for the validation time can also be acquired through different means. We consider the evaluation constant from [Ger+16] as a reasonable estimation for μ_d. Unfortunately, in this case we must make some assumptions on the variance. Alternatively, μ_d and σ_d^2 can be acquired experimentally, i. e., by averaging the processing time of a local Zcash node. We compare both approaches in our evaluation.

Varying the tolerance parameter ϵ should generally have no significant influence on the measurement results. However, larger values for ϵ increase the influence of the likelihood $P(t - \epsilon \leq t \leq t + \epsilon | H = h)$ and therefore decrease the influence of the prior probability $P(H = h)$ on the posterior probability.

4.3 EVALUATION

We evaluate our inference model using two different ways: through a simulation scenario and a real world measurement test with two Zcash nodes.

4.3.1 *Methodology*

In order to facilitate the simulation-based evaluation scenario, we created an undirected graph with a certain amount of vertices utilizing the Boost Graph Library [@Jer19]. Every vertex represents a Zcash networking node and creates 8 edges to randomly selected vertices, representing the 8 outgoing connections the reference client tries to establish. The mean degree of the random topology is therefore 16. We assign a country to each vertex and draw an according link latency value for each edge from a normal distribution that was parametrized with the iPlane data set [@iPl19]. As discussed before, we multiply this latency value by factor 1.5 to mind the entire three-way message exchange. Moreover, we simulate our vantage point which is connected to all other vertices and therefore able to observe the simulated propagation behavior in its entirety.

Given this network graph, we can deduce simulated time difference measurements by traversing the shortest path between any node and our vantage point, accumulating the edge weights accordingly. In order to retrieve statistically significant results, we repeated this process 50 times for varying edge weights and applied our topology inference model for each of the measurements. Each time, we calculated the probability under a tolerance of $\epsilon = 5\,ms$ for possible hop counts h of 1 to 9 and finally calculated the mean value for each distance. From this, we deem the value with highest mean probability as the likeliest estimated distance. As we are in full knowledge of the simulated topology, this scenario allows us to exactly determine the precision and recall of our model under the influence of different parametrizations for the processing time distribution $d = \mathcal{N}(x; \mu_d, \sigma_d^2)$ and block sizes s_b. The validation constants k_μ and k_{σ^2} determine the distribution parameters as $\mu_d = k_\mu \cdot s_b$ and $\sigma_d^2 = k_{\sigma^2} \cdot s_b$.

Theoretically, our model could predict the probability of h up to the network diameter. However, our model becomes less accurate for distances above three, since path lengths larger than two exhibit an increased possibility for parallel running paths. Therefore, our model is suited best to infer individual connections one by one. Additionally, as our model utilizes assumptions about the geographic locations of network peers to determine their edge latencies, inaccuracies in the geographic clustering can result in high amount of false positives for certain peer connections. For example, if we assume a scenario in which two nodes located in Germany are connected via a third node located in Russia, our model could predict a distance of five (low latency) edges, even though the simulated path consists only of three (high latency) links.

In our simulation, we considered topologies with 300 nodes, which resembles the measured network size. Furthermore, we evaluate our model in the simulation for distances up to three. For the geographic distribution, we chose two countries each from America and Europe, and three countries from Asia, whereby we aim to represent the northern Hemisphere. The distribution assigned as follows: 30 % United States, 20 % in Russia, 10 % Canada, 10 % China, 10 % France, 10 % Germany, and 10 % Japan.

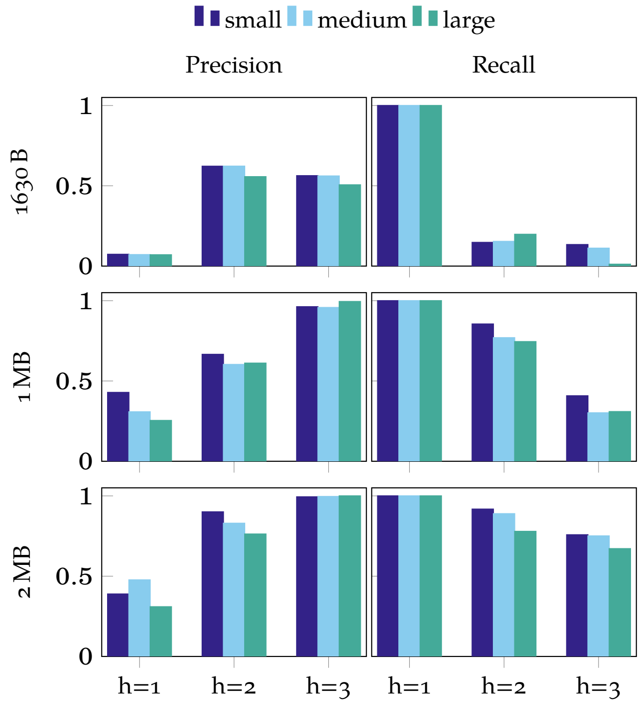


Figure 4.8: Precision and Recall for different block sizes, different variances and 300 nodes located in 7 different countries.

4.3.2 *Simulation Results*

We use a simulation scenario to evaluate different parametrizations of the processing time distribution d. That is, we evaluated precision and recall of our model for different block sizes s_b assuming the validation constants $k_\mu = 0.3796\,\mu s/B$ and $k_{\sigma^2} = 0.552049\,\mu s^2/B$ taken from [Ger+16]. We also ran the simulations for three different degrees of latency standard deviation: "small", "medium" and "large", assuming 10 %, 30 %, and 50 % of the mean, respectively.

The simulation results presented in Figure 4.8 show that the model generally performs better with larger block sizes, which yield a precision of up to 40 % and recall of up to 100 %. Surprisingly, the precision for a direct connection is slightly higher for a 1 MB block than for a 2 MB block. However, the validation time seems to be a good indicator for the number of hops, since the precision for distances two and three are above 50 % Moreover, the larger the block sizes, the higher are the estimated processing times. Hence, the lower is the share of the estimated latency, which makes the model less dependent on accurate latency estimations.

Table 4.1: Real-world measurement results.

	Model of [Ger+16]	**Test net.**	**Main net.**
k_μ	$0.38\,\mu s/B$	$8.55\,\mu s/B$	$12.74\,\mu s/B$
k_{σ^2}	$0.55\,\mu s^2/B$	$345.1\,\mu s^2/B$	$2128.16\,\mu s^2/B$
True Positive	18	30	33
False Positive	18	26	33
False Negative	22	10	7
Precision	50 %	53.5 %	50 %
Recall	45 %	75 %	82.5 %

4.3.3 *Real-World Experiments*

In the real-world evaluation scenario, we deployed two nodes in the peer-to-peer network over the course of one week. One node functioned as the measurement node and recorded the arrival time of all block inventory messages. Moreover, as a point of reference, the relay node recorded its connections over the time of measurement.

Afterwards, we apply our model to infer if a direct connection between the different block creators and the relay node existed in the given time frame. We calculated RTT estimations using the EWMA approach.

In the measurement period, we estimate the network size to be 316 nodes of which 87 nodes sent at least five blocks. Overall, we recorded 4,160 blocks with an average block size of $15,678\,B$ and 21 stale blocks. We also recorded the average validation times our client exhibited during the verification of 5,000 blocks from the Zcash main network. Resulting from this, we set the estimated validation constants to $k_\mu = 12.7357\,\mu s/B$ and $k_{\sigma^2} = 2128.16\,\mu s^2/B$.

For the further evaluation, we only consider direct connections with the (mining) nodes that announced at least five blocks.

As shown in Table 4.1, even despite the low block sizes, our model is able to achieve a precision of 50 % and a recall of 82.5 % under the discussed parametrization.

4.3.4 *Discussion*

This real-world evaluation scenario shows that we can consider half of the inferred connections as correct and only a small amount of direct connections as missing. The results support the general validity of our inference model. Considering the deliberate simplifying assumptions, the approach seems to be a promising step for inferring the topology of many cryptocurrencies similar to Bitcoin. Furthermore, countermeasures for our method—e. g., artificially delaying block relays—would increase the probability of stale blocks, which weakens the system. However, it may be possible to limit the reliability of the inference model by producing particularly small blocks yielding a decreased processing delay. As this also impairs the functionality of the Zcash network, it becomes clear that our method will remain usable in the future.

Moreover, it should be possible to improve the precision of the current model by acquiring more diversified latency measurements and considering additional information about the client location, like the AS number. For higher block sizes, a better modeling of the validation time and the transmission delay could also improve the results. Furthermore, it is known that blocks of the same size can have different validation time depending on the complexity, e. g., number of shielded transactions vs. number of transparent transactions.

Furthermore, the calculation of the probability of a certain amount of edges between nodes ($P(H = h)$) is calculated based on an Erdős-Rényi model, which is a random topology. However, it is unlikely that nodes are connected randomly in a peer-to-peer network in which certain nodes have a higher uptime than others. It is more likely that some central nodes have more than 16 connections, especially nodes of the core network. We leave these ideas open for future research.

5

KADCAST: A STRUCTURED BROADCAST PROTOCOL FOR BLOCKCHAIN NETWORKS

This chapter is based on the publications:
Elias Rohrer and Florian Tschorsch. "Kadcast: A Structured Approach to Broadcast in Blockchain Networks." In: AFT '19: Proceedings of the first ACM conference on Advances in Financial Technologies. Oct. 2019 [*RT19*]
and
Elias Rohrer and Florian Tschorsch. Kadcast-NG: A Structured Broadcast Protocol for Blockchain Networks. Submitted and currently under peer review. 2021 [*RT21b*]

As we have seen in prior chapters, current implementations of blockchain peer-to-peer networks typically rely on an unstructured overlay construction for block and transaction dissemination. While this approach is relatively robust, it however still may not be favorable for these kinds of operations. For instance, broadcast in unstructured overlays potentially suffers from a high messaging overhead, as duplicates are introduced to the system. To reduce the load, many networks have to reconcile these drawbacks by gossiping messages only to a subset of neighbors or by introducing advanced messaging patterns (cf. Section 2.2), both techniques which in the worst case might introduce additional propagation delays.

We however argue that many issues can be efficiently resolved by rethinking the networking architecture from first principles. In particular, the introduction of a structured overlay topology promises to address many of the shortcomings of the current networking layer. To this end, we present *Kadcast*, a new networking protocol for the strucutured dissemination of information in blockchain peer-to-peer networks.

5.1 THE KADCAST PROTOCOL

Kadcast is based on Kademlia [MM02], a DHT design that is typically used for efficient lookup procedures. Kadcast, however, makes use of Kademlia's overlay structure to enable an efficient broadcast operation. In the following, we describe the overlay construction and the broadcast algorithm as the two main building blocks of our approach. Moreover, we introduce means to improve the performance, reliability, and resilience of Kadcast.

5.1.1 *Overlay Construction*

Kademlia is a UDP-based peer-to-peer protocol in which nodes form a structured overlay network. Nodes in the network are addressed by unique L-bit binary node identifiers, in the following denoted as `ID`, which are generated upon joining the network. The `ID` determines a node's position in a binary routing tree that builds the foundation of Kademlia's overlay. An example of such a tree for a 4-bit address space is shown in Figure 5.1. Please note that this tree is never actually constructed and serves as a mental model only. Peers, however, still use their local state to traverse the network structure efficiently, yield-

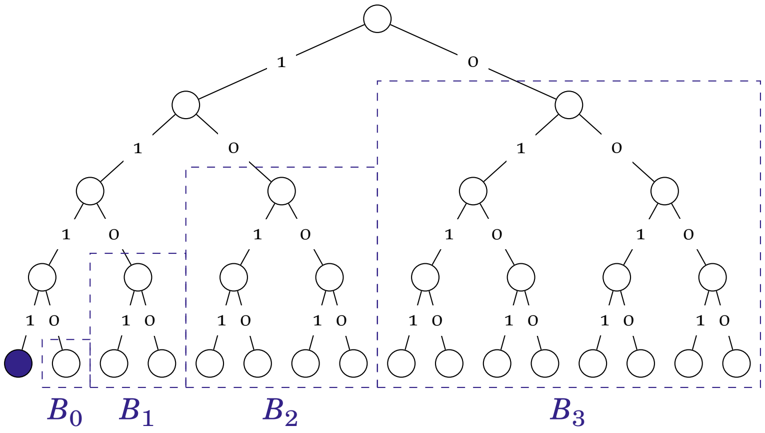


Figure 5.1: Fully populated overlay structure for a 4-bit address space and space covered by the buckets of node `1111`.

ing a message complexity of $\mathcal{O}(\log N)$. To this end, nodes maintain routing state and organize known nodes in so-called *k-buckets*, storing triplets `(ip_addr, port, ID)`. Each bucket is a list of the k least recently seen nodes that have a certain *distance*, in relation to the node identifier `ID`. The factor k is a system-wide parameter which determines the routing state and the lookup complexity.

Characteristically, Kademlia's notion of distance is based on the non-euclidean XOR-metric, calculated by applying the $\oplus$-operation on two node identifiers and interpreting the result as an integer number, i. e.,

$$d(x,y) = (x \oplus y)_{10}.$$

This means, that for node identifiers of length L, a node $\texttt{ID}_0$ holds buckets B_i, $i = 0 \ldots L-1$, whereby bucket B_i holds the node information of k nodes with $\texttt{ID}_j$ so that $2^i \leq d(\texttt{ID}_0, \texttt{ID}_j) < 2^{i+1}$. It follows that the node space covered by each bucket is exponential with i. This can be illustrated by that fact that, since the XOR-metric is unidirectional, the bucket B_0 only holds one specific node of distance one, while B_{L-1} covers a possible node space of 2^{L-1} nodes. The buckets can be thought of holding up to k nodes belonging to a series of subtrees with identifiers whose binary prefixes do not match the nodes' prefix, i. e., also not containing the node itself. For example, given the fully populated tree shown in Figure 5.1, the 4 buckets of node $\texttt{ID}_0 = \texttt{1111}$ would hold nodes from the ranges `1110`, `110*`, `10**`, and `0***`, respectively. If a node wants to add a new entry to a given bucket that already holds k entries, it employs a least recently used (LRU) drop policy. Before dropping an entry from the list, the peer will send a `PING` message (see Figure 5.2) to see whether the respective node is still reachable. Only if the node is not reachable anymore, it will be dropped. This way, the protocol favors older, more stable nodes over fresh ones. It thereby also circumvents an eviction bias towards fresh, potentially malicious peers, which hardens

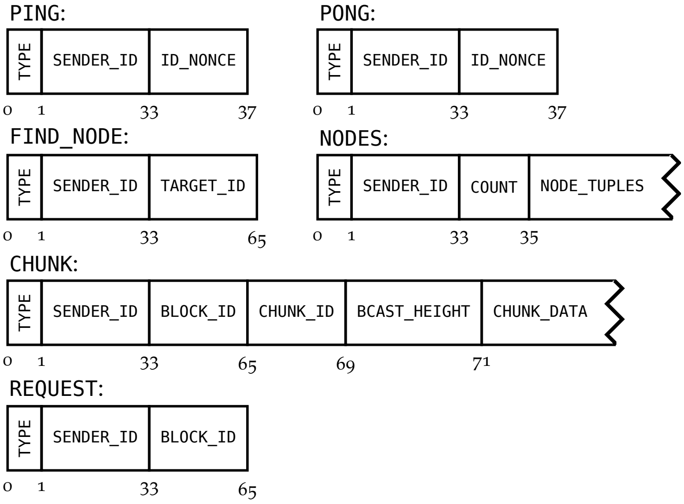


Figure 5.2: Kadcast message types.

the network against security issues, such as the eclipse attacks described in [Hei+15].

When a node first joins the network, it has to know the address of at least one bootstrapping node. It therefore sends `PING` messages to known nodes to check whether they are actually online. Additionally, `PING` transmits the sending node's routing information to the recipient, thereby distributing its existence in the network. In fact, similar patterns can be found throughout the protocol, where every seen message updates not only the sender's but also the recipient's buckets. This soft-state protocol design allows for a very lightweight overlay membership management that keeps the footprint of the required information base to a minimum.

After the initial bootstrapping step, each Kadcast node begins discovering the network to update its routing information, which it repeats periodically throughout its lifetime. Initially, the joining node looks up its own `ID`, which returns a set of nodes closely positioned to its own network location. Moreover, each node periodically refreshes every bucket it has not seen some activity from in the last hour: for each such bucket, it picks a random `ID` with appropriate distance and performs a look up to populate its buckets with fresh routing information.

The lookup procedure allows a node to retrieve a set of k nodes closest to a specific `ID` in the address space. The procedure of finding the k closest nodes is carried out by iteratively narrowing down the search space and issuing `FIND_NODE` messages (see Figure 5.2) to nodes which are closer to the `ID`. To this end, (1) the node looks up the α closest nodes regarding the XOR-metric in its own buckets. (2) It queries these α nodes for the `ID` by sending `FIND_NODE` messages. (3) The queried nodes respond with a set of k nodes they believe to be closest to `ID`. (4) Based on the acquired information, the node builds a new set of closest nodes

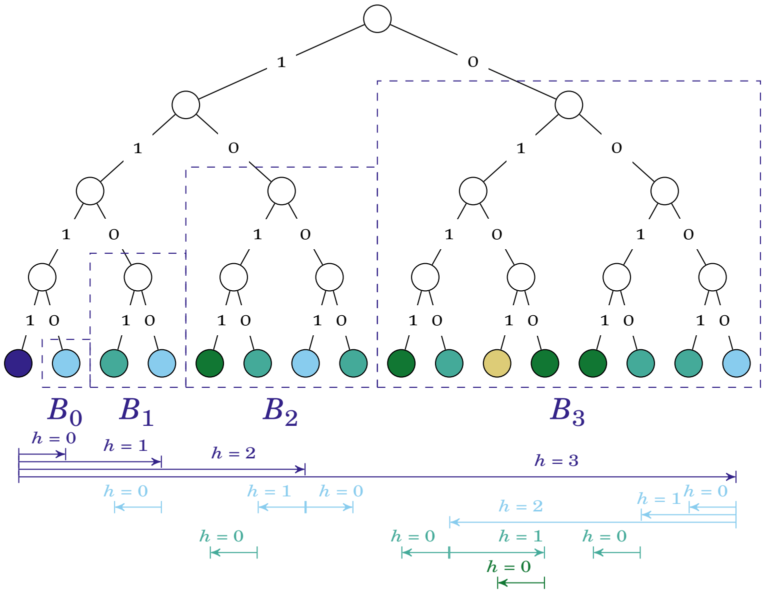


Figure 5.3: Example broadcast initiated by node 1111 ($\beta = 1$). Colors indicate node distances in the spanning tree, relative to the initiator.

and iteratively repeats steps (1)-(3), until an iteration does not yield any nodes closer than the already known ones anymore.

Like the bucket size k, α is a globally known parameter determining the redundancy (and hence also the overhead) of the lookup procedure. At the same time, these parameters influence the lookup latency, as the parallel nature of the lookup procedure optimizes the needed delay. Typical parameter values are $k \in [20, 100]$ and $\alpha = 3$. As the Kadcast protocol is not used to store and retrieve values, it does not incorporate other message types found in Kademlia.

5.1.2 *Message Propagation*

As described before, most blockchain networks rely on TCP-based transport protocols for block and transaction propagation, which ensure the reliable transmission of arbitrarily large data by retransmitting missing segments in case of packet loss. This method implicitly assumes long-lived connections and requires additional state-keeping in terms of connection management and is therefore less scalable. Retransmissions and head-of-line blocking might introduce additional delays and unpredictable message overhead. In contrast, transport protocols such as UDP or QUIC [Lan+17] enable a more scalable and dynamic approach with short-lived, low-cost transmissions, which complement Kadcast's design. While this allows for a lightweight protocol design with reduced state-keeping and tunable per-link message complexity, it also entails handling data serialization and reliable transmission on the application layer.

Algorithm 1 Redundant broadcasting algorithm.

```
    broadcast height $h$,
chunk data $c$,
set of known chunks $C$,
redundancy factor $\beta$
function BROADCAST_CHUNK($h, c, C, \beta$)
    if $c \in C$ then abort
    $C \leftarrow C \cup \{c\}$
    for $i = 0 \rightarrow h - 1$ do                      ▷ For all buckets up to height $h - 1$
        $R \leftarrow$ randomly_select ($\beta, B_i$)      ▷ Choose $\beta$ random bucket entries
        for all $r \in R$ do
            send_chunk($r, c, i$)                          ▷ Send chunk $c$ to peer $r$
        end for
    end for
end function
```

Therefore, when the propagation of a block or a transaction is initiated, Kadcast first segments its data in packet-sized *chunks* that are then distributed in the network via corresponding messages (see Figure 5.2) and according to the broadcast procedure (cf. Algorithm 1), which is a modified version of the algorithm in [CH13].

Kademlia's bucket logic partitions the identifier space in subtrees whose sizes depend on their distance to the current node. The Kadcast protocol makes use of this fact to generate a spanning tree that allows for an efficient broadcast operation: the algorithm delegates broadcast responsibilities for subtrees with decreasing height h to other nodes, which recursively repeat the process within their delegated area. Therefore, when a miner initiates the broadcast, it is responsible for the entire tree with height $h = L$. The miner picks a random peer from each bucket and delegates broadcast responsibilities by sending `CHUNK` messages, which carry the data and her routing information. It assigns a new height h, which effectively determines the receiver's broadcast responsibility. When a node receives a `CHUNK`, it repeats the process in a store-and-forward manner: it buffers the data, picks a random node from its buckets up to (but not including) height h, and forwards the `CHUNK` with a smaller value for h accordingly.

This means, with every step, another set of nodes is designated to be responsible for chunk delivery in their respective subtrees. A simple example for $L = 4$ can be seen in Figure 5.3: node $\mathtt{ID}_0 = \mathtt{1111}$ initiates a broadcast in the network, and sends four `CHUNK` messages with heights $h = 0 \ldots 3$ to one random node picked from each of the respective buckets $B_i,\ i = 0 \ldots 3$. The receiving nodes repeat this procedure, again issuing messages to nodes from bucket numbers less then their assigned height. Hence, the broadcast operation is performed on decreasing subtree sizes, and therefore guaranteed to terminate in $\mathcal{O}(\log n)$ steps. Upon receipt of all chunks required to rebuild a block or transaction message, the node follows Bitcoin's typical verification procedure before continuing the broadcast operation.

5.1.3 *Reliability of UDP-based Message Delivery*

If we assume constant transmission times, honest network participants, and no packet loss in the underlying network, the propagation method just discussed would result in an optimal broadcast tree. In this scenario, every node receives the required data exactly once and hence no duplicate messages would be introduced by this broadcasting operation. Unfortunately, we cannot make these assumptions and have to consider packet losses, as well as adversarial and random failures during transmission.

In the example of Figure 5.3, if a chunk on its way to node `0000` is corrupted or this node refuses to forward a chunk, the whole bucket B_3, i. e., the right half of the tree, would not receive the corresponding message. That is, in the worst case, a single transmission failure caused by the unreliable UDP transport protocol could result in a network coverage of fifty percent only. Therefore, the broadcast algorithm is improved and secured by two different approaches, which both introduce redundancy.

First, instead of having a single delegate per bucket, we select β delegates. This severely increases the probability that at least one out of the multiple selected nodes is honest and reachable. It therefore protects the broadcasting operation against random and adversarial node failures on the propagation path. Moreover, this parallelized broadcasting method improves the propagation performance in terms of latency: nodes with the best connection receive the transmitted chunk first and will proceed to propagate the chunks in the bucket. As this repeats on every hop, and Kadcast nodes ignore duplicate chunks, only the fastest routes are used for message delivery.

Secondly, Kadcast has to consider transmission failures due to corrupted and/or dropped packets on every hop of the propagation. When Kadcast is implemented on top of an unreliable transport protocol, such as UDP, it therefore needs to increase the reliability of the transmission and hence employs a forward error correction scheme based on RaptorQ [Lub+11] codes. The adoption of this scheme allows Kadcast nodes to recover transmitted block data after the reception of any s source symbols out of n encoding symbols, which are transmitted via `CHUNK` messages. As this results in more transmitted data overall, an overhead of $n - s$ additionally transmitted symbols per transmission is introduced. The FEC overhead factor can be adjusted through the parameter $f = \frac{n-s}{s}$. Utilizing FEC gives the receiver the ability to correct errors without the need for retransmissions, which lead to additional delay. We therefore optimize our protocol in terms of latency and accept an additional overhead. However, in order to allow nodes to recover from the rare case that message delivery fails entirely, and to enable the initial bootstrapping of the blockchain, the Kadcast protocol incorporates a simple `REQUEST` message (cf. Figure 5.2) that allows nodes to query others for specific blocks or transactions, and is answered by the corresponding `CHUNK` messages.

In the following, we analyze and discuss our methods for improved broadcast reliability.

5.1.4 *Analysis of Parallelized Broadcast*

Kadcast implements broadcast redundancy by parallelizing the algorithm. To this end, we introduce the system parameter β, which describes how many distinct delegates per bucket should be selected (and thus how many nodes per bucket should receive a copy of the message). This improved algorithm can be seen in Algorithm 1. Please note that for $\beta = 1$, Algorithm 1 describes the "optimal" broadcast from Section 5.1.2.

Along the lines of [CH13], we model the propagation reliability as the expected node coverage of the broadcast operation, which is based on the average probability of transmission failures. Thus, given the failure probability ϵ and $\beta = 1$, a single broadcast chunk would reach its next hop with probability $p = 1 - \epsilon$. The expected number of nodes receiving this chunk can therefore be expressed by $M = (1 + p)^L$, assuming a balanced distribution tree of height L, which is highly plausible due to the uniform random distribution of node identifiers. It follows that the ratio of covered nodes is

$$m = \frac{M}{2^L} = \left(\frac{1+p}{2}\right)^L .$$

Note however that this expression models the transmission of a *single chunk without redundancy* only. In order to express the coverage of a redundant broadcast, we need to extend this model.

Therefore, we model the parallel execution of our algorithm as the probability that at least one of the redundantly sent chunks is successfully delivered, i. e., $p_\beta = 1 - \epsilon^\beta$. Moreover, let X be a random variable expressing the number of received chunks. The probability that we receive all s chunks of a block or transaction is thus $p_b = P(X = s) = p^s$, which induces a failure probability of $\epsilon_b = 1 - p_b$. Accordingly, the probability to deliver a message with redundancy β is given by $p_{b,\beta} = 1 - \epsilon_b^\beta$. These observations yield an expected coverage ratio of

$$m_{b,\beta} = \left(\frac{1+p_{b,\beta}}{2}\right)^L .$$

Based on this model, we first analyze the transaction broadcast reliability. As transactions typically fit in a single UDP packet, redundancy is best introduced through increasing broadcast parallelization. Figure 5.4 shows the achieved network coverage in dependence of an assumed packet loss rate and the redundancy factor β. We observe that the coverage quickly drops if no redundancy is introduced ($\beta = 1$), but that,

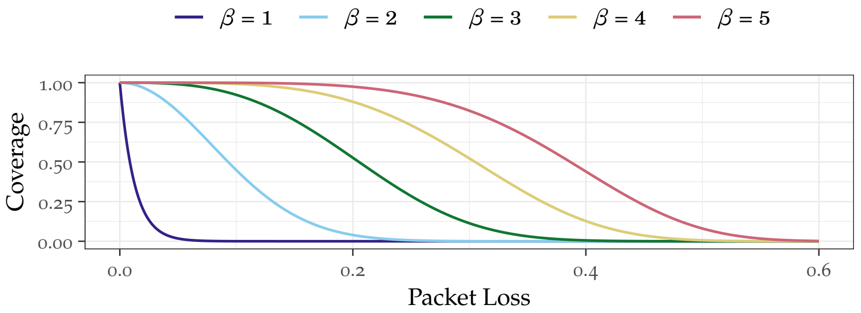


Figure 5.4: Transaction broadcast reliability $(L = 160)$.

even in the face of considerable packet losses, a reliable broadcast can be achieved for $\beta > 1$.

However, as the transmission of blocks is only successful, if all chunks are received, the propagation of entire blocks is not as reliable. To this end, we analyze the expected UDP-based block broadcast coverage in the face of different packet loss rates, $\beta \in \{1, 3\}$, and an assumed block size of 1 MB. The results are shown in the upper part of Figure 5.5: again we observe that without redundancy even the smallest packet loss makes the probability of delivering a block drop immediately, hence rendering the chance of covering the entire network virtually impossible. While a redundancy factor $\beta = 3$ has a positive impact on the block propagation, it is not sufficient on its own to guarantee the reliable transmission of entire blocks over a lossy channel. However, the parallelized broadcast is still necessary either way to compensate adversarial and random node failures, and to improve the propagation performance, as discussed before.

In order to further increase the block transmission reliability, a Kadcast node employing the RaptorQ forward error correction has to successfully receive s or more arbitrary symbols out of the n transmitted in order to recover a full block, an event which can be modeled by a binomial distribution, i. e.,

$$p_{b,f} = P(X \geq s) = 1 - P(X < s) = 1 - \sum_{i=0}^{s-1} p.$$

Figure 5.5 clearly shows the improved transmission reliability offered by introducing forward error correction with 15% redundancy $(f = 0.15)$: this approach ensures that broadcasted blocks reach full network coverage for packet loss rates up to around 9%. This is quite a number for Internet standards and is even enough to cover the large packet loss rates exhibited by connections towards mainland China.[1] However, this can still be improved by combining the FEC approach with redundancy,

---

[1] Kaiser et al. describe that, induced by the Chinese "Great Firewall", connections exhibit 6.9% packet loss, which leads to artificially delayed block propagation of Chinese Bitcoin miners [KJL18].

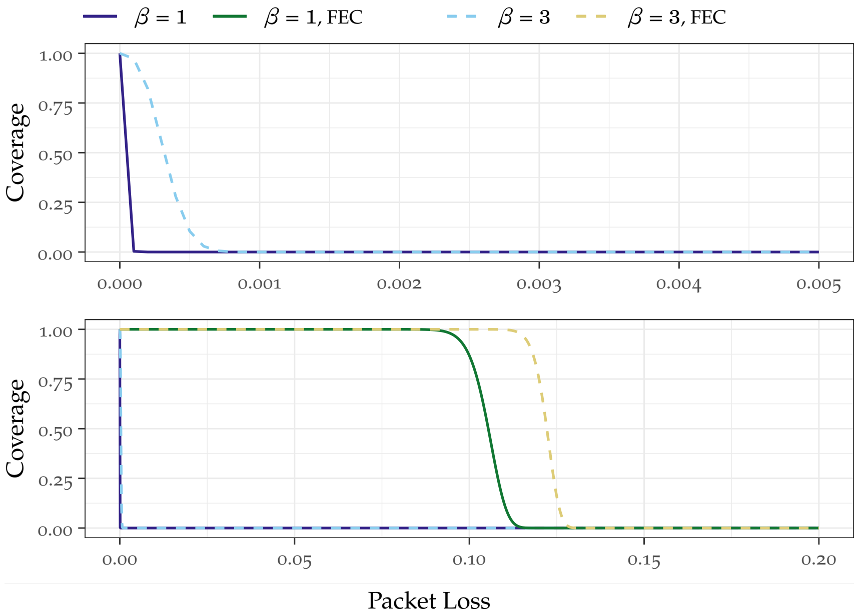


Figure 5.5: Reliability of UDP-based block propagation over unreliable channels (block size of 1 MB, $L = 160$, and FEC overhead factor $f = 0.15$).

i. e., $\beta > 1$. In this case, the success probability is $p_{b,\beta,f} = 1-(1-p_{b,f})^{\beta}$, which is shown for $\beta = 3$ in Figure 5.5 as well. The combination of FEC and parallelization ensures full network coverage, even if on average 12% packets are lost during transmission.

The analysis results highlight that FEC is a favorable way to ensure reliable transmission of data over an unreliable network infrastructure: it allows to significantly increase the reliability of the broadcast while introducing a relatively small linear overhead. In contrast, the overhead introduced with increasing the replication factor β introduces a larger increase in messaging complexity. However, broadcast redundancy is still required in cases where the weak point is not just an unreliable network link, but a malicious node obstructing block or transaction delivery.

5.2 KADCAST SECURITY AND PRIVACY

As discussed earlier, fast and fair block propagation may be considered security-critical for the consensus layer of blockchain-based systems. However, the peer-to-peer network and the block propagation mechanism may also become themselves subject to attacks on security and privacy. In the following, we therefore discuss the security properties of the Kadcast network protocol.

5.2.1 *Threat Model and Mitigation Strategies*

The Kadcast design is based on the well-studied and widely-used structured network design of Kademlia [MM02]. Numerous previous entries study Kademlia's security properties, its behavior when attacked by a range of adversaries, and designs improving on its security [BM07; CCF09; KLR09; Loc+10; SEB07; UPS11; Wan+08; Wan+13]. We now discuss the most prevalent adversarial threats to the security of blockchain peer-to-peer networks in general and Kadcast in particular.

5.2.1.1 *Sybil attacks*

THREAT: The notion of a Sybil attack [Dou02] describes the possibility of a single adversary to embody a large number of network entities by forging additional identities. By doing so, the adversary aims to outnumber the honest nodes participating in a distributed system, effectively increasing the share of malicious nodes in the system. Moreover, a Sybil attack is especially enticing when the forged identities can be used to trick the system and enable unwanted behavior. In systems based on the Kademlia overlay, Sybil attacks may be used to generate a lot of identities that can fill up a victim's buckets [KLR09; SEB07]. The ability to run this kind of attack is often a prerequisite to be able to run Eclipse attacks (see next section) on Kademlia-based systems.

In the case of Kadcast, if an adversary can forge arbitrary IDs and position herself close to a target, she may be able to increase the likelihood of receiving lookups and broadcasts from this node. This may enable the adversary to simply refuse block delivery and thereby obstruct the block propagation, which we discuss further in Section 5.2.3. Hence, we observe that the ability to create valid node identifiers at arbitrary positions in the network is detrimental to the security of the system.

COUNTERMEASURES: The Kadcast protocol employs a number of countermeasures in order to increase its resilience to Sybil attacks.

For one, the Kadcast protocol can easily be extended to incorporate cryptographic puzzles as Sybil protection, similar to [BM07; Bor06; VCS03]. Along the lines of proof-of-work mining, Kadcast follows a simple scheme: a joining node has to find a nonce, so that the hash of concatenation of its identifier and nonce adheres to a certain difficulty level. This is, the binary value of the hash has to be less than the chosen difficulty target, i. e., $H(\texttt{ID} \,\|\, \texttt{ID\_NONCE}) < t_{diff}$, where t_{diff} is a global parameter of the system. Every node that receives a new node identifier validates this property before it inserts the new node to its buckets. It can run the validation quickly, while the node generation can take quite some time, depending on the chosen parameter t_{diff}. Thereby, the inclusion of this hash puzzle scheme seriously impairs the ability of an adversary to quickly generate a large number of node identifiers.

Moreover, additional effective countermeasures encompass stricter bucket policies which enforce a certain degree of diversity from an AS-level and/or subnet perspective [AZV17; FSW14].

5.2.1.2 *Eclipse attacks*

THREAT: All peer-to-peer networks rely on some kind of routing scheme that allow nodes to decide where to forward data or which nodes to query for a specific data item. However, these routing decisions are made on the basis of an underlying data structure, the routing table. Eclipse attacks describe a family of attacks on peer-to-peer networks in which the adversary manipulates the routing tables of its targets to contain only nodes controlled by the adversary. Once she isolated her target from the rest of the network, the adversary is in full control of the data streams coming from and to the target node. This may be used by the adversary to completely block data delivery, selectively obstruct data transmission, or even foist spurious data.

In blockchain networks, Eclipse attacks are a serious threat, since they could be used to monopolize the connections of a target node and then further exploit the protocol. They have been shown to enable double-spending and selfish-mining attacks [Hei+15; MHG18]. In the past, the feasibility of Eclipse attacks on the Kademlia protocol have been studied in literature [KLR09; Loc+10; SEB07]. These studies show that, *if* an adversary would come to control a large number of node identifiers, she may try to flood all buckets of a target node with addresses of nodes in her control. This technique could be used to isolate Kadcast nodes from the rest of the network.

COUNTERMEASURES: The Kadcast protocol includes Sybil protection to hinder an adversary from attaining control of an unlimited number of nodes. Moreover, Kadcast follows a bucket eviction policy which favors older, more stable nodes over newly acquired node addresses. This policy hinders the adversary from supplying all nodes known to the victim. In order to furthermore impede the adversary's capability of foreseeing and exploiting a target node's bucket layout, we introduce temporary identifier space randomization on a per-broadcast level, which is further discussed in Section 5.2.2. In conclusion, by safeguarding the node identifiers through the means of cryptographic puzzles and applying techniques such as identifier space randomization, as well as enforcing rigorous bucket policies, Kadcast follows best practices for Sybil and Eclipse protection [MHG18].

5.2.1.3 *Denial-of-Service attacks*

THREAT: Broadcast protocols aim to distribute information to all nodes in the network. This inherent asymmetry immediately raises the question on whether they allow an adversary to flood the network with arbitrary data, i. e., how susceptible they are to denial-of-service (DoS) attacks.

COUNTERMEASURES: In order to avoid these kind of attacks, blockchains like Bitcoin employ a store-and-forward propagation policy: each block a node receives is first stored and validated (i. e., check the proof of work), before it is announced to neighbors. This way, an adversary trying to flood the network with fake block data would have to solve a proof-of-work hash puzzle for each of the forged blocks, making it a very unattractive attack vector. The Kadcast protocol adapts this DoS protection: every node first validates received blocks before they are forwarded in the broadcast tree.

Additionally, previous work [VTM14] highlighted that node operators have become targets of DDoS in the past. In the worst case, this results to a node failure, which possibly impairs the broadcast operation, as we discuss further through our analysis of obstructed block delivery in Section 5.2.3.

5.2.1.4 *Attacks on Transaction Privacy*

THREAT: Previous works also highlighted that the propagation of messages on the network layer may compromise user privacy. In particular, it has been shown that malicious nodes may passively monitor the network and link transaction data to identifying information such as IP addresses, thereby possibly deanonymizing their origin [BKP14; FV17; KKM14]. Furthermore, the notion of AS-level adversaries monitoring the data flows of blockchain networks has been discussed in literature [AZV17; FSW14]. Such passive attacks on privacy are a threat to the Kadcast network protocol in particular, since broadcast messages include a height field, which is additional information that may allow an adversary monitoring transaction propagation to infer her distance from the transaction's origin.

COUNTERMEASURES: Kadcast implements a number of measures to mitigate network-level threats to transaction privacy. Firstly, as previously discussed, the Kadcast protocol implements identifier space randomization which impedes an adversary's capability of learning and exploiting the network topology. Moreover, Kadcast intends for messages to be encrypted at the transport layer [RM12], which hinders an AS-level adversary from passively monitoring the network traffic. In this regard, we suggest to implement a trust-on-first-sight public key pinning policy and to derive the node identifiers from the corresponding public keys, which is an easy way to bind protocol to cryptographic identities without necessitating a dedicated (possibly centralized) public-key infrastructure (PKI). Finally, Kadcast adapts a two-phase transaction propagation scheme similar to the Dandelion [VFV17] protocol, which we further describe and analyze in Section 5.2.4.

5.2.2 *Identifier Space Randomization*

As just discussed, it is essential for network security and privacy to impair an adversary's capability to determine and exploit a node's position in any distribution graph for a given block or transaction broadcast. In particular, an adversary should not be able to pre-populate nodes' buckets with pre-calculated identifiers in a targeted fashion. To this end, Kadcast employs *identifier space randomization* in order to facilitate distribution graphs that cannot be easily anticipated by an adversary. However, a complete randomization of the identifier space would abandon the network structure and hence also render Kadcast's efficient broadcast algorithm based on the XOR-metric non-deterministic. We therefore implement pseudo-randomized views of the identifier space for each broadcast operation. In this regard, additional temporary routing tables are created for each block or transaction broadcast, in which buckets are populated based on a salted distance metric

$$d_s(x,y) = (x \oplus y \oplus s)_{10},$$

where s describes a salt value derived from the message data to be propagated, e. g., the hash of the corresponding block or transaction message. By applying such a temporary transformation to the node identifier space, we make sure that, on the one hand, node locations cannot be predicted by any adversary, but that on the other hand the network view of all nodes remains the same on a per-broadcast basis.

5.2.3 *Obstruction of Block Delivery*

The reliablity of Kadcast depends on the responsiveness and compliance of delegate nodes. An adversary however may have an interest to obstruct the block delivery. To this end, she could position herself on the distribution path during the broadcast operation, and refuse to comply when chosen as delegate. In the following, we will elaborate and analyze this general attack vector.

First, an adversary may try to prevent a specific node from publishing a new block. In order to intercept *outgoing* blocks generated by a target node, an adversary needs to fill every bucket of the target with malicious nodes. We assume that an adversary is able to spawn M out of N nodes, but cannot foresee or cheat the placement mechanism, i. e., has to hash node identifiers like everyone else, resulting in a uniform coverage of the randomized identifier space. In fact, this is a set of very conservative assumptions, since we neglect the previously discussed bucket filling and eviction policies that would heavily skew this towards stable and honest nodes. Moreover, for the sake of censoring outgoing blocks, all

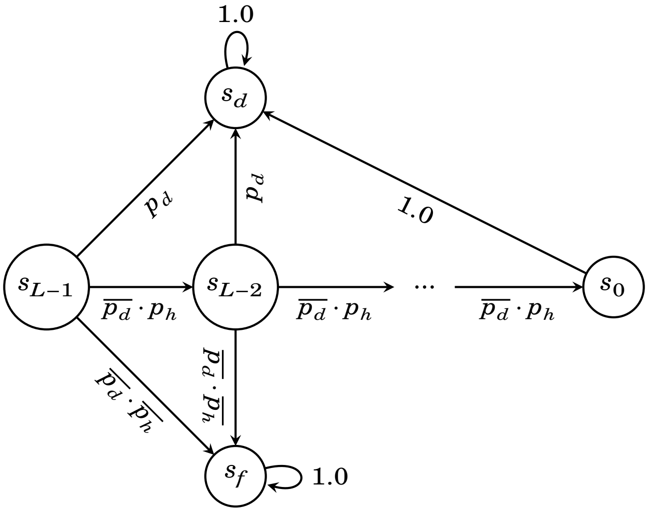

1.0
s_d
p_d
p_d
1.0
s_{L-1}
s_{L-2}
…
s_0
$\overline{p_d} \cdot p_h$
$\overline{p_d} \cdot p_h$
$\overline{p_d} \cdot p_h$
$\overline{p_d} \cdot \overline{p_h}$
$\overline{p_d} \cdot \overline{p_h}$
s_f
1.0

Figure 5.6: Markov chain model.

buckets are equally attractive targets, since the covered space does not only determine the amount of affected nodes, but in equal manner the probability to be selected by the target's broadcast operation. For example, as the bucket size in a network of N nodes can be estimated to be

$$b_{s,i} = \left\lfloor \frac{2^i}{2^L} \cdot N \right\rfloor,$$

a successful attack on the transmission to bucket B_{L-1} may lead to only a coverage of $N/2$ nodes. However, the required number of nodes in this bucket space is also proportionally harder to acquire for the adversary. Due to Kadcast's parallel route selection, it becomes highly unlikely that all β nodes per bucket are picked from the adversary's pool. In particular, when the adversary can acquire control of M nodes, we can assume that the same share, $\epsilon = M/N$, describes the situation in every bucket and hence determines the failure probability of a single broadcast operation. Accordingly, this would result in a parallelized broadcast failure probability of $p_\epsilon = (M/N)^\beta$, which exponentially decreases with the redundancy factor β, as we discussed and analyzed in Section 5.1.3.

The more interesting case is an adversary trying to interfere with the block delivery *to* a specific node. As discussed before, a true Eclipse attack is unfeasible in the Kadcast network, since it strictly applies best practices as well as identifier space randomization. However, in the following we analyze security of block delivery when faced with an adversary that is able to spawn a certain amount of network nodes, i. e., attempting a Sybil attack. In order to calculate the probability of successful block delivery in face of such an attacker, we model the broadcast operation as a simple Markov chain, which is depicted in Figure 5.6. The block propagation starts in an arbitrary distance i from

Table 5.1: Markov Simulation Results

Parameters				**Results**	
N	M	ϵ	β	p_ϵ	p_d
11,000	1,000	0.09	3	0.00075	0.993
12,000	2,000	0.16	3	0.0046	0.957
13,000	3,000	0.23	3	0.0122	0.888
13,000	3,000	0.23	5	0.00065	0.994
15,000	5,000	0.33	5	0.0041	0.963

the target. For instance, if the origin would fall in bucket B_2, the model only needs to consider the operations in height two or smaller. The broadcast operation succeeds, when the block is delivered to the target node, and only honest nodes were visited during path traversal.

The initial state in the Markov model is s_i, and without loss of generality we can assume it to start at s_{L-1}, the state representing broadcast in the largest bucket. From state i, the Kadcast algorithm can delegate the targeted node directly and transition to the *success* state s_d with probability

$$p_{d,i} = 1 - \left(\frac{b_{s,i} - 1}{b_{s,i}}\right)^{\beta}.$$

Alternatively, the algorithm chooses some other node in the bucket with probability $\overline{p_{d,i}}$. The chosen node may be either honest or malicious. If it is honest (again, probability $p_h = 1 - p_\epsilon = 1 - (M/N)^{\beta}$), the broadcast operation continues and the model transitions to state s_{i-1}. If it is malicious, it would obstruct the block delivery, and hence the model transitions to the *fail* state s_f with probability $\overline{p_h} = 1 - p_h$. Once in the *success* or *fail* state, the fate of the broadcast operation is decided, hence the Markov model reaches a steady state after a maximum of $L - 1$ state transitions.

We implemented the Markov chain model utilizing the R package `markovchain` [Spe17], and simulated the success probability of block propagation for different shares of malicious nodes ϵ and redundancy factors β. As these simulations assume the source to be in the bucket of highest distance $(L - 1)$, they yield worst-case estimations for the steady-state success probability. The results are shown in Table 5.1; even for adversaries that control 9 % of network nodes, Kadcast delivers block with more than 99 % probability. Moreover, by adjusting the redundancy factor, Kadcast is able to deliver blocks with more than 96% probability in a highly adversarial environment where 33 % of nodes are controlled by the adversary.

5.2.4 *Privacy of Transaction Propagation*

In the following, we discuss what privacy Kadcast can provide during transaction propagation and what improvements to the baseline protocol should be considered in other to further mitigate the possibility for attacks on user privacy.

5.2.4.1 *Prepending a Random Walk*

As the Kadcast protocol is designed with privacy in mind, its protocol messages generally do not include unnecessary information that may be utilized by an adversary in order to gain additional knowledge on their origin. However, one necessary exception to this rule is the *height* field that is part of the broadcast messages carrying block and transaction data. As this field is initialized with value L and is decreased with every subsequent forward, an adversary receiving a broadcast message is able to infer her distance from the broadcast originator in the propagation graph. This is particular worrisome on the first hop, as an adversary receiving a message with height $L - 1$ would be able to conjecture that the sender of this message is also its origin. While this may be no issue in the case of block propagation—which can be assumed to be not privacy-critical—it may be very detrimental to user's privacy in the case of transaction propagation.

Borrowing from the idea of the Dandelion protocol [Fan+18; VFV17], Kadcast therefore improves the privacy of transaction propagation by prepending the spreading process with a privacy phase, i. e., introducing an initial random walk. To this end, the originator of a transaction broadcast initially sets the height field to an otherwise unused magic number (e. g., $L + 1$) in order to signal that the broadcast is still in the privacy phase and sends corresponding messages to β initial peers. After receiving these messages, each of these peers reads the height field and throws a weighted coin in order to decide whether to continue the random walk, i. e., forward the message to a single other peer, or immediately initiate the spreading of the transaction message as discussed above. It follows that the expected length of the random walk can easily be adjusted through the weight parameter of the coin toss.

5.2.4.2 *Resilience to Passive Attacks on Privacy*

In order to study the privacy that Kadcast can provide, we in the following assume scenarios in which an adversary controls an embedding of M out of N nodes that passively monitor the network for propagated transactions. We furthermore assume that the adversary has no knowledge of additional information that would allow her to infer the propagation path, but is only capable of passively recording from which node the controlled observation points received a transaction. That is, the adversary classifies observations according to a so-called *first-spy*

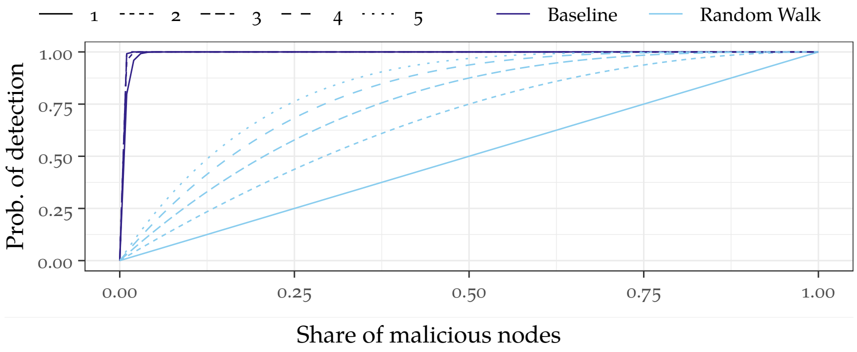


Figure 5.7: Probability of detection with and without a prepended privacy phase in dependence of the share of malicious nodes and the parallelization factor β ($L = 160$).

estimator model and assumes the first encounter to be the transaction's origin [VFV17]. As additional confounding factors—such as considering varying verification times, link latencies, or bandwidths—would only increase entropy and hence reduce an adversary's capabilities, we omit modelling any message delivery timings, but assume that network messages arrive in the order in which they were initially sent.

As a consequence, the adversary estimates the correct transaction origin, if *any* of the M malicious nodes receive a propagated message first, i. e., it is selected by the origin as one of the $\beta \cdot (L-1)$ initial bucket delegates. This *probability of detection* is the probability that at least on malicious node is initially chosen, i. e.,

$$p_m = \overline{p_h} = 1 - \left(\frac{N-M}{N}\right)^{\beta(L-1)} .$$

By utilizing the privacy phase, the number of delegates directly contacted by the originator is reduced from $\beta \cdot (L-1)$ to β, hence yielding a modified probability of detection

$$p'_m = 1 - \left(\frac{N-M}{N}\right)^{\beta} .$$

Figure 5.7 shows the probability of detection with and without a prepended privacy phase in dependence of the parallelization factor β. We observe that in the baseline Kadcast protocol, an adversary would indeed have a really high probability of guessing the originator of a transaction correctly. However, we also can see that the privacy phase immensely improves this and depending on the degree of parallelization may even come close to the optimal probability proportional to the share of malicious nodes M/N. Finally, we observe that there is a trade-off between privacy and reliability: on the one hand, the more nodes β the transaction originator initially contacts, the higher is the probability that a passive adversary gains information on the transaction's origin. On the other hand, as we previously explicated, a too low β value may leave the broadcast operation open for censorship attacks.

5.3 EVALUATION: NETWORK SIMULATIONS

In this section, we evaluate the block distribution performance, the broadcast reliability and efficiency, as well as the security impact of the Kadcast protocol on an empirical basis. For this, we gathered data from a comprehensive network simulation study, which are discussed in the following.

5.3.1 *Simulation Model*

Our network simulation study utilizes the `bns` blockchain simulation framework introduced previously. The parameters that fuel our blockchain simulation are therefore mainly chosen in reference to the Bitcoin network, as discussed in Section 3.2.

While our evaluation is based on a number of different setups, the results described in the following are based on scenarios simulating the mining process in networks with $N = 500$ nodes. Every scenario was repeated 30 times for different seed values to ensure statistical significance of the conducted measurements. During the three hour simulation time, the miners generated blocks and initiated broadcast operations employing one of the networking stacks, i. e., the unstructured *Vanilla* baseline or the *Kadcast* protocol. We furthermore implemented and evaluated the impact of different messaging patterns, i. e., *header*, *cmpctblock*, and *unsolicited* block propagation for the Vanilla stack, as well as *cmpctblock* and *unsolicited* propagation for Kadcast. In the Kadcast case, if not stated otherwise, the results are based on the default parameters $L = 64$, $k = 100$, $\alpha = 3$, $\beta = 3$, and an FEC overhead factor $f = 0.05$.

In order to analyze the protocol behavior under different network conditions, we furthermore evaluated all protocol variants in the Hub & Spoke, as well as in the geographic topology models (cf. Section 3.2.2). The Hub & Spoke model is a typical setup for the assessment of peer-to-peer overlays, and while it captures some network effects, it does not rely on additional assumptions about the underlying topology. As it furthermore assumes the Internet connectivity to not be a bottleneck, it creates an idealized simulation scenario that gives us the capability to assess the networking stacks based on a neutral, common ground. The geographic topology model however captures a heterogeneous and partly resource-restricted environment that enables simulations in more complex network scenarios, which incorporate a high degree of network effects.

5.3.2 *Protocol Evaluation*

In order to show the benefits of the Kadcast protocol in different environments, we created simulation scenarios with parametrizations

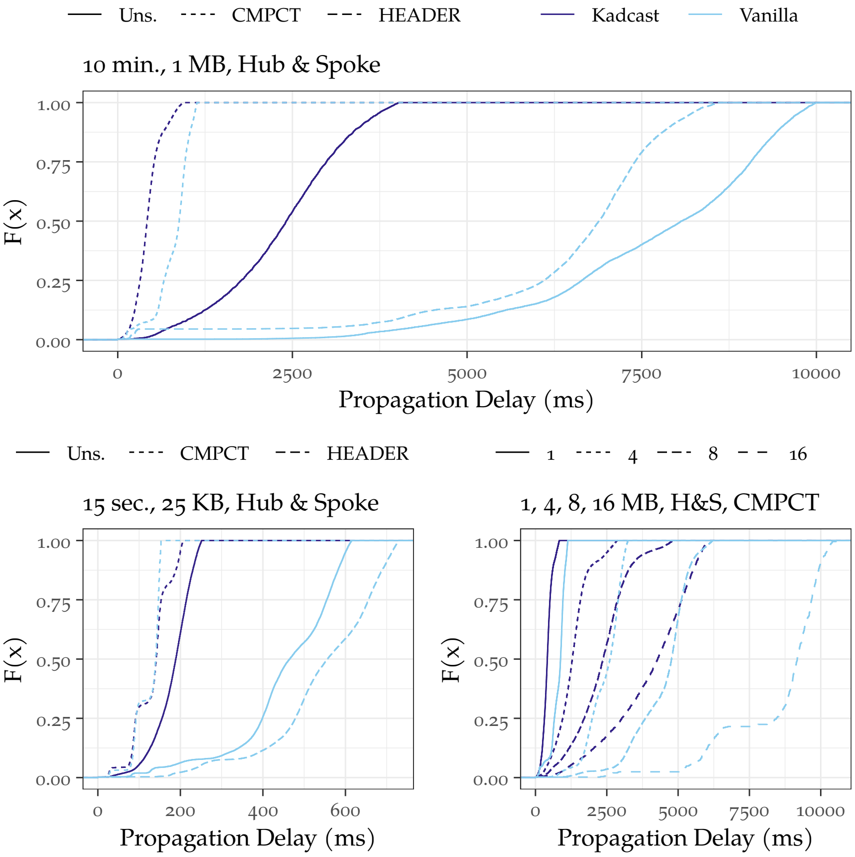


Figure 5.8: Block propagation delay for a different parametrizations in the Hub & Spoke model.

mimicking Bitcoin (10 min. block interval and 1 MB block size limit) and Ethereum (15 sec. block interval, proportionally smaller block size limit of 25 KB [@Eth19b]). Moreover, in light of the debates on block size limits in the Bitcoin community, we additionally analyzed the block propagation for increased block size limits of 4, 8, and 16 MB.

5.3.2.1 *Block Propagation Delay*

As a first study, we investigated the performance of Kadcast compared to different instantiations of Vanilla broadcasting. Figure 5.8 shows the block propagation delay to reach 90% of all nodes as cumulative distribution functions $F(x)$: as expected, the block distribution time heavily depends on the block size and block intervals, as well as the employed messaging pattern. The Kadcast protocol, however, delivers blocks significantly faster compared to Vanilla in all cases. For example, Kadcast exhibits a mean propagation time of 436 ms to deliver compact blocks with a 1 MB block size limit (Bitcoin-like scenario, upper plot), which is close to twice as fast as Vanilla with enabled compact blocks. The mean propagation delay for unsolicited Kadcast propagation is 2,349 ms, 63%

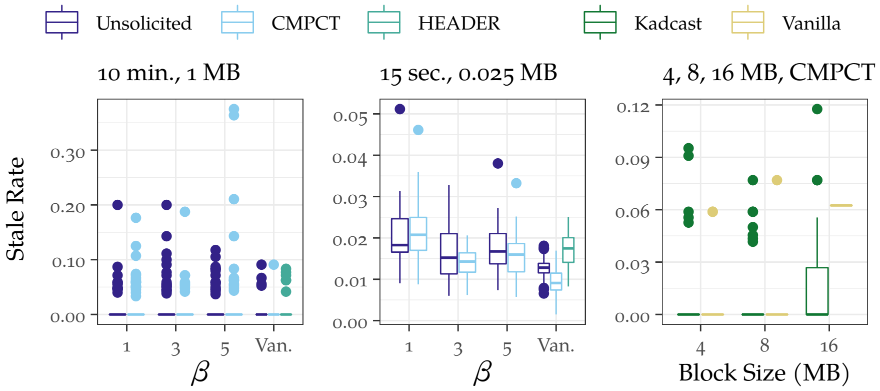


Figure 5.9: Stale rates for different parametrizations in the Hub & Spoke model.

faster than header-based, and even 70% faster than unsolicited Vanilla propagation in Bitcoin-like scenarios.

The improved propagation speed is also reflected by an overall faster network coverage: while it takes Vanilla in the Bitcoin case with enabled compact blocks 1,133 ms to reach 90% of the network, Kadcast is able to reach the same number of nodes around 22% faster. Unsolicited propagation via Kadcast reaches 90% coverage even respectively 53% and 60% faster than header-based and unsolicited propagation in the unstructured Vanilla networking layer.

Furthermore, as shown in the bottom-left of Figure 5.8, Kadcast's performance is competitive with Vanilla's in the case of smaller intervals and smaller block sizes: in the Ethereum-like scenario, unsolicited Kadcast is able to deliver blocks on average more than 60 % faster than the unsolicited Vanilla baseline, and even 66% faster than the header-based propagation. However, Vanilla performs slightly better in the compact block case, in which it is around 7% faster than Kadcast. For the larger block sizes of 4 MB, 8 MB, and 16 MB (cf. the bottom-right plot), Kadcast is on average also able to consistently deliver blocks more than 50% faster.

These results highlight on the one hand the messaging pattern has a significant impact on the performance of block propagation. On the other hand, we conclude that Kadcast is able to immensely speed up the block distribution and may be beneficial to a wide variety of blockchain networks.

5.3.2.2 *Impact on Consensus Stability*

The effect of quicker block propagation is also reflected in the median stale rate, i. e., rate of blocks that are mined, but do not become part of the final blockchain. As increased blockchain forks and wasted mining power weaken consensus security, the stale rate is an indicator for how the networking layer impacts security [Ger+16].

The boxplots in Figure 5.9 show the stale rate in dependence of the applied messaging pattern and different choices for the redundancy parameter β: in the Bitcoin-like scenario Kadcast achieves a median stale rate of zero, barring the occasional outliers. This is comparable to the Vanilla cases, which exhibit similar behavior. However, in the case of a decreased block interval, the ratio of propagation delay to block interval gets much larger, resulting in an overall increased stale rate of around 1-2%. In this Ethereum-like scenario, again the influence of the messaging pattern is clearly visible, as compact block based propagation beats unsolicited and header-based propagation every time. In accordance with the results of the previous section, the compact block based Vanilla variant achieves the lowest median stale rate of 1.3%.

The simulations with compact block relaying of larger blocks indicate that the additional stress on the network layer negatively impacts consensus security: while Vanilla and Kadcast can mostly retain a median stale rate of zero, the number of outliers increase in both cases. Though, in the case of the 16 MB block size limit, a clear difference becomes apparent, as Kadcast can retain an average stale rate of 1.6% versus Vanilla's 6%.

In summary, the improved block propagation of Kadcast leads to a median stale rate that is comparable and often better than Vanilla. This indicates that the consensus security of blockchain systems could benefit from employing the Kadcast protocol. Moreover, since blocks reach a larger share of the network much faster, the adoption of Kadcast could help to mitigate time-dependent adversarial mining strategies, such as selfish mining [ES14].

5.3.2.3 *Broadcast Efficiency*

In order to confirm the adjustability and efficiency of the Kadcast protocol, we recorded the total amount of traffic t_{total} produced during our simulation time. Furthermore, we accumulated the block sizes of all blocks generated during this time, t_{blocks}. As all blocks need to be transmitted to each node at least once, the minimum amount of traffic for the broadcast operation can be calculated as $N \cdot t_{blocks}$. Accordingly, we define the overhead ratio as $r_o = (t_{total} - N \cdot t_{blocks}) / (N \cdot t_{blocks})$, which describes how much additional traffic was generated during a simulation run, including all signaling messages.[2]

Figure 5.10 shows the resulting overhead ratios in dependence of the applied messaging patterns and different parametrizations of the redundancy parameters β and f. Firstly, we observe that Kadcast's overhead immensely depends on the applied messaging pattern and that it increases linearly with the redundancy factors β and f. As expected,

---

[2] Note, that since we only consider block propagation for the traffic estimation and due to the existence of stale blocks, the overhead ratio may assume values below 1 or even 0, which however does not impair the validity of this metric.

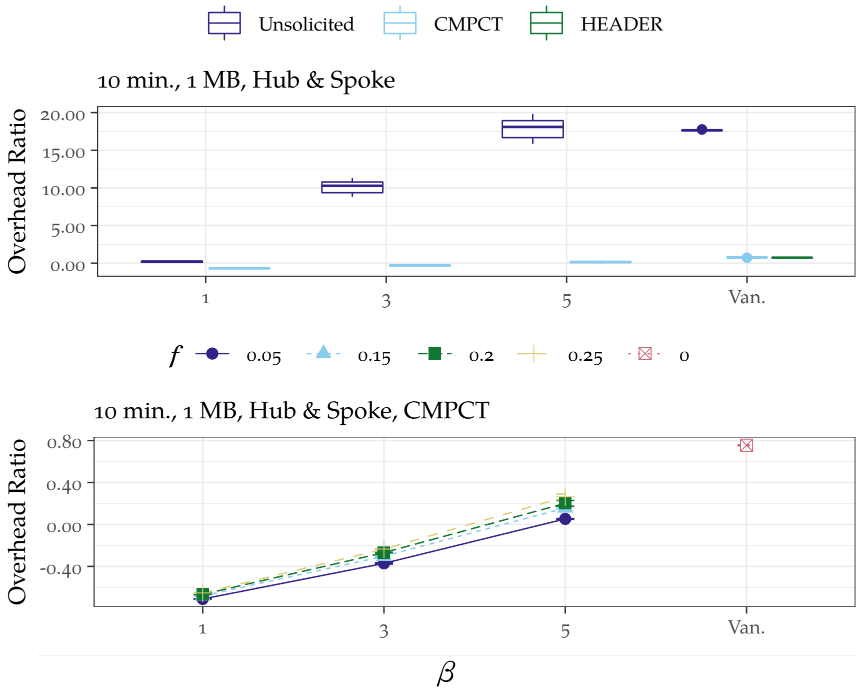


Figure 5.10: Overhead ratios in dependence of applied messaging pattern and parametrizations for f and β.

unsolicited block propagation overall results in a significantly higher overhead than the propagation through headers or compact blocks. We moreover note that throughout all parametrizations, Kadcast's overhead remains below its Vanilla counterpart: for $\beta = 1$, Kadcast even with unsolicited block relay results in an overhead ratio below the header-based and compact block variants of Vanilla, and for $\beta = 5$, it is comparable to unsolicited Vanilla propagation. In the bottom half of Figure 5.10, we can finally observe that the overhead of compact block propagation through Kadcast is consistently below the Vanilla counterpart. This shows the adjustability of the Kadcast approach, which allows for a fast block relay with low and controllable overhead.

5.3.3 *Protocol Behavior under Attack*

We moreover empirically evaluated how the Kadcast protocol fares in the face of an adversary obstructing block delivery. For this, we set up simulation scenarios in which a fraction ϵ of nodes were marked as adversarial and henceforth would cease to forward blocks. The upper part of Figure 5.11 shows the network coverage in dependence of ϵ and β: while Kadcast of course reaches 100 % network coverage for $\epsilon = 0$, its block propagation is severely hindered when malicious nodes are introduced and no redundancy exists ($\beta = 1$). However, the effect of the redundancy factor β is also clearly visible, ensuring 99 % coverage

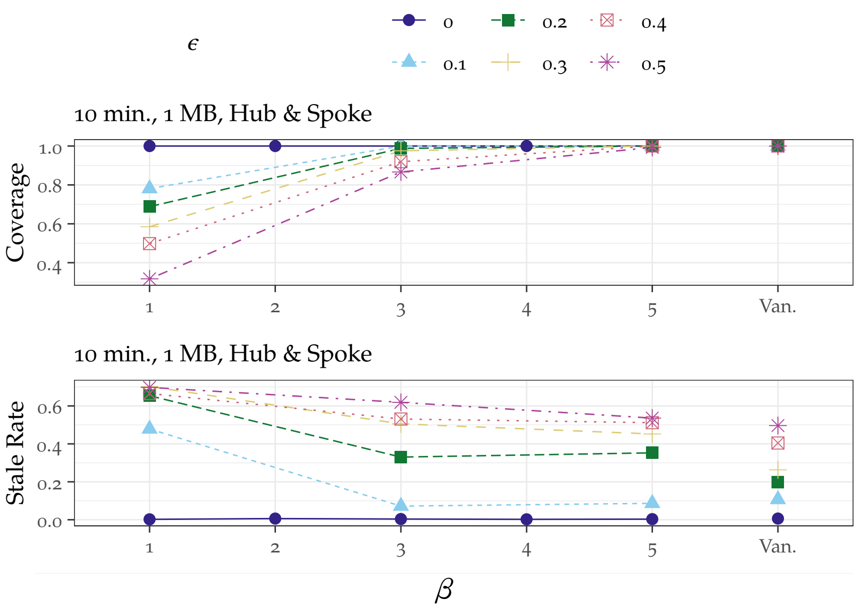


Figure 5.11: Network coverage and stale rate, when a share ϵ of adversarial nodes is introduced to the network.

for $\beta = 3$ when $\epsilon \leq 0.3$ and for $\beta = 5$, when the share of malicious nodes would be even higher.

Interestingly, while Vanilla's network coverage is not impaired by the introduction of adversarial nodes, it does exhibit degraded propagation performance due to the almost fragmented network. In fact, the resulting stale rates of both protocols are very similar, when confronted with such a powerful adversary (cf. lower part of Figure 5.11). The results show that, with a reasonably chosen set of parameters, Kadcast is resilient to a large amount of adversarial nodes and compares to the currently deployed Vanilla networking layer.

5.3.4 *Protocol Behavior in Heterogeneous and Resource-Restricted Environments*

Additionally, in order to evaluate the Kadcast protocol in more heterogeneous networking environments, we reproduced the previously introduced scenarios in the resource restricted geographic topology model. Due to the lower bandwidths, larger latencies, packet losses, and more complex structure of this model, much more network effects come into play here. However, the results shown in Figure 5.12 follow the same tendencies as discussed before: in general, Kadcast provides faster block propagation than Vanilla. And again, it does so especially when compact block relaying is enabled.

Nevertheless, the results also show that the UDP-based Kadcast implementation may exhibit degraded performance when facing slower or more congested networking environments. In the case of unsolicited

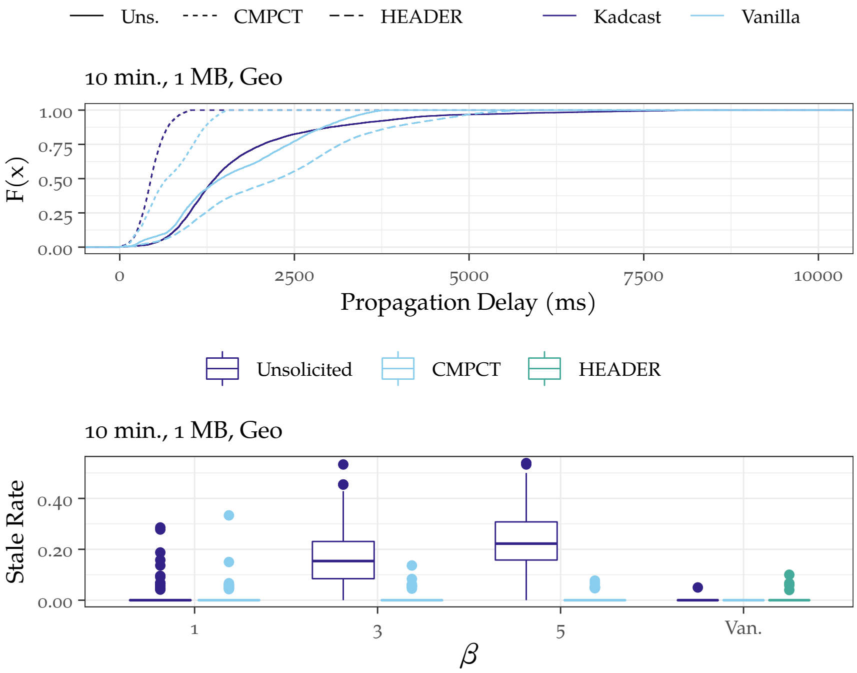


Figure 5.12: Block propagation delays and stale rates for different parametrizations in the geographic topology model.

block propagation through Kadcast, we can already see the first signs of congestion: even though the average propagation delays are comparable with Vanilla, the propagation delay distribution exhibits a longer tail. The degraded network performance is also reflected by higher stale rates for $\beta > 1$. As the effect is only exhibited with unsolicited block propagation, enabling compact block relaying ascertains low stale rates even in the face of these restricted and heterogeneous environments.

However, as the results highlight the robustness of the TCP-based Vanilla in comparison to the UDP approach of this Kadcast implementation in heavily congested networks, we consider the implementation of Kadcast on the basis of a suitable congestion control mechanism to be a logical next step. As our simulation results furthermore show that degraded performance may be mitigated by compact block relaying schemes, more blockchain networks should consider adopting such a beneficial messaging pattern. This could help to further alleviate network bottlenecks in heterogeneous environments and hence foster decentralization.

5.4 EVALUATION: TESTBED DEPLOYMENT

In the following, we evaluate the properties of the Kadcast protocol through the deployment of a prototype implementation in a large-scale testbed.

5.4.1 *Kadcast-NG: A QUIC-based Prototype Implementation*

In order to enable testing and evaluation in real-world networks, we developed `kadcast-ng`, a prototypical node implementation of the Kadcast protocol written in Rust [@Tea21].[3] Following the separation of concerns principle and for the sake of comparability, `kadcast-ng` is designed to run on top of the Bitcoin Core [@Bit19] reference implementation of the *vanilla* Bitcoin protocol. To this end, the prototype leaves the enforcement of the consensus protocol to `bitcoind`, with which it interacts through the RPC [@Bit21a] and ZeroMQ [@The21] interfaces. This allows `kadcast-ng` to merely act as a relay node that receives and submits transaction and block data which it (de-)serializes using the `rust-bitcoin` [@Rus21] library implementation.[4]

As we have seen in the previous section, Kadcast may exhibit degraded performance in congested network settings when it is implemented on top of the unreliable UDP transport protocol. We therefore base the `kadcast-ng` prototype implementation on the reliable QUIC [Lan+17] transport protocol, which fits the Kadcast use case very well: QUIC features data transmission via multiple concurrent streams, which eliminates head-of-line blocking and therefore reinforces the benefits of Kadcast's parallelized broadcast operation. In contrast to the comparatively slow TCP handshake, QUIC furthermore facilitates connection establishment in one or even zero round-trip times (RTTs). This is an important feature, since it enables `kadcast-ng` to continue to follow the lightweight soft-state approach in which short-lived connections are only established when they are needed. Moreover, QUIC is an encrypted and authenticated transport protocol which provides many benefits to the security and privacy of the Kadcast protocol, as discussed in Section 5.2. In this regard, the `kadcast-ng` prototype utilizes the asynchronous Quinn [@Dir21] library implementation of the QUIC protocol and establishes an encrypted overlay network based on a trust-on-first-sight principle.

5.4.2 *Testbed Setup*

In accordance with real-world blockchain networks, we create a testbed scenario that enable the evaluation based on a large and distributed network setting. To this end, we deployed 1,000 instances of type `e2-small` in different regions of the Google Compute Cloud. In particular, we follow the previously measured node distribution and deploy the instances close to the regional center of the geographic topology model (cf.

---

[3] The `kadcast-ng` codebase is open source and publicly available under: `https://git.tu-berlin.de/rohrer/kadcast-ng-public`

[4] At the time of writing, `rust-bitcoin` does not (yet) support Bitcoin's compact blocks [@Cor16]. The `kadcast-ng` prototype implementation is therefore currently also limited to unsolicited block and transaction propagation.

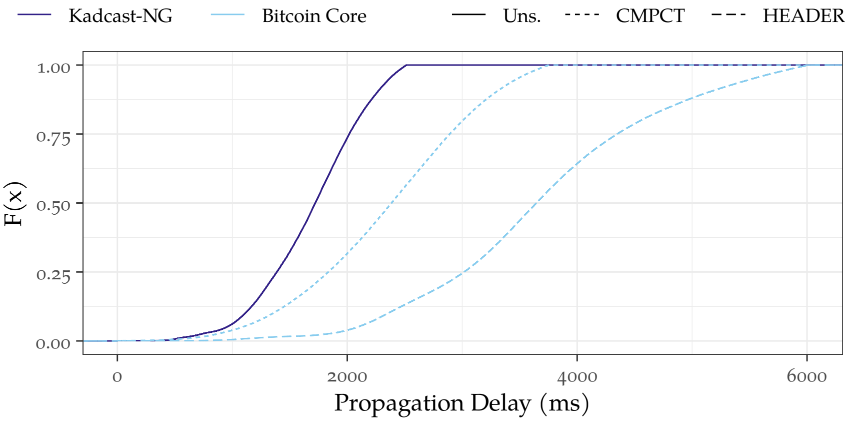


Figure 5.13: Block propagation delays for the QUIC-based `kadcast-ng` prototype and Bitcoin Core in a testbed of 1,000 nodes.

Chapter 3), i. e., in the regional zones: `us-west1-a` (NA), `southamerica-east1-a` (SA), `europe-west3-a` (EU), `asia-south1-a` (AS), `australia-southeast1-a` (OC), and `asia-east2-a` (CN).

All node instances are provisioned with Ubuntu Linux 20.10, Bitcoin Core in `regtest` mode, and `kadcast-ng`. Of all deployed nodes, 15 nodes are chosen as "miners" based on the geographic mining distribution. They are then configured with a certain *mining rate*, which denominates at which exponentially distributed rate they should produce blocks to induce a network-wide average block interval of 10 minutes. According to this parameter, new blocks are generated through the invocation of `bitcoind`'s `generatetoaddress` remote procedure call, which in the `regtest` mode allows to instantaneously create blocks, i. e., simulate the mining process without actually searching for a proof-of-work solution. In order to fill the blocks, new transactions are created based on a *transaction rate* parameter, which we assume to be uniformly distributed over all nodes. To this end, on average a total of 2,000 transactions should be created for every block interval through the distributed invocation of the `sendtoaddress` RPC. This parameter corresponds to the current average number of transactions per block in the Bitcoin network [@Blo21c]. To enable this distributed issuance of transactions, we deploy a pre-generated blockchain and funded wallets to each individual node during provisioning.

As before, we configure `kadcast-ng` with the default parameters $k = 100$ and $\beta = 3$, and employ an initial waiting period to ensure that all nodes have finished their initial bootstrapping before the experiment's measurement period of 24 hours starts.

5.4.3 *Block Propagation Performance*

In order to study the block propagation performance of the `kadcast-ng` prototype, we evaluated its unsolicited broadcast in comparison to the Bitcoin Core baseline with and without compact block [@Cor16] relaying enabled. Over the measurement period of 24 hours an average of 143 blocks were produced and propagated in the described testbed scenarios of 1,000 nodes, each of which recorded the time each of the blocks arrived from the ZeroMQ interface. When evaluating the Bitcoin Core scenarios, block and transaction propagation via Kadcast were disabled, but the measurements were still conducted via the `kadcast-ng` node in an identical fashion in order to ensure optimal comparability.

The results are shown in Figure 5.13: firstly, we observe that the characteristics of the block propagation behavior concur with the simulated scenarios, thereby validating our prior results. Moreover, we observe that also in the testbed deployment Kadcast is able to show its benefits, resulting a block delivery time that is on average 43% faster than the header-based propagation of Bitcoin Core, and even still 27% faster than the case with enabled compact block relay.

The evaluation of the QUIC-based `kadcast-ng` prototype implementation therefore does not only show the general feasibility of a large-scale deployment of the Kadcast protocol, but also once more underlines its significant impact on the block propagation performance in many scenarios.

Part II

PAYMENT CHANNEL NETWORKS

6

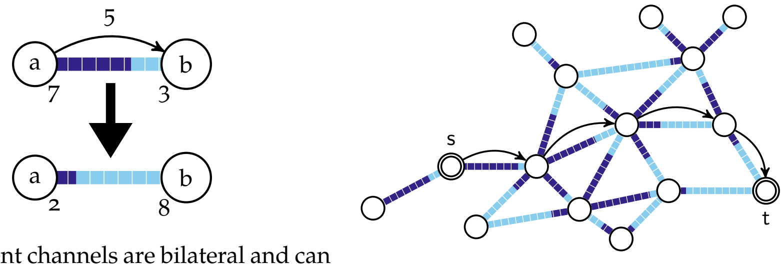
STATE OF PAYMENT CHANNEL NETWORKS

So far, we studied properties of real-world blockchain networks, discussed the limitations inherent to their design, and presented new networking layer protocols that improve on-chain scalability. We now turn our attention to the possibility of *off-chain* scaling, i. e., the notion that transactions can be securely processed on a second layer, which promises to improve transaction throughput, latency, and privacy.

Second-layer solutions, such as Bitcoin's Lightning Network [PD16] or Ethereum's Raiden [@Rai20] promise to mitigate the shortcomings of the base layers by establishing a network of off-chain payment channels. Payment channels enable rapid payment processing between the two channel endpoints without consulting the blockchain every time. That is, incremental updates are negotiated locally instead of requiring a global agreement. Local transactions are not only much faster, but also do not leak information to noninvolved third parties. However, this local negotiation is only possible in a secure manner, because the parties deposit a collateral during the initial channel establishment. In the following, we give a primer on the principles enabling transaction processing in the Lightning Network and discuss work related to our contributions towards this field of research.

6.1 LIGHTNING NETWORK PRIMER

The Lightning Network is the most prevalent *payment channel network* (*PCN*) to date, i. e., it is a network of *payment channels* which are established between two endpoints by locking a certain amount of funds (the channel *capacity*) on-chain. The transfer of funds over a channel is performed through updating the individual channel allocations, i. e., the respective channel *balances* (see Figure 6.1a), which can be negotiated rapidly between the two involved parties. Payments can also be routed over multiple intermediate channels, which allows to send funds to remotely connected receivers (see Figure 6.1b). This multi-hop payment process is secured through the application of a *Hashed Time-Locked Contract* (*HTLC*) protocol. The HTLC protocol ensures that a intermediary node forwarding a payment is reimbursed in the case of success, and in case of a payment failure may still retrieve its locked funds after the *time-lock delta* safety period has passed.



(a) Payment channels are bilateral and can be used to transfer funds through updating the balances.

(b) Payment channel networks facilitate multi-hop payments.

Figure 6.1: Visualization of payment channels and payment channel networks

6.1.1 *Connection and Channel Establishment*

A new peer joining the Lightning Network has first to establish a network connection to a node connected to Lightning's TCP-based peer-to-peer overlay network. Since every node in the network holds an associated long-term `secp256k1` ECDSA public key [Bro10] by which it is identified, all inter-peer communications following the initial key exchange handshake are authenticated and encrypted based on the Noise [@Per18] protocol framework.

In order to initiate the establishment of a new payment channel to a neighboring node, the peer sends an `open_channel` message that is typically answered by an `accept_channel` message, through both of which the channels parameters—in particular the channel capacity and initial balances—are negotiated. Using the exchanged information, the initiating peer is then able to issue a funding transaction which it broadcasts in the Bitcoin network. After the funding transaction is confirmed on-chain, the channel is established and may be used for payment processing. Furthermore, if the new peer wants to act as payment hub, i. e., forward payments for others, it can announce the node's and channel's existence to the network by disseminating the respective `node_announcement` and `channel_announcement` messages in the peer-to-peer network. As these messages also contain the necessary routing information, such as the channel capacity and associated routing fees, they are broadcasted in Lightning's overlay network. These messages also include the `cltv_expiry_delta` parameter, which allows a node to declare the maximum time it is willing to have its funds locked up in case an HTLC is not fulfilled in an orderly fashion.

6.1.2 *Payment Routing*

Let us assume that Alice already connected her node A to the network and established at least one channel over which she is able to send and receive payments. If she now wants to send a payment to a destination node C, she has to first find a suitable path in the network and then has

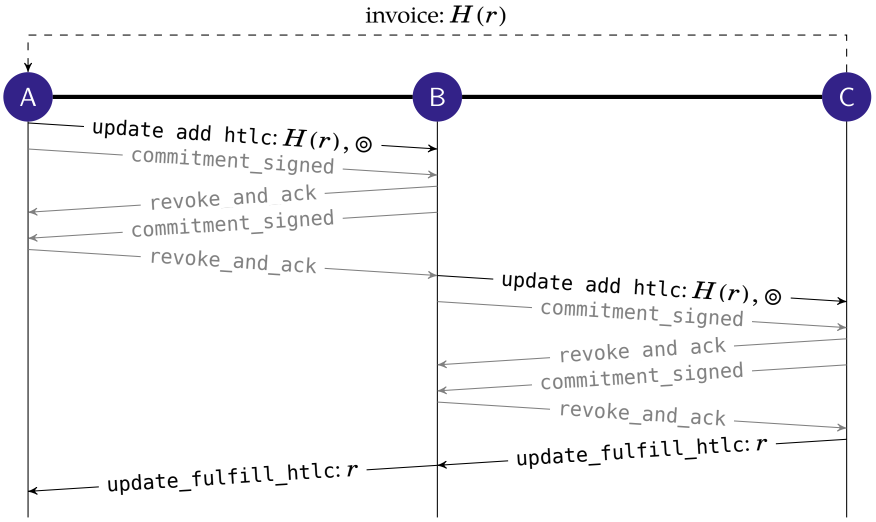


Figure 6.2: Message exchange during payment routing.

to setup the corresponding HTLC to conduct the payment. In order to illustrate this example, the sequence of exchanged messages is shown in Figure 6.2.

Initially, it is required for A and C to have some out-of-bounds communication channel over which C can supply an *invoice* to A that includes, without limitation, its identifying public key, the amount to be paid, as well as the *payment hash*, i. e., the hash $H(r)$ of a random secret r. Based on the publicly available routing information, A then employs *source routing* in order to determine a path to C that has sufficient capacity to possibly be able to route her payment. While the behavior of the source routing algorithm is not specified as part of the BOLT specifications [@Dev20c], typically a modified version of Dijkstra's shortest path algorithm [Dij59] that considers routing fees and past payment success is utilized for route selection. Note that this algorithm might fail, if there is no path of sufficient capacity available. However, let's assume without loss of generality that this is not the case and the algorithm yields a path over the intermediate node B.

Given the discovered path, Alice is able to initiate the HTLC construction, i. e., a number of conditional payments that either may be redeemed by producing the pre-image r to the challenge $H(r)$, or would time out after a certain lock-time. In order to facilitate the payment, A calculates two essential values for each respective hop:

1. the amount this hop should forward, which is calculated by adding the accruing fees for each respective hop to the payment amount, which is hence decreasing towards the destination.

2. the necessary remaining time-lock value for the outgoing hop, which is increasing towards the destination.

Alice then encodes this information in an onion routed packet corresponding to the Sphinx packet scheme [DG09]. That is, the packet is constructed through multiple layers of encryption, each wrapping information identifying the next hop, the amount to forward, as well as the remaining lock-time value. She then initiates the payment by sending an `update_add_htlc` message carrying the payment hash $H(r)$ as well as the onion packet (represented as ◎) to the next hop, i. e., B. The latter, as each intermediate node along the payment path, is then able to decrypt the packet to receive its payload and forward the HTLC offer to the next hop. However, B only proceeds after the new conditional payment based on the challenge $H(r)$ is incorporated in the state of the affected payment channel and this state change is *irrevocably committed* through a handshake of `commitment_signed` and `revoke_and_ack` messages.

After the pending state updates have been negotiated, C forwards the HTLC by attaching the remaining part of the onion packet to an `update_add_htlc` message sent to the next hop, which proceeds in the same way. In case any of the intermediary nodes does not agree with the payment, e. g., when the channel does not hold a sufficient balance or the fee and time-lock values determined by A do not meet their expectations, they may fail the HTLC by replying with an `update_fail_htlc` message carrying failure message that is onion-encrypted and propagated back along the path to the origin node A. As this may happen at any point in time, Lightning does not provide any guarantees on payment reliability and hence can be classified as a best-effort network.

Once the HTLC construction reaches the final destination, C supplies the solution r to the payment hash challenge $H(r)$ via a corresponding `update_fulfill_htlc` message, which is propagated back on the inverse payment path, allowing intermediary nodes to redeem their conditional payments. Thereby, they settle the pending HTLC and gain the determined fee. Note that while the `commitment_signed` and `revoke_and_ack` messages are only exchanged between immediate neighbors, the respective `update_add_htlc`, `update_fulfill_htlc`, and `update_fail_htlc` messages are forwarded back and forth the payment path, which makes them observable by intermediate nodes.

6.2 RELATED WORK

6.2.1 *Channel Design and Payment Routing*

Payment channels were introduced to scale cryptocurrencies to high transaction rates. To this end, several channel designs have been proposed over the past years [DRO18; DW15; GM17; Hei+17; Mil+19]. While some designs are restricted to single-hop payments, others support multi-hop payments and may therefore be used in payments channel networks. The Lightning Network [PD16] for Bitcoin and the Raiden

Network [@Rai20] for Ethereum are the most widely used channel implementations to date. A further overview of the PCN design space is provided by Gudgeon et al. [Gud+20].

Besides the makeup of the payment channels themselves, the question how payments can be routed efficiently and securely has recently been a focus area of PCN research. Prior contributions introduced algorithms beyond the currently utilized source-routing, for example landmark-based [Pri+16; Wan+19] and distributed [Roo+18; Yu+18] routing algorithms have been explored. As our work introduced in Chapter 8 showed, the network may be utilized more efficiently when larger payments are split up and routed over multiple payment paths. To this end, algorithms considering multi-path routing [BNT20; Siv+18] and payment splitting [Eck+20; PN18] have been discussed in literature.

6.2.2 *Network Graph Properties and Creation*

In recent years, a number of works analyzed the Lightning Network graph and discussed the resulting decentralization and reliability of payment routing [Lin+20; Ser+19; WH20], as well as possible consequences with regard to payment privacy [BSB19a; MF20]. Our work presented in Chapter 7 introduced the first study to analyze graph-theoretic properties of the Lightning Network's channel topology and highlighted its vulnerability to targeted attacks.

While most of these entries take the network topology as a given, few entries study how the network structure emerges and which algorithms for graph creation are preferable. To this end, prior entries showed that centralized structures can make the network more efficient and stable [Ava+20; Rin+20; SZ20]. Most related to our work studying attachment strategies for payment channel networks (Chapter 10), Ersoy et al. [ERE20b] introduce a strategy aiming for profit maximization of payment providers. As the utilized approximation algorithm is however very costly, our work deliberately opts to implement a more practical approach and explores a broader range of strategies and design goals. Related to our work are moreover approaches for optimal channel balance calculation [LMZ20], rebalancing [KG17; PN20], as well as algorithms considering the network demand [Aum+20; KR21].

6.2.3 *Attacks on Security and Privacy*

More and more works study the possibility of attacks on the security and privacy of second-layer solutions in general and payment channel networks in particular. Our analysis presented in Chapter 7 showed that the Lightning Network is vulnerable to channel exhaustion and node isolation attacks, both of which are particularly dangerous if the adversary can leverage *payment griefing* in order to keep its stakes low. Mizrahi and Zohar showed that the network in similar fashion may suffer from congestion attacks [MZ20]. Moreover, Tochner et al. showed how the routing algorithms employed by PCNs may be manipulated in order to facilitate inclusion of a malicious node in the payment path, thereby increasing the danger of denial-of-service attacks [TSZ19]. Additionally, previous entries showed the HTLC constructions to be susceptible to bribing [KNW20; TYE20] and flooding attacks [HZ20].

Furthermore, recent entries have discussed the possibility of discovering the private channel balances by probing [Dam+20; Her+19; Tik+20] and analyzed how much privacy could be retained, if noisy channel balances were to be made public [Tan+20]. Similarly, Béres et al. [BSB19a] and Tikhomirov et al. [TMM20] empirically analyzed the privacy properties of PCNs with the assistance of model-based traffic simulation. While both of these entries discuss the possibility of deanonymization attacks, they essentially apply a variant of the *first-spy* estimator, i. e., estimate immediate predecessors and successors to be senders/receivers. Concurrently to our work of Chapter 9, Kappos et al. [Kap+20] refined prior approaches of traffic simulation and introduce a probabilistic model based on observed path lengths in order to estimate probable payment endpoints. Moreover, while a recent entry by Nisslmueller et al. [Nis+20] mentioned the possibility of timing attacks, their investigation remained in a preliminary state. To the best of our knowledge, our work presented in Chapter 9 is therefore the first to study the impact and feasibility of timing attacks on privacy in payment channel networks in depth.

Orthogonally to this work, a growing body of contributions aims to improve the privacy guarantees of second-layer solutions. While some designs allow to anonymously transact over payment hubs [GM17; Hei+17], privacy-preserving routing mechanisms [Mal+17a; Maz+20; Roo+18] promise to enable anonymous transactions over multiple intermediaries, i. e., payment channel networks. Malavolta et al. proposed provably secure payment protocols [Mal+17b] and introduced a Lightning-compatible anonymous locking mechanism based on ECDSA signatures that allows for the decorrelation of payment paths [Mal+19]. While the adoption of such improved protocols would likely not entirely mitigate the possibility of attacks on privacy, they would force an adversary to take additional error-prone measures for payment correlation.

7

ANALYZING AND ATTACKING THE LIGHTNING NETWORK TOPOLOGY

This chapter is based on the publication: Elias Rohrer, Julian Malliaris, and Florian Tschorsch. "Discharged Payment Channels: Quantifying the Lightning Network's Resilience to Topology-Based Attacks." In: S&B '19: Proceedings of IEEE Security & Privacy on the Blockchain. June 2019 [RMT19]. While this was a collaborative effort, the implementation of the network analysis should be attributed to fellow co-authors.

In this chapter, we analyze the Lightning Network's topology and provide details on the network's properties, as well as its resilience towards random failures and targeted attacks. To this end, we introduce the notions of *channel exhaustion* and *node isolation* attacks, and evaluate the feasibility of a series of adversarial strategies. Most notably, we show that the Lightning Network is susceptible to these attacks and can indeed be disrupted.

7.1 LIGHTNING NETWORK ANALYSIS

In the Lightning Network, every node has a global view of the payment channel network (PCN) topology and is responsible for finding routes on the basis of this data set, i. e., conducting source routing. However, mainly due to privacy reasons, actual channel balances are not included in this routing information. As a consequence, it cannot be predetermined whether sufficient funding is available to route a payment. In case of a failed payment attempt, the client needs to repeat the process until successfully completed.

The payment channel design of the Lightning Network ensures balance security for multihop payments through the use of hash time locked contracts (HTLCs). We can see that HTLC processing requires all participants to be online and responsive. Otherwise, if a node goes offline, funds may be locked for an extended time period. In the worst case, it is even possible that an adversary publishes outdated states, effectively stealing coins. The robustness of the PCN topology therefore is the baseline for the resilience of the Lightning Network, e. g., to node failures due to DoS attacks. In the following, we evaluate metrics like the betweenness centrality, clustering coefficient, and degree distribution to draw conclusions on whether the topology exhibits properties similar to small-world or scale-free networks. These common properties provide insights on the degree of centralization and the sensitivity to random failures as well as targeted attacks.

7.1.1 *Data Collection and Methodology*

Over the span of two measurement periods (Oct.–Nov. 2018 and Jan.–Feb. 2019), we gathered information on the Lightning Network's topology. To this end, we used two virtual machines based on Ubuntu Server 18.04, which run `bitcoind` and `lnd`, respectively. We utilized the RPC

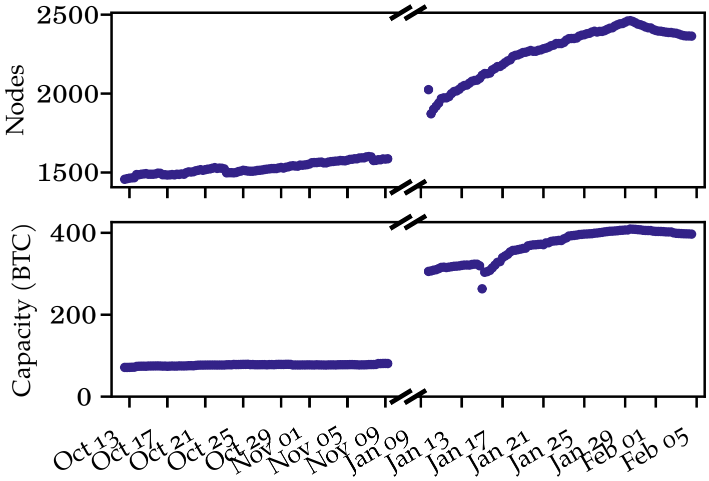


Figure 7.1: Time series showing the Lightning Network's number of nodes and its total capacity (i. e., sum of all payment channel capacities).

`describegraph` to retrieve the topology and regularly store snapshots in a `.json` file. For the data analysis we developed a python evaluation script, using the `networkx` [HSS08] and `powerlaw` [ABP14] libraries. Our code and all data sets are available online.[1]

In general, a node's view of the topology depends on the information it gets from its neighbors. Moreover, not all channels have to be announced publicly. Therefore, there is no guarantee to have a complete view of the network. However, we assume that the publicly available data characterizes the network's essential traits. While the network has immensely grown in terms of the number of nodes as well as in total capacity over the past months (see Figure 7.1), we can observe that the topology's characteristics have not changed significantly. The following analysis is based on a data set which was captured on Feb. 1, 2019 0:00 AM.

7.1.2 *Graph Measures and Metrics*

We consider the PCN topology as a graph $G = (V, E)$, where V is the set of nodes and E the set of edges, i. e., payment channels. The degree $deg(v)$ of a vertex v is defined as the number of its channels. A channel between node v_i and node v_j is denoted as e_{ij}. A path between two nodes consists of one or more channels. The distance between two nodes is defined as the shortest path between these nodes. The diameter is the longest distance between any two nodes in the network. Similar to the diameter, the average path length is defined as the average distance between any two nodes.

---

[1] `https://git.tu-berlin.de/rohrer/discharged-pc-data`

Table 7.1: Comparison of graph measures for different graph types.

	PCN	Scale-free	Random
Node count	2400	2400	2400
Edge count	13884	11975	13941
Diameter	6	5	6
Average distance	2.92	3.25	3.45
Central point dominance	0.16	0.09	0.005

Betweenness is a metric for centrality [Fre77a]. The betweenness of a node is the number of shortest paths between any two nodes in the network that pass through the node. The betweenness centrality $c_B(v)$ of a node v is given by

$$c_B(v) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(st)},$$

where $\sigma(st)$ is the number of all shortest paths between s and t, and $\sigma(s,t|v)$ is the number of all shortest paths between s and t that include v. To normalize the value such that $c_B \in [0,1]$, c_B is divided by the number of all pairs of nodes that do not include v, that is, $(n-1)(n-2)/2$ for undirected graphs with n being the total number of nodes. Accordingly, it describes the share of shortest paths that pass this node. In general, betweenness centrality is an indicator of how much control a node has over the network.

A subgraph $G' = (V', E')$ where $V' \subseteq V$ and $E' \subseteq E$ is called connected component if any node $v' \in V'$ can be reached by any other node in V'. A graph is biconnected if it is still connected after removing any arbitrary node. A *biconnected component* is a largest possible biconnected subgraph. If a node is member of more than one biconnected component, it is called an articulation point or cut vertex. As the removal of nodes that have a high betweenness and/or are articulation points may have an increased impact on network connectivity, they are potential targets of directed attacks.

Graph metrics typically gain meaning only in comparison to other graphs of the same size. In Table 7.1 we compare the PCN graph to a random Erdös-Renyi graph [ER59] and a scale-free Barabasi-Albert graph [BA99]. Note that while the graphs share similar parameters, the scale-free graph has less edges due to the method of preferential attachment.

From the comparison, we can observe that currently all three graph types share a diameter of 5–6 hops. At the same time, though, the PCN graph has the lowest average distance, which is a favorable property for users as it reduces the failure probability and saves routing fees. In terms of centrality, we compared the central point dominance, defined as the maximum betweenness centrality of all nodes. We can see that the PCN and the scale-free graph have a central point dominance more than ten times as high as the random graph's. The result suggests that the Lightning Network relies on few central nodes in order to process payments.

7.1.3 *Small-World Analysis*

Small-world networks are characterized by nodes that tend to cluster and have a high density of edges. More formally, the diameter grows logarithmically with the number of nodes. In order to test the "small-world-ness" of the Lightning Network, we use the method introduced in [HG08]. It is based on comparing the clustering coefficient to a clustering coefficient of a random graph with similar parameters, which serves as a reference.

While different definitions of the global clustering coefficient exist, we use the *transitivity* definition [Boc+06]. Accordingly, the clustering coefficient C is defined as

$$C = \frac{3 \cdot \text{number of triangles}}{\text{number of paths of length 2}}.$$

Note, a factor of 3 is used to compensate that each triangle has three paths of length 2. Thus, $C = 1$ for cliques.

Now, let L_g and L_r denote the mean shortest path length of the PCN graph G and a random Erdös-Renyi graph R, respectively. Likewise, let C_g and C_r describe the clustering coefficient. In accordance with [HG08], we consider a network as *small-world* if $S \gg 1$, where

$$S = \frac{\gamma_g}{\lambda_g} \quad \text{with} \quad \gamma_g = \frac{C_g}{C_r} \text{ and } \lambda_g = \frac{L_g}{L_r}.$$

Applying the described method to our empirical data yields $C_g = 0.085$ and $L_g = 2.92$ for the PCN graph and $C_r = 0.005$ and $L_r = 3.45$ for the random graph. We can already see that the Lightning Network is more clustered and yields on average shorter distances. We can conclude that the Lightning Network can be classified as a small-world network, as $S = 19.439 \gg 1$.

7.1.4 *Scale-Free Analysis*

Scale-free networks are characterized by a few nodes having a very high degree and many nodes having low degrees. More specifically, the degree distribution is similar to a power law distribution, where the fraction of nodes $P(k)$ having a degree k is described as $P(k) \sim k^{-\alpha}$ with α typically ranging between 2 and 3 [CSN09].

Scale-free networks emerge if a new node can choose its neighbors freely and prefers well-connected nodes. In the Lightning Network, we

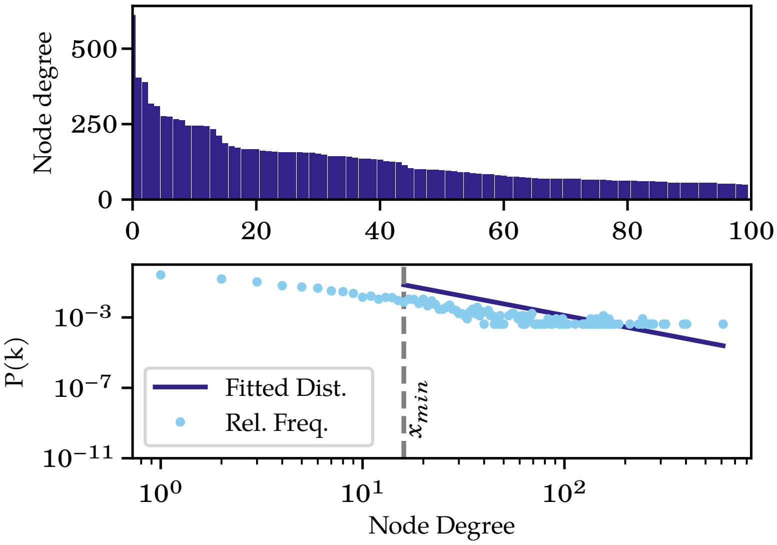


Figure 7.2: Degree distribution of the Lightning Network. The log-log plot additionally shows the fitted power-law distribution.

have a comparable situation. New nodes have an incentive to preferably open channels to highly connected nodes, hence reaching a larger share of the network with fewer hops. In comparison to random graphs, scale-free networks are generally robust to random failures, as the chance of a critical amount of high-degree nodes failing concurrently is very small. However, since a few nodes have high degrees, scale-free networks are prone to targeted attacks.

In Figure 7.2, we show the degree distribution of the 100 nodes with highest degree in the Lightning Network: the initial impression makes the hypothesis of a power-law distribution, i. e., scale-free network, plausible. To examine whether the Lightning Network is actually scale-free, we investigate the degree distribution along the lines of [CSN09]. The empirical data are plotted in a log-log plot in Figure 7.2. On the x-axis we show the node degree k and on the y-axis the probability for a certain node degree $P(k)$. For a power law distribution we expect a negative linear trend, where the slope determines the scaling factor α. However, this alone is not sufficient to draw conclusions. To get sound results, we additionally perform a power-law fit using a maximum likelihood estimator. More specifically, we use the Kolmogorov-Smirnov (KS) distance [Mas51] to determine the difference between the actual data and a proposed power-law fit. By minimizing the KS distance for x, we retrieve an x_{min}.

With α and x_{min}, we can derive a power-law distribution, but to draw conclusions it requires a goodness-of-fit test. Based on a number of synthetic data sets and respectively fitted distribution parameters (derived from the regression model), it generates a p-value, which we use to accept or reject the hypothesis of a power-law distribution.

Table 7.2: Average number of connected components after random failures for differen graph types.

Failures	Lightning Network	Scale-free	Random
1	1.13	1.00	1.00
2	1.20	1.00	1.00
3	1.61	1.00	1.00
5	2.86	1.00	1.01
10	3.66	1.00	1.00
50	14.00	1.00	1.03

The authors of [CSN09] suggest to use synthetic data sets with a high number of samples (ideally $10,000$ samples) and to reject the scale-free hypothesis if $p \leq 0.1$.

Applying this method to the empirical PCN data yields an $\alpha = 2.18$ and an $x_{min} = 16$. For the goodness-of-fit test we created $10,000$ data sets, that in sum yielded a p-value of $p = 0.3314$, clearly substantiating the scale-free hypothesis.

In conclusion, the degree distribution of the Lightning Network can be classified as power-law distributed, suggesting a scale-free network structure overall. Therefore, the network may benefit from the robustness property of scale-free networks against random failures.

7.1.5 *Robustness Analysis*

To draw conclusions about the robustness of the Lightning Network, we again compare it to other graph types (Table 7.2). We randomly removed a certain amount of nodes to simulate random failures. Each time, the simulation was run 100 times for the Lightning Network, a scale-free Barabasi-Albert graph [BA99], and a random Erdös-Renyi graph [ER59], respectively. The random removal of nodes has nearly no impact on the random graph and the scale-free graph but separates the PCN graph. The isolated components mostly consist of one and two nodes and therefore will barely affect routing efficiency. Yet, the graph separates and a random failure is very likely to separate at least one node from the network. In conclusion, the impact of random failures on the routing efficiency is very low. Nevertheless, the prospects of targeted attacks seems promising. Based on this insight, we discuss several attack vectors and their impact on the network in the following sections.

7.2 ATTACKING THE LIGHTNING NETWORK

As we have seen in the previous section, the Lightning Network is actually rather centralized and exhibits a heavily skewed degree distribution. This raises the question how the network topology copes with attacks that target specific points of interest. In the following, we present specific attack vectors, including channel exhaustion and node isolation attacks, and discuss a number of feasible attack strategies. Moreover, we introduce metrics that allow us to quantify the adversarial advantage of each strategy.

7.2.1 *Adversary Model*

With its growing success, the Lightning Network becomes an increasingly interesting target for different kinds of adversaries. We assume an active adversary that may participate in the peer-to-peer network, or attack its topology from the outside. At this point, we do not make any assumptions about the adversary's resources as this will be a parameter of our evaluation. In general, we assume however that the adversary is always eager to act as efficiently as possible, i. e., minimizing resources to maximize its adversarial advantage.

The adversary's motivation (or goal) may vary and therefore determines the attack vectors and strategies. For example, an adversary may be interested in eliminating single nodes, e. g., to impede or censor their participation in the network. She could also be interested in disrupting the network as a whole and aim for a partitioning attack that could impair the payment processing or even inhibit it entirely. Lastly, the adversary may be a "selfish" node in the Lightning Network, e. g., a payment hub, and interested in increasing her fee gain by sabotaging competing nodes and payment paths.

7.2.2 *Attack Vectors*

DENIAL OF SERVICE We consider denial-of-service (DoS) attacks as a general attack vector to disrupt a node's connection to the Lightning Network by using "external" means, i. e., not directly speaking the Lightning protocol. DoS attacks are typically mounted by flooding nodes with superfluous requests to overload their system. We however also include a broader range of DoS attack techniques, such as BGP hijacking to make nodes unreachable.

In general, a DoS attack on specific nodes in the Lightning Network allows an adversary to inhibit these nodes from partaking in regular payment processing. This attack vector usually requires a reasonably strong adversary controlling a botnet or having access to the Internet backbone. In March 2018, the Lightning Network was reportedly hit by a DDoS attack that took 20% of nodes offline [@Tru]. This incident shows that DoS is not a mere theoretical threat, but a feasible attack vector that has to be taken into account.

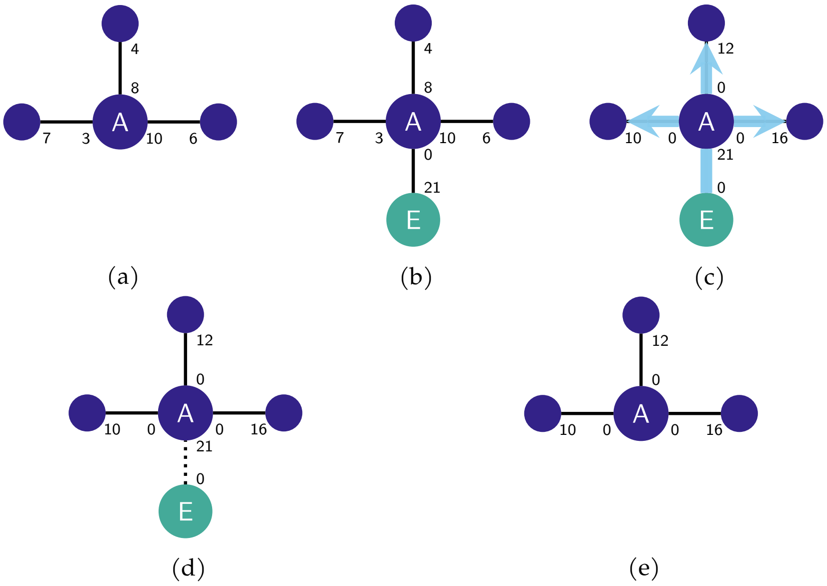
CHANNEL EXHAUSTION Each payment channel in the Lightning Network has a certain capacity and can therefore only route payments up to this capacity. We argue that this fact provides an attack vector: an attacker with sufficient funds is able to exhaust, i. e., block, a payment channel by routing a payment over the targeted channel in the respective direction. The attacker may not be able to infer the current channel balance as nodes only announce a channel's initial capacity. She therefore may need to route multiple payments to eventually exhaust a channel. To this end, the attacker could perform a binary search, starting with the maximum channel capacity and trying to route this volume. In order to not waste funds, the attacker is able to route funds back to herself, in which case only marginal routing fees accrue.

Using this technique, an adversary is able to disturb the payment flow in the network and manipulate it to its advantage. In particular, this attack may be used to cut parts off the network graph, leaving it in decomposed state.

PAYMENT GRIEFING The threat of channel exhaustion can currently be elevated by combining it with an attack vector called *payment griefing* [@Fou; Rob19]: as there is currently no fee on failed routing requests, the adversary may initiate an arbitrary number of HTLC payments to a node under her control. This node may then simply ignore incoming HTLC requests, forcing the involved nodes to wait for the time locks to expire. Upon expiry, the entire state is rolled back, circumventing fee deduction. Therefore, payment griefing allows an adversary to temporarily claim channel capacity free of charge.

Channel exhaustion in general and payment griefing in particular can be amplified by choosing longer payments paths. In this case, the adversary's stake is able to exhaust/lock funds along the path. With payment griefing, though, we can eliminate specific edges and paths in the PCN topology.

NODE ISOLATION By deliberately exhausting *all* channels of a node, we can isolate this node completely and effectively hinder it from participating. As shown in Figure 7.3, a malicious node E can zero all outbound channel balances of a target node A. The attack requires E to first open a channel with a capacity that is equal or greater than the total balance of A's outbound channels (cf. Figure 7.3b). Since the Lightning Network implements source routing, the attacker is able to determine routes and exhaust each channel by issuing a number of payments with respective payment volumes (cf. Figure 7.3c). Of course, to improve efficiency, the attacker can also make use of payment griefing.



(a) (b) (c)

(d) (e)

Figure 7.3: Node Isolation: the adversary E establishes a sufficiently large payment channel to the target node A. It then exhausts all outgoing capacity of A and closes her channel.

This attack vector can be considered a generalization of the channel exhaustion, described previously.

Node isolation leaves the target node unable to route outbound payments. While the node can of course still receive payment requests, it is unable to fulfill them because all of its funds have been shifted to the adversary's channel (cf. Figure 7.3d). In this situation, both sides can decide to close the channel, which would return the funds to the target node on-chain. Note that this would additionally deter the target node from using the funds for 1–2 block intervals, i. e., 10–20 minutes. The adversary however could also leave the channel open and refuse to process payment requests at which point the target node is forced to close the channel unilaterally. In this case, the settlement transaction can only be redeemed after the expiration a of the lock time, which adds a delay that is typically larger than 1–2 block intervals.

This attack vector effectively incapacitates the node from functioning as a payment hub, i. e., it eliminates the node from the routable network graph. In order to recover from this state, the target node needs to open at least one newly funded channel. As its old funds can only be reused after the closing channel was successful settled, it has to invest additional funds to be able to start rebalancing its channels and eventually regain its routing capabilities. Note that this comes with an additional overhead for the target node, since it has to pay fees for the funding transaction in the Bitcoin network.

To conclude, even by design, it is possible to remove edges and nodes from the routable Lightning Network. However, depending on the utilized attack vector, the adversary may have to provide more or less resources to carry out the attack.

7.2.3 *Attack Strategies*

Equipped with the means to remove edges and nodes, the adversary may try to cause maximum damage along the lines of the previously discussed adversary goals. Depending on these goal, however, she may choose a different strategy, i. e., a different set of nodes and edges to attack. In the following we discuss a number of attack strategies.

HIGHEST DEGREE/BETWEENNESS/EIGENVECTOR CENTRALITY NODES An adversary aiming to damage the network as a whole might try to destabilize or even partition the network graph, effectively impeding oder hindering cross-network payment routing. For this, the adversary might try to remove the participants according to their importance in the network. Promising strategies therefore prioritize central nodes with respect to some kind of centrality metric. Therefore, as an initial strategy, we propose that the attacker may target nodes based on their *degree* in descending order.

Moreover, the attacker may target nodes based on the previously mentioned betweenness centrality [Fre77a] or their eigenvector centrality [Gou67], which can consider not only the topological location, but also the edge capacities of a node.

HIGHEST RANKED MINIMUM CUT SETS A minimum cut set of a graph is a set of edges with minimal accumulated capacity that, when removed, partitions the graph. Therefore, minimum cuts are prime target when an adversary aims for network partitioning. However, not all cuts are created equal: while some may partition the network quite effectively, others may only cut off a single node. Given that the adversary has only limited resources at her disposal, it is important to prioritize the targeted minimum cuts according their importance in real world payment scenarios. Therefore, we propose to calculate a high number of potential (s, t)-cuts for randomly picked terminals s and t, and rank the individual cuts by the number of their occurrences. By targeting the highest-ranked cuts, the adversary focuses on the network bottlenecks hindering payment processing first.

HIGHEST RANKED PARALLEL PATHS An adversary that participates in the network as a payment hub may be interested in increasing her revenue by eliminating competitors. Of course, the adversary may again target competing hubs by their importance in the network, e. g., by node degree. However, such a strategy would not consider how payments are routed in the Lightning Network. Therefore, we propose to simulate random payments and record the resulting payment paths. Excluding the paths involving the adversary's hub, nodes can be ranked according to their involvement in the remaining payment paths. Accordingly, the adversary eliminates nodes that are part of many competing routes with the intention to increase her fees by processing more payments.

7.2.4 *Quantifying Adversarial Success*

In the following, we propose a number of metrics that allow us (and the adversary for that matter) to quantify the impact of an attack strategy. To provide an overall metric, we define the *adversary's advantage* of an adversary, i. e., success of the attack, as the *relative decrease* in the respective metric concerning the a priori measurement m and the a posteriori measurement m':

$$\Delta_m = \left|\frac{m - m'}{m}\right| = \left|1 - \frac{m'}{m}\right|.$$

The higher Δ_m becomes the higher the adversary's success according to metric m will be. It generally provides a way to relatively compare the prospect of different attack strategies from different perspectives.

An attacker may try to partition the network into a number of subgraphs. While the impact of such an attack is limited when the adversary is only excluding single nodes, it may be much more severe if she can cut off larger segments of the network. A sound measure for general network robustness should capture the share of nodes that are disconnected from the network graph. We therefore propose the *number of reachable nodes* r as a metric. Given all connected components C_i, we define the largest connected component C_1 to be *the* Lightning Network. Accordingly, we can calculate r as the network's node cardinality $r = |C_1|$. This metric can be used to calculate the adversarial advantage Δ_r as defined above.

However, as some nodes are more central and provide a larger share of the network's total capacity than others, the impact of node isolation on the liquidity of the network may vary heavily depending on the target. To quantify the impact, we propose the *average maximum flow* as another metric: for n rounds, we draw a pair of nodes $s_i, t_i \in V$, $i \in \{1 \dots n\}$ by uniform random sampling and calculate the maximum flow $F_i(s_i, t_i)$ along the lines of [FF56; GT88a]. The average maximum flow is then given by

$$\overline{F} = \frac{\sum\limits_{i=1}^{n} F_i(s_i, t_i)}{n}$$

and can be used to calculate the adversarial advantage Δ_F.

While the average maximum flow is a good indicator of the routable capacity in the network, it does not necessarily reflect the actual expected payment success, since currently the Lightning Network only uses single-path routing to fulfill payments. Therefore, we additionally introduce the *expected payment success ratio* s as a metric for how likely payments can be processed by the network. To get a sound estimation, we simulate a high number of transactions between random nodes and calculate the success ratio as

$$s = \frac{\#\text{successful payments}}{\#\text{attempts}}.$$

Accordingly, the adversarial advantage is given by Δ_s. As the validity of this measure heavily depends on the transaction model, it will especially benefit from parametrization based on empirical data.

In order to quantify the potential success of an internal adversary aiming for increased revenue, we propose to simulate a high number of payments and accumulate the fee gain $g_{i,h}t$ for the adversary's hub h_a that accrues over all simulated payments $i \in \{1 \dots n\}$. The *average fee gain*

$$\overline{g_h} = \frac{\sum_{i=1}^{n} g_{i,h_a}}{n}$$

may then be used to indicate the adversary's success.

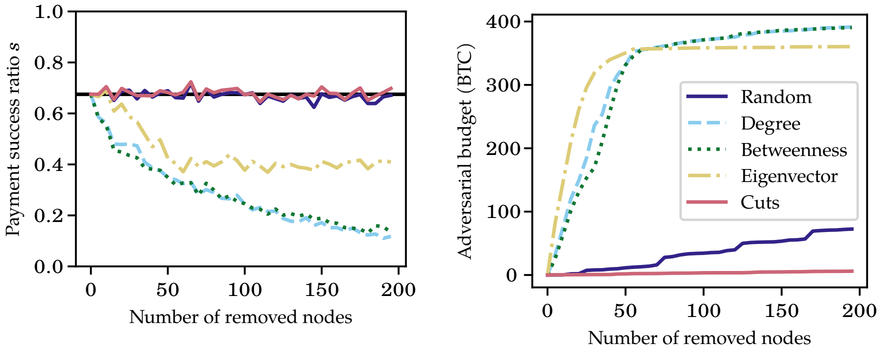
7.3 EVALUATION

7.3.1 *Proof of Concept*

In order to validate the feasibility of our node isolation attack, we built a simple toy scenario mimicking the attack shown in Figure 7.3. We ran five independent `lnd` instances, which were connected to the Bitcoin testnet. The target node A established three channels with outbound capacities set to 75,000, 100,000, and 125,000 satoshis, respectively. The attacker E established a channel with a total capacity of 400,000 satoshis to A, which is sufficient to exhaust A's channels and therefore hinder any other node from routing through A. In our example, we repeatedly sent payments of declining size until A was able to route no more than 100 satoshis (currently ≈ 0.0037 USD), at which point we considered the attack to be successful.

7.3.2 *Evaluation Model*

The following evaluation of topology-based attacks on the Lightning Network is based on simulations we implemented using the python library `networkx` [HSS08]. As before, the snapshot from Feb. 1, 2019 is used as our reference dataset. While the dataset provides real-world data on nodes, edges, and edge capacities, it does not include the actual channel balances. We therefore assumed the given capacities to be the



(a) Payment success ratio s before (horizontal line) and after the removal of n nodes.

(b) Required adversarial budget to remove n nodes by node isolation.

Figure 7.4: Node removal according to different attack strategies.

balance both ways, likely resulting in an overestimation of the routable funds, i. e., yielding a best-case estimation for the considered metrics.

The proposed metrics and attack strategies rely on the availability of a solid payment model that reflects how transactions of a certain volume traverse the network. Due to the lack of real-world transaction data of the Lightning Network we draw source and target nodes uniformly at random from the network nodes. By doing this, we refrain from introducing unnecessary complex and artificial assumptions. We also assume a single-path routing scheme as currently implemented by the Lightning Network: each payment is processed by first excluding all edges with insufficient capacities from the routable network graph. On the remaining graph, shortest path routing is performed.

As our data base for payment volumes, we collected real world payment data from the Ripple network [@Pro]. For this, we retrieved all XRP transactions that occurred at our reference date Feb. 1, 2019 and converted it to the respective values in satoshis. The transaction volumes are chosen by uniform random sampling from this data set.

All algorithms are repeated 1,000 times to ensure statistical significance. Furthermore, to ensure the reproducibility of the applied metrics, we opted to fixate the pseudorandom number generator's seed value for each round of simulation. Thereby, the same input data is utilized by all metrics, improving the comparability between measurements.

7.3.3 *Partitioning Attacks*

As we have seen, a capable adversary may isolate single nodes in the network. In the following, we analyze the Lightning Network's resilience to an adversary aiming for maximal damage to the network, i. e., network partitioning. To this end, we assume that the attacker is capable of removing a certain number of nodes from the routable network graph, e. g., by the means of DoS attacks or node isolation attacks. We

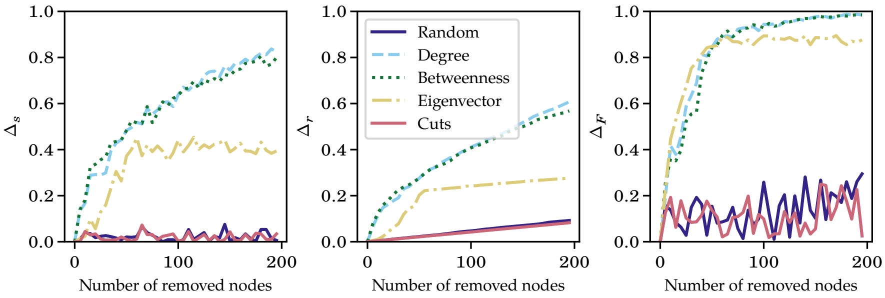


Figure 7.5: Adversary's advantage for success ratio, reachability, and average maximum flow after the removal of n nodes according to different attack strategies.

simulated the previously introduced attack strategies and recorded the network state before (a priori) and after (a posteriori) the attack: removing nodes by decreasing *degree*, by decreasing *betweenness* and *eigenvector* centrality, and by highest ranked minimum *cuts*. Moreover, we evaluated uniform *random* node removal as a baseline.

As an initial measure, Figure 7.4a shows the payment success ratio s before (horizontal line) and after the respective attacks happened. Notably, even before the attack only 675 out of the 1,000 tried payments succeeded. Moreover, we can see that the strategies work out quite differently, with degree-based removal and betweenness-based removal resulting in a steep decrease of the success ratio (down to around 10% success), while the random and minimum cut strategies measure around the baseline. Accordingly, the adversary's advantage for the success rate Δ_s is the highest for the former strategies, as shown in Figure 7.5. Similar results can be seen for the reachability, where the degree and betweenness strategies achieve Δ_r values of larger than 0.5, i. e., cutting off more than half of the network.

Moreover, the average maximum flow of the network is heavily impaired when removing central nodes. Here, the eigenvector strategy is superior until it is outperformed by the degree and betweenness strategies. The latter two eventually lead to a near-total collapse of the maximum flow, i. e., $\Delta_F \approx 1.0$, rendering the remaining network useless.

So far, targeting nodes based on their centrality seems to be the most promising strategy for an adversary that is capable of eliminating a certain number of nodes. However, as we see in Figure 7.4b, the centrality-based strategies also require the largest budget to successfully mount node isolation attacks, while the minimum cut strategy exhibits really low budget requirements. In the following, we therefore evaluate the efficiency of our attack strategies.

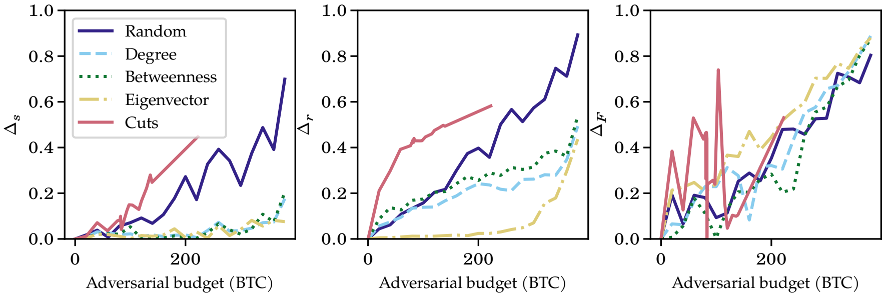


Figure 7.6: Attainable adversary's advantage depending on her budget for node isolation attacks.

7.3.4 *Efficiency of Node Isolation Attacks*

In order to evaluate the efficiency of the attack strategies, we assigned each adversary a budget and then analyzed the "damage" it can cause. In particular, we simulate a node isolation only, if the attacker's (remaining) budget is large enough to remove all edges. Otherwise, the simulation skips the node and tries to utilize the available funds on the next target proposed by the respective strategy. Likewise, we only remove complete cuts. As a consequence, the minimum cut strategy does not always consume the full budget.

The results are shown in Figure 7.6: independently of the strategy, we observe that it requires high budgets to give the adversary the power to reliably disturb all payment attempts, which would result in a high advantage score Δ_s. Notably, the previously underperforming minimum cuts and random node removal strategies exhibit the best efficiency properties, e. g., an adversary could attain an advantage of $\Delta_s = 0.4$ when spending around 200 BTC. Similar behaviour can be seen for the impact on the adversary's advantage in terms of the reachability Δ_r and average maximum flow Δ_F. While some strategies seem to be subject to fluctuations, which do not always allow to infer a clear ordering in efficiency, the highest ranked minimum cut strategy again clearly stands out as the most efficient in terms of the Δ_r and Δ_F, exhibiting values above 0.5 in both cases. This is not surprising, since the maximum cut strategy targets the connecting edges and nodes first, whose removal has a significant impact on graph connectivity and the available network capacity.

7.3.5 *Fee Gain*

In contrast to disrupting the network, an adversary might be interested in increasing its own profit by strategically eliminating competing nodes. We assumed that this adversary is an established payment hub in the network (amongst the top 10 nodes ranked by total capacity). According to our strategies, most notable highest ranked degree and highest ranked parallel paths, we eliminated up to 30 nodes. We used the average fee gain as metric to quantify the adversary's success. While the adversary can indeed profit from eliminating nodes, we cannot observe a clear trend. In fact, we believe that node-based elimination strategies are too coarsely grained. Instead, we argue that an channel-based elimination strategy might be superior, which we intend to investigate in the future.

7.3.6 *Discussion*

In our analysis, we assume that each node functions as a payment hub, i. e., accepts incoming payment channels. Moreover, the attacker is assumed to find an adequate endpoint with sufficient capacity to route payments. This is especially important if the attacker does not execute payment griefing attacks, but needs to route payments back to herself. In this case, the attacker has to ensure that some nodes in the network first establish channels of sufficient volume to her secondary node, i. e., her target node is connected with high enough inbound capacity.

In order to mitigate the possibility of node isolation attacks, the client software should employ rate limiting techniques to limit the number of incoming channels and incoming channel volume. This would make it harder for an adversary to quickly establish high-volume channels from an advantageous position in the topology. However, a client probably cannot mitigate the risk of node isolation attacks entirely, since the attacker may circumvent simple rate limiting strategies by splitting the funds over multiple identities and channels.

Moreover, network partitioning attacks may be counteracted by the so-called *autopilot* algorithms responsible for automated payment channel creation. This may be achieved by monitoring previously discussed metrics and restructuring the topology accordingly to make it less susceptible to targeted attacks.

8

EXPLORING MULTI-PATH ROUTE SELECTION FOR PAYMENT CHANNEL NETWORKS

This chapter is based on the publication: Elias Rohrer, Jann-Frederik Lass, and Florian Tschorsch. "Towards a Concurrent and Distributed Route Selection for Payment Channel Networks." In: CBT '17: Proceedings of the 1st International Workshop on Cryptocurrencies and Blockchain Technology. Sept. 2017 [RLT17]. The locking extension to the push-relabel algorithm should be attributed to fellow co-authors.

In order to facilitate multi-hop payment routing, the Lightning Network pursues a source routing paradigm in which payment senders individually select routes with reference to a global information base. However, as we have seen so far, the lack of coordination may lead to a variety of detrimental effects, such as network congestion and payment failures. In particular the fact that currently only a single path is used to route payments limits payments' sizes and the probability of success.

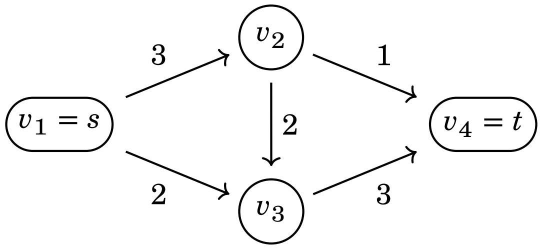
To this end, we now study how much efficiency could be gained by utilizing a multi-path routing approach in which payments are considered network flows. Moreover, we identify that routing multiple concurrent flows is reducible to the multi-commodity flow problem and explore the possibility of a distributed routing algorithm that takes multiple routing demands into account.

8.1 PAYMENTS AS FLOWS

In the following, we model PCNs as *flow networks* and consider payments flows. Such a *payment flow* describes a flow of units between pairs of nodes in a payment channel network. Figure 8.1 shows a mock example of a payment channel network in which node s wants to send a payment to node t. We consider the payment channel network as a peer-to-peer network in which nodes communicate directly with each other and build an overlay network congruent with the PCN topology.

In order to process the payment, a path between s and t must exist. Every path is a concatenation of payment channels. Since payment channels have a capacity, as indicated by the edge labeling in Figure 8.1, a path's transaction volume is limited by the smallest payment channel capacity of this path. While we cannot eliminate this limit, we can use multiple paths, which in sum provide a higher transaction volume.

Determining the maximum transferable amount poses a challenge. For example, simply finding all paths from source to sink and summing up their respective capacities does not suffice; paths may have common edges and thus need to share the respective capacities. For the example in Figure 8.1, this naive approach would violate payment channel capacities.



Path	**Vol.**
$s \to v_2 \to t$	1
$s \to v_2 \to v_3 \to t$	2
$s \to v_3 \to t$	2
max. flow	4

Figure 8.1: Payment channel network example.

8.1.1 *The Push-Relabel Algorithm*

The problem of finding the largest payment flow between two nodes s and t in a capacitated flow network is known as the *maximum-flow problem*. Several algorithmic solutions to the maximum-flow problem exist. In the following, we elaborate on the efficient and well-studied *push-relabel* [GT88b] algorithm and adopt it for the route selection of payment flows in payment channel networks. The push-relabel algorithm is based on the notion of *pre-flows*, which are pushed from node to node through the network. The algorithm assigns a *height* to every node and tries to exhaust the capacities of outgoing edges towards smaller heights. If capacities of a node's outgoing edges are exhausted and it still holds excess flow, the overflow is pushed back towards the source. The overflow may then be used by other candidate paths.

Intuitively, this can be thought of as a flow of water running through a network of reservoirs of varying height that are connected by pipes with different capacities. Water entering the network at the source will flow downhill towards the sink as far as it can and thereby saturate the pipes' capacities in the process. Overflowing water will be held back in the reservoirs and it may even flow back towards the source, if it is indeed positioned lower. When the water flow stops, we can lift up a reservoir, until the water starts flowing downhill again. By repeating this process, we will eventually distribute the water flow to the pipe system in a way that maximizes the water throughput. For a further description of the push-relabel algorithm, additional terminology is needed.

We consider a network of payment channels as a directed graph $G = (V, E)$ and a non-negative function $c : V \times V \to \mathbb{R}_{\geq 0}$. We call c the capacity function, which determines a channel's capacity $c(u, v)$ with $u, v \in V$ and $(u, v) \in E$. Moreover, nodes s and t are the source and sink of the flow. The resulting network $F = (G, c, s, t)$ is called a *flow network*.

Definition 1 (pseudo-flow, pre-flow, feasible flow)**.** *A **pseudo-flow** on the capacitated graph* (G, c) *is a mapping* $f : V \times V \to \mathbb{R}$ *with the properties:*

$$
\begin{aligned}
&f(u,v) \leq c(u,v),\ \forall (u,v) \in E && \textit{(capacity constraint)}\\
&f(u,v) = -f(v,u),\ \forall (u,v) \in E && \textit{(skew symmetry)}
\end{aligned}
$$

*Note that pseudo-flows do not require incoming and outgoing flows of a node to be equal. Therefore, nodes can hold **excess flow**, denoted by*

$$x_f(u) = \sum_{v \in V} f(v,u) - \sum_{v \in V} f(u,v).$$

*A pre-flow and a feasible flow are special kinds of pseudo-flows with one of the following constraints. A **pre-flow** requires*

$$x_f(v) \geq 0,\ \forall v \in V\ \{s,t\} \qquad \textit{(non-negativity constraint)}$$

*and a **feasible flow** requires*

$$x_f(v) = 0,\ \forall v \in V\ \{s,t\} \qquad \textit{(conservation constraint)}.$$

Definition 2 (residual capacity and residual graph)**.** *The **residual capacity** c_f with regard to the pseudo-flow f of an edge* $(u,v) \in E$ *is defined as the difference between the edge's capacity and its flow:*

$$c_f(u,v) = c(u,v) - f(u,v).$$

*Then, the **residual graph** $G_f(V, E_f)$ indicates when changes can be made to flow f in the network* $G(V,E)$*, where*

$$E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}.$$

Note that edges (u,v) *do not have to be in the original set of edges* E*.*

Definition 3 (height function)**.** *A mapping* $h : V \to \mathbb{N}$ *is a **height function** for the push-relabel algorithm, if*

$$h(s) = |V|, \qquad h(t) = 0, \qquad h(u) \leq h(v) + 1,\ \forall (u,v) \in E_f.$$

At the beginning, the generic push-relabel algorithm initializes node heights and flow excess, as well as the edge pre-flow values with 0. Please note that source node s, in contrast to all other nodes, is set to a height $|V|$. Moreover, s's outgoing edges are saturated according to the height function's third condition. After these initialization steps, the algorithm repeatedly selects a node u as active node and applies one of the two basic operations `push` and `relabel`. Both operations have mutually exclusive conditions, which ensure that either `push` or `relabel` is applicable at a time.

The `push` procedure (cf. Algorithm 2) tries to push an excess δ from node u towards a neighbor v with a smaller height. The maximum possible δ is determined as the minimum between the excess flow and

Algorithm 2 `push(u,v)`

```
Ensure: $x_f(u) > 0, c(u,v) > 0, h(u) = h(v) + 1$
  $\delta := \min( x_f(u),\ c_f(u,v) )$
  $f(u,v) := f(u,v) + \delta;\ f(v,u) := f(v,u) - \delta$
  $x_f(u) := x_f(u) - \delta;\ x_f(v) := x_f(v) + \delta$
```

Algorithm 3 `relabel(u)`

```
Ensure: $x_f(u) > 0, \forall (u,v) \in E : h(u) \leq h(v)$
  $h(u) := 1 + \min ( h(v) : (u,v) \in E )$
```

the residual capacity of edge (u,v). Accordingly, edge capacities and excess values are updated to reflect flow changes in the residual graph. The procedure requires that u has excess flow and that an unsaturated edge (u,v) to a neighbor v one level below u exists.

Eventually, node u will saturate all outgoing edges that lead to neighbors on a lower level. In this case, the `relabel` procedure (cf. Algorithm 3) “raises” node u to a higher level. The procedure calculates the minimal height of its neighbor nodes and sets u’s height to the level above this minimum. Therefore, the excess of node u is guaranteed to be “pushable” in the next step.

The generic push-relabel algorithms continues until the conditions fail for all nodes. That means, the highest possible transaction volume has been pushed to the sink t and all network excess has been pushed back to the source, i. e., $x_f(v) = 0,\ \forall v \in V$. At this point, the push-relabel algorithm has transformed the pre-flow into a maximum flow and hence solved the maximum-flow problem.

8.1.2 *Feasible Flows*

In a payment channel network, however, it is often not necessary to know the maximum transaction volume. Rather, we want to find a payment flow that can process a certain amount only. This is a slightly different problem, which is known as the *feasible-flow problem*. Fortunately, the push-relabel can easily be modified to solve the feasible-flow problem: in order to find a payment flow from source s to sink t with a transaction volume d, we can simply insert a new (virtual) node to the payment network. We call it the *pre-source* s', with a single edge (s',s) and capacity $c(s',s) = d$. The virtual edge caps the transferable amount at exactly d. Note that this slight modification of the input data enables the push-relabel algorithm, as described before, to find feasible flows in the network.

So far, we assumed only one instance of the push-relabel algorithm. If multiple flows ought to be found subsequently in the same network, the initial flow of one instance is the result of the last instance. A generalization for subsequent flows, however, is easily possible. This subsequent approach can be used to find payment flows in a centralized or federated fashion. The following section is dedicated to show how the push-relabel algorithm can be adapted to enable route selection for concurrent and distributed payment flows.

8.2 concurrent and distributed payment flows

The push-relabel algorithm can find a single feasible flow in a payment channel network. However, in a real-world payment network, there may be multiple concurrent flows that need to be considered during route selection at any given time. To this end, simply running multiple instances of the push-relabel algorithm in parallel is not enough: one instance for flow f_1, for example, could consume the reverse edges' residual capacity that belong to another instance for flow f_2. We call this issue *capacity stealing*. Capacity stealing may ultimately prevent f_2's instance from pushing back its flow excess to the source, which violates the termination criteria ($x_f(v) = 0,\ \forall v \in V\ \ \{a,b\}$). Thus, the traditional push-relabel algorithm does not (and does not intend to) guarantee correctness and termination in the face of concurrent flows.

The problem domain of finding flows $f_1, \dots, f_k$ for k commodities with source-sink pairs $(s_1, t_1), \dots, (s_k, t_k)$ that meet the total capacity constraint

$$F(u,v) = \sum_{i=1}^{k} f_i(u,v) \leq c(u,v),\ \forall (u,v) \in E,$$

are known as *multi-commodity flow problems.*

8.2.1 *Capacity Locking*

As our main contribution, we propose a modified push-relabel algorithm that allows to find feasible flows in a concurrent multi-commodity scenario. To this end, we introduce the concept of *capacity locking*: flow volumes are accounted for every commodity independently, while still respecting each payment channel's total capacity constraint. The capacities on the reverse edges created by a flow f_1 are therefore *locked* for another flow f_2, which prevents capacity stealing.

Definition 4 (locked capacities and new residual capacity)**.** *Let the* ***locked capacity*** *and* ***total locked capacity*** *of flow f_i on edge (u,v) be*

$$l_i(u,v) = max(0, f_i(u,v)) \qquad and \qquad L(u,v) = \sum_{i=1}^{k} l_i(u,v).$$

Accordingly, the ***residual capacity*** *is redefined as*

$$c_i(u,v) = c(u,v) - L(u,v) + l_i(v,u),$$

which yields an individual residual graph $G_i(V, E_i)$ for each commodity i.

Algorithm 4 `locked-push(i,u,v)`

```
Ensure: $x_i(u) > 0, c_i(u,v) > 0, h_i(u) > h_i(v)$
  $l_i(u,v) := \max(0, f_i(u,v)); l_i(v,u) := \max(0, f_i(v,u))$
  $c_i(u,v) := c(u,v) - L(u,v) + l_i(v,u)$
  $\delta := \min(x_i(u), c_i(u,v))$
  $f_i(u,v) := f_i(u,v) + \delta; f_i(v,u) := f_i(v,u) - \delta$
  $L(u,v) := L(u,v) + \delta; L(v,u) := L(v,u) - \delta$
  $x_i(u) := x_i(u) - \delta;\ x_i(v) := x_i(v) + \delta$
```

Definition 4 ensures that there is always enough residual capacity on the reverse edges available to push the existing excess back to the source. Except for this augmented definition of the residual capacity, the `locked-push` procedure (cf. Algorithm 4) is similar to the original `push` procedure. Note, however, that the modified push-relabel algorithm does not necessarily yield optimal flows in the multi-commodity scenario. It guarantees *validity*, though, which makes it superior compared to other approaches from this domain [AL94].

In the following, we prove validity for our proposed algorithm. As the skew-symmetry and flow-conservation constraints follow directly from the definition of the algorithm, it suffices to show that it yields flows that respect the total capacity constraint.

Lemma 1. *The total capacity constraint* $F(u,v) \leq c(u,v),\ \forall (u,v) \in E$ *is never violated.*

Proof. For a `locked-push` of commodity i on edge (u,v), the change in flow volume δ is always chosen to be at maximum the remaining residual capacity of the flow on this edge. Accordingly, lock $l_i(u,v)$ cannot be greater than δ. Therefore, the locked capacity never exceeds the edge capacity for each individual edge. It follows that the total capacity constraint is never violated:

$$\begin{aligned} F(u,v) &= \sum_{i=1}^{k} f_i(u,v) \\ &\leq L(u,v) = \sum_{i=1}^{k} l_i(u,v) \\ &\leq \sum_{i=1}^{k} c_i(u,v) \\ &\leq c(u,v). \end{aligned}$$

□

8.2.2 *Towards Distributed Route Selection*

In order to execute the modified algorithm in a distributed scenario, the asynchronous distributed algorithm, introduced in [GT88b], is

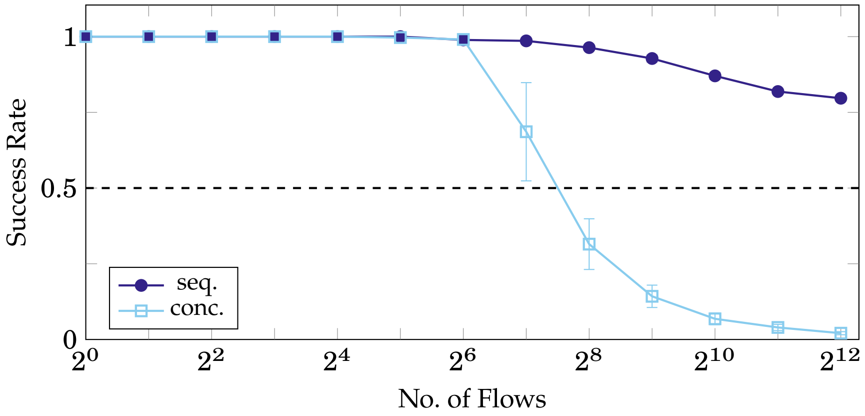


Figure 8.2: Mean success rate over 10 runs in dependence of the number of flows. Error bars show the 95% confidence interval.

adapted to our needs: each node maintains a local view on flow states, channel capacities, and its neighbors' height. Furthermore, each node maintains routing information and its own height. Then, every node u with positive excess tries to push its excess along an unsaturated outgoing edge to a neighbor v of smaller height. A `locked-push` can only be committed, if v acknowledges u that it is has indeed a smaller height. Alternatively, v can reject the `locked-push` and respond with its actual height. This way, u learns its neighbors' height and can trigger `relabel`, if necessary. After relabeling, u sends height updates to its neighbors. The source and sink node can determine the termination of the algorithm and communicate the result to finalize route selection. The payment flow, i. e., the selected multi path, is secured with Hashed Timelock Contracts (HTLC) in the same way as a single path. Therefore, the payment flow can be atomically resolved.

8.3 EVALUATION

In order to evaluate our approach in a controlled and reproducible environment, we construct a Watts-Strogatz graph [WS98] with $\beta = 0.5$, $n = 200$, and a node degree of 10. Channel capacities are generated by uniform random sampling from $[0, 10]$. In the following, we compare the sequential (*seq.*, cf. Section 8.1) and the concurrent (*conc.*, cf. Section 8.2) algorithm. Moreover, we contrast our results with the capabilities of single-path routing approaches.

8.3.1 *Payment Success*

First, we are interested in the number of flows that each algorithm can handle. To this end, we sampled the transaction volume from $[0, 20]$ and calculated the mean success rate over 10 runs, i. e., the share of successfully found flows. The results, shown in Figure 8.2, indicate that

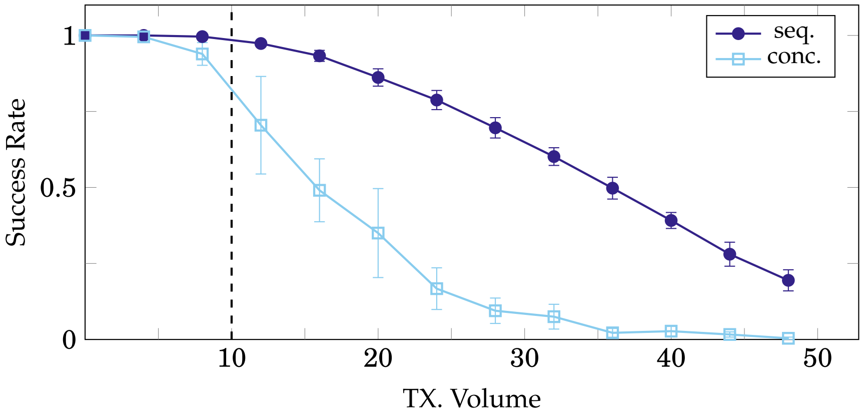


Figure 8.3: Mean success rate over 10 runs in dependence of transaction volume. Error bars show the 95% confidence interval.

both algorithms are able to find a large number of flows (relative to the network size). At some point, when network capacities are exhausted, the success rate eventually drops. However, The concurrent execution and the concomitant locked capacities clearly lead to an earlier exhaustion of the capacities and thus to an earlier drop in the success rate.

Single-path approaches, in contrast, achieve in the best case a 0.5 success rate (cf. horizontal line in the plot): while the maximum channel capacity is 10, on average every second transaction volume is in $(10, 20]$ and therefore not feasible with a single path. Effectively, this reduces the utilization of the available capacities by 50%.

8.3.2 *Routable Volume*

Second, we are interested in the transaction volume that we can achieve by aggregating multiple paths. To this end, we set the number of flows to 128, increased the transaction volume, and calculated the mean success rate. The results, in Figure 8.3, suggest that again both variations are able to route relatively large volumes. In more than 50% of the cases, the concurrent algorithm still manages to process all 128 flows for up to a volume of 15 each. This is especially noteworthy, as a single-path approach would not be able to route a single payment with a volume exceeding 10 in our scenario (cf. vertical line in the plot).

In summary, these first results illustrate that single-path source routing approach currently deployed by PCNs is detrimental to the network's efficiency and severely limits the capacity of transactions, as well as their success probability. To this end, our approach emphasizes the need for a multi-path routing algorithm with improved payment coordination.

9

TIMING ATTACKS ON PRIVACY IN PAYMENT CHANNEL NETWORKS

This chapter is based on the publication: Elias Rohrer and Florian Tschorsch. "Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks." In: AFT '20: Proceedings of the second ACM conference on Advances in Financial Technologies. Oct. 2020 [RT20].

As previously discussed, payment channel networks enable rapid payment processing between any two channel endpoints, without consulting the blockchain every time. To this end, incremental channel updates are negotiated locally instead of requiring a global agreement, which is not only much faster, but also does not leak information to noninvolved third parties. In an attempt to furthermore avoid unnecessary information leakage to intermediate nodes, all messages exchanged during the processing of multi-hop payments are encrypted according to an onion routing scheme; more specifically, the Sphinx mix packet construction is applied [DG09].

In the following, we however show that the privacy guarantees of the Lightning Network may be subverted by an adversary conducting timing attacks on the message exchange during payment processing. In particular, an on-path adversary may reduce the anonymity set of potential sender and receiver nodes based on the payment amount and the HTLC's time-lock delta value. Following this initial reduction of privacy, the adversary may apply timing-based estimators to infer the likeliest payment path, potentially deanonymizing the sender and receiver of a payment.

9.1 MODEL

In the following, we introduce the models and notations that serve as the basis for the further analysis in this chapter.

9.1.1 *Network Model*

As discussed in Chapter 6, PCNs typically exhibit multiple layers: while inter-peer communication is handled by the peer-to-peer network layer, the payments themselves are sent and forwarded in the network of payment channels. While peers may join the peer-to-peer network without establishing payment channels, peers with an established payment channel have to be connected in the peer-to-peer network. We in the following assume the peer-to-peer network to be congruent with the channel layer and build a unified model based on the public network of payment channels.

A PCN can therefore be modeled as a single graph $G = (\mathscr{V}, \mathscr{E}, \phi)$, where $\mathscr{V} = \{v_0, \dots, v_n\}$ is the set of the network's nodes and $\mathscr{E} = \{e_0, \dots, e_m\}$ represent the set of edges, i. e., payment channels. Since every node may have multiple payment channels to any other node, G is a loopless multigraph and $\phi : \mathcal{E} \to \{\{u,v\} \mid u,v \in \mathcal{V} \wedge u \neq v\}$ associates the set of edges with their endpoint nodes.

In each direction, an edge e is associated with a *balance* $\mathbf{bal}(e,u,v)$ that denotes the available balance from u to v on channel e, where $u,v \in \phi(e)$. Edges also have an associated *fee function*, defined by $\mathbf{fee}(a|e,u,v)$ that takes the payment amount a as a parameter and yields the fees that accrue when forwarding over this channel. Note that during routing directionality matters and hence balance and fee functions are asymmetric. That is, generally $\mathbf{bal}(e,u,v) \neq \mathbf{bal}(e,v,u)$ and $\mathbf{fee}(a|e,u,v) \neq \mathbf{fee}(a|e,v,u)$. In contrast, the edge *capacity* is symmetric and defined as the sum of balances, i. e., $\mathbf{cap}(e) = \mathbf{bal}(e,u,v) + \mathbf{bal}(e,v,u)$. Furthermore, associated with the edges are the respective *time-lock delta* values $\Delta_{tl}(e,u,v)$ that indicate the maximum time in block height the forwarding node is willing to have its funds locked in case an HTLC fails. And lastly, the function $\mathbf{lat}(e)$ assigns a *latency distribution* to each network edge that represents the network and processing delays, which are induced when messages traverse the edge in the underlying peer-to-peer network. Note that while balances and latencies are changing frequently and are only known locally, fees, capacities, and time-lock requirements are considered static and are publicly accessible by all nodes in order to enable the source routing process. We denote this public graph information as $G_{pub} = (G, \mathbf{cap}, \mathbf{fee}, \Delta_{tl})$.

Based on this model of payment channel networks, we can now introduce the following definitions.

Definition 5 (Payment)**.** *A* payment *is defined as the tuple*

$$(s,t,a,\Delta_{max}),$$

where s and t respectively denote the origin and destination nodes, a is the amount sent by s, and Δ_{max} signifies the maximal total time the payment amount may be locked.

Definition 6 (Path Validity)**.** *A* path *from node s to node t in the network is a sequence of connecting edges, denoted as a tuple*

$$p = (e_0, e_1, \dots, e_l),$$

where $\phi(e_0) = \{s, v_1\}, \phi(e_1) = \{v_1, v_2\}, \dots, \phi(e_l) = \{v_l, t\}$.

A path p is called timelock-valid *w.r.t. a total time-lock delta Δ_{max}, if the remaining time the payment might be locked is smaller than the time the forwarding node would be willing to accept, i. e.,*

$$\forall e_i \in p,\ u_i, v_i \in \phi(e_i) : \Delta_{tl}(e_i, u_i, v_i) \geq \Delta_{max} - \sum_{j=0}^{i-1} \Delta_{tl}(e_j, u_j, v_j)$$

Furthermore, a path p is capacity-valid *w.r.t. an amount a, if all edges have capacities higher than the forwarding amount including accruing fee, i. e.,*

$$\forall e_i \in p,\ u_i, v_i \in \phi(e_i) : \mathbf{cap}(e_i) \geq \mathbf{f}_i,$$

where $\mathbf{f}_i$ *is defined recursively as* $\mathbf{f}_l = a$ *and* $\mathbf{f}_{i-1} = \mathbf{f_i} + \mathbf{fee}(\mathbf{f}_i|e_i, u_i, v_i)$*. Note that a path* p *being capacity-valid for amount* a *does not necessarily imply that a payment of size* a *can actually be routed over* p*.*

A payment can be routed only if the path does not only exhibit sufficient capacities, but also balances to forward the respective amount,

$$\forall e_i \in p,\ u_i, v_i \in \phi(e_i)\ :\ \mathbf{bal}(e_i, u_i, v_i) \geq \mathbf{f}_i,$$

in which case we call it balance-valid *or* a-routable*.*

Independently of this special case, however, we call a payment path generally valid *or just* valid*, if it is timelock-valid* and *capacity valid. Note that therefore path validity describes if a path could potentially be used to route a payment with respect to the parameters, not if it actually may be used or is being used.*

Definition 7 (Routing Algorithm)**.** *A* routing algorithm $\mathfrak{R}$ *is a function that takes a payment* $x = (s, t, a, \Delta_{max})$ *and the public graph information* $G_{pub} = (G, \mathbf{cap}, \mathbf{fee}, \Delta_{tl})$ *as arguments and outputs a valid payment path, i. e.,*

$$\mathfrak{R}(x|G_{pub}) \rightarrow p,$$

where p *is a capacity-valid and timelock-valid path from* s *to* t*.*

Definition 8 (Reachability)**.** *In the graph* G*, a node* t *is* reachable *from a node* t *if there is a path between them. Similarly, we call* t capacity-reachable,balance-reachable*, or* timelock-reachable *from* s*, if there exists respectively a capacity-valid, balance-valid, or timelock-valid path from* s *to* t*, w.r.t. a given payment* (s, t, a, Δ_{max})*.*

Note that these reachability notions induce subgraphs $\mathcal{R}_{\mathbf{cap}}$*,* $\mathcal{R}_{\mathbf{bal}}$*, and* $\mathcal{R}_{\Delta}$*, where*

$$G \supseteq \mathcal{R}_{\mathbf{cap}} \supset \mathcal{R}_{\mathbf{bal}} \quad \text{and} \quad G \supseteq \mathcal{R}_{\Delta}.$$

9.1.2 *Adversary Model*

9.1.2.1 *Lightning's Security Goals*

Given payments are routed directly between source and destination nodes, are secured by the HTLC construction, and the path is obscured by employing the Sphinx-based onion routing scheme, the Lightning Network aims to deliver the following security goals (cf. [Mal+17b]):

BALANCE SECURITY: No third party should be able to steal funds, or otherwise alter channel balances without the implicit consent of the involved parties.

OFF-PATH LOCAL UNOBSERVABILITY: Only nodes on the payment path should be informed about an occurring payment.[1]

---

[1] Note however that this assumes a local perspective on the network. Payment unobservability may not hold when we assume a more powerful attacker model, such as an

OFF-PATH VALUE PRIVACY: Since all communications during payment processing are encrypted, only on-path nodes should get to know the amounts forwarded.

ON-PATH SENDER/RECEIVER-ANONYMITY: Every node only knows its immediate predecessor and successor on the payment path. That is, it does not know the payment path's length and its position on the payment path and therefore should not be able to identify the sender or receiver of a payment.

RECEIVER'S SENDER-ANONYMITY: The receiver of a payment should not be able to identify who initiated a payment. [2]

9.1.2.2 *Adversarial Goals and Capabilities*

While the off-path unobservability of payments could potentially be subject of a deanonymization attack run by a global passive adversary, in this work we analyze the feasibility of subverting the on-path anonymity properties of nodes that send and receive payments. In particular, we focus on attack vectors that allow a local adversary incorporating side-channel information to potentially subvert *on-path sender/receiver anonymity* as well as *receiver's sender-anonymity*.

To this end, we assume an internal local adversary that controls a set $M = \{m_0, \dots, m_k\}$ of malicious nodes in the network that act as payment-processing intermediaries, which in accordance with literature may also be referred to as *spies*. We furthermore assume that the adversarial nodes M behave according to protocol and are able to send and receive protocol-compatible messages, e. g., in order to probe the network to build a latency model of their surroundings.

When payments are routed over an adversarial node $m_i \in M$, it keeps track of each network message `msg` arriving over the edge e_m, as well as the corresponding timestamp, i. e., they store the datasets

$$\mathscr{D}_i = \{(m_i, e_m, t_{\texttt{msg}}, \texttt{msg})\}.$$

Based on the merged dataset $\mathscr{D} = \bigcup_i \mathscr{D}_i$, the public graph data G_{pub}, and the estimated link latencies $\widehat{\textbf{lat}}$, the adversary then aims to associate any observed payments $x = (s, t, a, \Delta_{max})$ with the respective source node s and destination node t. For this classification, the adversary may apply different *source and destination estimators* $\mathfrak{M}_s$ and $\mathfrak{M}_t$ that given the input data yield a respective estimation, i. e.,

$$\mathfrak{M}_s(x|\mathscr{D}, G_{pub}, \widehat{\textbf{lat}}) = \hat{v}_s \quad \text{and} \quad \mathfrak{M}_t(x|\mathscr{D}, G_{pub}, \widehat{\textbf{lat}}) = \hat{v}_t,$$

---

adversary that has access to large parts of the underlying network infrastructure. Such adversaries are known to be potentially capable of advanced deanonymization attacks, and are notoriously hard to defeat. [Joh+13]

[2] Notably, the Lightning Network currently does not guarantee the inverse, i. e., the possibility for a receiver to stay anonymous. However, this may feasible in the future, when the currently discussed *Rendez-Vous Routing* proposal is implemented. [@Dec20]

where $\hat{v}_s, \hat{v}_t \in \mathscr{V}$. For the sake of brevity, we in the following refrain from always giving an exhaustive list of arguments and opt to abbreviate notation as $\mathfrak{M}_s(x|\mathscr{D})$ and $\mathfrak{M}_t(x|\mathscr{D})$.

9.1.3 *Anonymity Metrics*

In order to quantify adversarial success and analyze the privacy properties of the network, we utilize the following privacy metrics.

Well known performance measures for the adversarial success of estimator-based deanonymization attacks are the combination of *precision* and *recall* [Fan+18]. Assuming X is the set of all payments and $C \subseteq X$ the set of all payments observed and classified by the adversary. Let furthermore $X_u \subseteq X$ denote the set of all payments that originate from (or end at, in case of destination estimation) node u and analogously C_u denote the set of payments classified to originate from (end at) node u, i. e.,

$$C_u = \{x \mid \mathfrak{M}(x|\mathscr{D}) = u\}.$$

Then the *precision* D of the estimator $\mathscr{V}$ is defined as the share of classified payments that were indeed correctly classified, i. e.,

$$D = \frac{|C_u \cap X_u|}{|C|}.$$

The estimator's *recall* R however is the share of all payments in the network that were correctly classified,

$$R = \frac{|C_u \cap X_u|}{|X|}.$$

A unified measure for the accuracy of an estimator is given by the harmonic mean of precision and recall, also known as the F_1-measure:

$$F_1 = 2 \cdot \frac{D \cdot R}{D + R}.$$

9.2 TIMING ATTACKS ON PRIVACY

In the following, we describe the steps necessary to conduct timing attacks on privacy in payment channel networks.

9.2.1 *Improving Topological Advantage*

The attacker wants to maximize the number of payment paths it is included in by the victim's routing algorithm. While the client-side routing behavior is not standardized as part of Lightning's BOLT specifications, most implementations of the Lightning protocol rely on modified versions of Dijkstra's shortest path algorithm [Dij59] that consider the channel fees as well as other parameters. Note that, as recent literature has observed [MZ20], more than 90% of today's Lightning Network nodes run the `LND` implementation of the Lightning protocol. In the following, we therefore assume `LND` to be the default implementation and use it as the base of our further analysis. `LND`'s path finding algorithm selects candidate edges based on a weight function w_e that considers routing fees for the routed amount a, as well as a risk factor r_f that aims to capture the worst-case lock time:

$$w_e = \mathbf{fee}(v|a,e,u) + a \cdot \Delta_{tl}(e,u,v) \cdot r_f,$$

where $r_f = 1.5 \cdot 10^{-8}$ is the default configuration.

Therefore, by setting time lock parameters and channel fees to the minimal allowed values, the adversary can minimize the routing weight function of the victim's client software, and thereby maximize the probability that at least one of the malicious nodes is included in the payment path. Tochner et al. [TSZ19] studied this kind of *route hijacking* in the context of denial-of-service attacks. They showed that, at the time of writing, ten nodes are part of 80% of all payment paths, and 30 nodes of over 95% of payment paths. Furthermore, while the problem of optimal edge additions for maximum betweenness centrality has previously been show to be NP-hard [Ava+20; Ber+18a; ERE20a], the authors provide a greedy algorithm with which an adversary improve its topological advantage. To this end, they were able to show that the creation of only fifteen edges would suffice to hijack more than 80% of `LND` payment paths. Their observations are generally in accordance with our findings regarding adversarial path inclusion (cf. Section 9.3.3.3) and highlight the relevance of the on-path attacker model.

9.2.2 *Building the Latency Model*

As a data basis for the classification of observed payments, the adversary initially has to probe the network to retrieve characteristic timing measurements. These measurements allow her to build a model of latencies $\widehat{\mathbf{lat}}$ that are encountered when payments are routed over a specific link, which then in turn are used as a priori knowledge for the estimators.

9.2.2.1 *Retrieving Path Latency Measurements*

In order to probe for the characteristic latency measurements, the adversary can exploit the fact that due to Lightning's use of the Sphinx packet format, invalid or failing payments can only be discovered by the node that they are actually failing at. That is, as all nodes only see the parts of onion-routed data they are able to encrypt and do not know the full path's properties, they have to optimistically forward all payment requests based on the assumption that it will succeed. Therefore, the adversary is able to craft payments that look valid to all intermediaries, but are bound to fail at a specific hop along the path, e. g., because of insufficient fees or an invalid maximum time-lock value. Utilizing this probing method, the adversary can record the time difference between sending the initial `update_add_htlc` message and retrieving the final `update_fail_htlc` to retrieve a measurement that encompasses all delays that were encountered along the measured payment path.

9.2.2.2 *Estimating Edge Latencies*

The adversarial node utilizes the described probing method to retrieve a reliable model for paths covering every link in the network. To this end, she iteratively increases the probing path lengths and calculates link latencies by subtracting the estimated latencies of partial paths. That is, the adversarial node m_i starts by repeatedly probing paths lengths $l = 1$ that cover its immediate neighbors v_j, i. e., $p_1 = (e_1), m_i, v_j \in \phi(e_1)$, and calculates the mean μ_{e_1} and standard deviation σ_{e_1} values for these links, i. e.,

$$\widehat{\mu}_{e_1} = \frac{\sum_{i=0}^{n} \mathtt{probe}_i(p_1)}{T \cdot n},$$

$$\widehat{\sigma}_{e_1} = \sqrt{\frac{\sum_{i=0}^{n} (\mathtt{probe}_i(p_1) - \widehat{\mu}_{e_1})^2}{T \cdot n}},$$

where T is a normalizing factor accounting for the number of link traversals incurred during the message exchange over the measured hop. In this case, we assume $T = 4$, i. e., three traversals for the `commitment_signed` and `revoke_and_ack` handshake and one for `update_fulfill_htlc` (see Figure 6.2).

The adversary can then increase the path lengths and iteratively build the latency model for these longer paths $p_l = (e_1, \dots, e_l)$:

$$\widehat{\mu}_{e_l} = \frac{\sum_{i=0}^{n} \mathtt{probe}_i(p_l)}{T \cdot n} - \widehat{\mu}_{e_{l-1}} - \dots - \widehat{\mu}_{e_1},$$

$$\widehat{\sigma}_{e_l} = \sqrt{\frac{\sum_{i=0}^{n} (\mathtt{probe}_i(p_l) - \widehat{\mu}_{e_1})^2}{T \cdot n} + \dots + \widehat{\sigma}_{e_{l-1}}^2 + \widehat{\sigma}_{e_l}^2}.$$

Given these parameters, the adversary can build the normally distributed edge latency model as

$$\widehat{\mathbf{lat}}(e_i) = \mathcal{N}(\widehat{\mu}_{e_i}, \widehat{\sigma}_{e_i}^2).$$

Note that this model does not just include the network delay, but also incorporates any processing delays arising on the intermediate nodes. As this unified latency model captures various side effects, we can refrain from considering them separately in the attack estimators.

Moreover, modeling timing behavior in such an approximative way is bound to induce a certain margin of error. This uncertainty is expressed

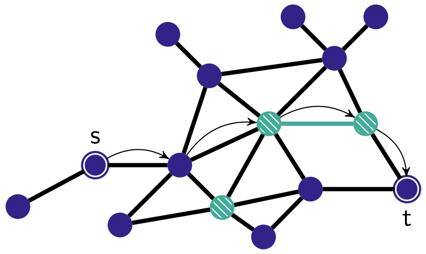


Figure 9.1: Payment routed over malicious observation points.

by the variances growing with increasing lengths of the measured paths. In the following we therefore propose a method to aggregate the timing models from multiple malicious vantage points to increase overall accuracy.

9.2.2.3 *Model Aggregation*

As the adversary may control multiple nodes in the network to increase the probability of inclusion in payment paths, each malicious node may create a timing model from their point of view. As the margin of error increases with each additional hop in the measured paths, the aggregated model should not simply average over all measurements. Instead, it merges the individual model by applying an arithmetic mean weighted with the reciprocal distance from the measured node, i. e.,

$$\forall m_i \in M, v_i \in V : w_i = \frac{1}{d(v_i, m_i)},$$

$$\widehat{\mu}_{e,tot} = \frac{\sum_{i=0}^{n\cdot|M|} w_i \widehat{\mu}_{i,e}}{\sum_{i=0}^{n\cdot|M|} w_i},$$

$$\widehat{\sigma}_{e,tot} = \sqrt{\frac{\sum_{i=0}^{n\cdot|M|} w_i (\widehat{\mu}_{i,e} - \widehat{\mu}_{e,tot})^2}{\sum_{i=0}^{n\cdot|M|} w_i}}.$$

The adversary therefore retrieves the aggregated latency model

$$\widehat{\mathbf{lat}}_{tot}(e_i) = \mathcal{N}(\widehat{\mu}_{e,tot}, \widehat{\sigma}^2_{e,tot}).$$

9.2.3 *Estimator-based Deanonymization Attack*

In order to be able to deanonymize the sender and receiver of a payment, it has to be routed over at least one observation point controlled by the adversary (see Figure 9.1). In contrast to previous approaches that apply a *First-Spy* estimator that simply estimates the node adjacent to the point of observation to be the payment's respective endpoint, our approach builds a maximum likelihood estimator (MLE) over all paths the observed payment could possibly have taken. To this end, the malicious nodes record the time differences of interactive message exchanges and, after reducing the candidate set by considering only valid paths (see Definition 6), estimate the source or destination of the payment according to the likelihood that the time differences stem from a message exchange over this particular path.

9.2.3.1 *Recording Time Differences*

To utilize the timing of messages in order to estimate the source or destination of a payment, the adversarial nodes M have to observe end-to-end transmitted messages belonging to the same payment at two different points in time, i. e., t_0 and t_1. This allows to calculate the time difference $\delta_t = t_1 - t_0$ it took the observed payment to travel from the first point of observation $m_0 \in M$ to the source or destination, and back to the second point of observation $m_1 \in M$.

In particular, the malicious intermediate nodes record the point in time t_0 when they forward a payment via `update_add_htlc` and t_1 upon receipt of the corresponding `update_fulfill_htlc`, which yields a time difference δ_t corresponding to the distance to the payment's destination (cf. Figure 6.2). In this case, the adversary does not have to interfere with the payment processing protocol in order to collect the necessary information to conduct destination estimation. Hence, the adversary acts in a purely honest-but-curious model, and therefore cannot be detected by outside parties.

However, since the message exchange from the source node to the intermediate node is non-interactive, an advanced attack strategy is required in order to retrieve suitable timing measurements in this direction. To this end, the adversary intentionally fails the first observed payment attempt by sending an `update_fail_htlc` message. She also records the current time as t_0. After receiving the failure message, the payment's sender is forced to retry the failed attempt, which is typically done immediately to avoid further delays. When the second payment attempt is observed at time t_1, the adversary can calculate the $\delta_t = t_1 - t_0$ value which corresponds to its distance from the source node.

In general, the chosen paths and points of observation may be different, in which case the adversary has only a certain chance of observing the second payment attempt. While this would introduce additional uncertainty to this part of the adversarial strategy, as we discuss later in Section 9.3.2, the adversary may force a sender to send the second payment attempt over the same path as before, which removes this uncertainty. This is possible in practice due to an implementation detail of `LND`.

We therefore in the following assume the two observations to occur at the same malicious node, i. e., $m_0 = m_1$. Moreover, in the case that multiple malicious nodes are part of the payment path and observe the payment, the source or destination estimation is based on the mea-

Algorithm 5 Source / Destination Estimator

```
function ESTIMATE($\delta_t, a_{obs}, \Delta_{obs}, e_{obs}, G_{pub}$)
    remove all capacity-invalid paths from $G_{pub}$
    remove all timelock-invalid paths from $G_{pub}$
    for all $v \in V$ do
        initialize
    end for
    $v_{fst} \leftarrow$ GET_NEIGHBOR($e_{obs}$)                          ▷ First hop is known
    QUEUE_CANDIDATE($v_{fst}$)
    $\widehat{\textbf{lat}}_{fst} \leftarrow T_{obs} \cdot \widehat{\textbf{lat}}(e_{obs})$
    path_lats[$v_{fst}$] $\leftarrow \{\widehat{\textbf{lat}}(e_{obs})\}$
    likelihood[$v_{fst}$] $\leftarrow \widehat{\textbf{lat}}_{fst}(\delta_t)$
    while $v_{cur} \leftarrow$ NEXT_UNVISITED do
        $D_{cur} \leftarrow$ path_lats[$v_{cur}$]
        $\widehat{\textbf{lat}}_{cur} \leftarrow \sum_{d_i \in D_{cur}} T_i \cdot d_i$                ▷ Aggregate dists.
        $p_{cur} \leftarrow \widehat{\textbf{lat}}_{cur}(\delta_t)$
        for all $v_n$ in NEIGHBORS($v_{cur}$) do
            $e_n \leftarrow$ CHEAPEST_EDGE($v_{cur}, v_n$)
            $D_n \leftarrow D_{cur} \cup \{\widehat{\textbf{lat}}(e_n)\}$
            $\widehat{\textbf{lat}}_n \leftarrow \sum_{d_i \in D_n} T_i \cdot d_i$
            $p_n \leftarrow \widehat{\textbf{lat}}_n(\delta_t)$
            $p_{old} \leftarrow likelihood[v_n]$
            if $p_n \leq p_{cur}$ or $p_n \leq p_{old}$ then             ▷ Only increasing likelihood
                skip
            end if
            likelihood[$v_n$] $\leftarrow p_n$                         ▷ Update candidate
            path_lats[$v_n$] $\leftarrow D_n$
            QUEUE_CANDIDATE($v$)
        end for
    end while
    for all visited $v$ do
        return candidate with max. likelihood
    end for
end function
```

surements recorded by the malicious node closest to the respective endpoint.

9.2.3.2 *Source and Destination Estimation*

The estimation of source and destination of a payment relies on selecting the likeliest paths the payment could have taken before it arrived at the observation points. Therefore, in order to reduce the initial uncertainty, the adversary excludes paths that are capacity-unreachable or time-lock unreachable given the observed amount a_{obs} and Δ_{obs}, i. e., she only considers nodes in

$$\mathscr{R}_{\textbf{cap}} \cap \mathscr{R}_{\Delta} \subseteq G_{pub}.$$

The adversary then builds candidate aggregated latency distributions

$$\widehat{\textbf{lat}}_p = T_{obs} \cdot \widehat{\textbf{lat}}(e_{obs}) + \ldots + T_l \cdot \widehat{\textbf{lat}}(e_l)$$

for each candidate path $p = (e_{obs}, ..., e_l)$, where e_{obs} denotes the edge the measurement was conducted through. Furthermore, the weights T_i denote the number of messages that would have been exchanged over the edge e_i. Note that the possibility of such an aggregation relies on the fact that the sum of normally distributed variables may be calculated as

$$\mathcal{N}(\mu_1, \sigma_1^2) + \mathcal{N}(\mu_2, \sigma_2^2) = \mathcal{N}(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2).$$

Then, the adversary ranks all candidate paths according to the likelihood that the observed time difference δ_t was drawn from the respective aggregated distribution, $\max_p(\widehat{\mathbf{lat}}_p(\delta_t))$, and estimates the final hop of the path to be the payment's source or destination.

Therefore, the adversary generally would have to rank all possible paths in the network. However, Algorithm 5 implements the estimators $\mathfrak{M}_s$ / $\mathfrak{M}_t$ as an iterative algorithm that traverses the graph starting from the point of observation. During execution, it adds new candidate paths as long as they would result in an increased likelihood of observing δ_t, and stops when all candidate paths have been visited.

9.3 EVALUATION

In the following, we evaluate the feasibility, accuracy, and reliability of the presented attacks on privacy in payment channel networks.

9.3.1 *Ethical Considerations*

Research on the security and privacy of live communication systems is always in danger of infringing on the rights of the participating individuals. In accordance with the Menlo Report [Bai+12], we aim to minimize our interference with the live network as well as the data collected from unknowing parties.

That is, in order to evaluate the presented attacks on privacy in payment channel networks, we pursue a two-pronged strategy: First we show the feasibility of the attacks through a proof-of-concept implementation that was installed on an entirely segregated part of the Lightning Network `testnet`, which ensures that no involuntary parties were affected by our experiments.

Second, to be able to evaluate larger attack scenarios and analyze the effect these attacks have on the network's privacy overall, we rely on model-based network simulation that is not connected in any way to unknowing individuals and hence does not raise any ethical concerns. In particular, while the simulations utilize latency measurements that were retrieved through *external* means, i. e., ICMP `ping` on nodes from the public internet, we explicitly refrain from conducting *internal* latency measurements as discussed in Section 9.2.2, since such measurements could interfere with the functionality of the network.

9.3.2 *Proof-of-Concept Implementation*

The described attacks on payment privacy in payment channel networks rely on the ability of malicious intermediary nodes to retrieve latency measurements from the source and to the destination of an observed payment. As a proof of concept that obtaining these measurements is indeed practical, we implemented a respective plugin for `c-lightning` [@c-l20].[3]

RETRIEVING LATENCIES TO DESTINATION When the plugin is started on the intermediary node, it registers to be notified of forwarded payments. In particular, it utilizes the `forward_event` notification to record the times t_0 HTLC payment hashes $H(r)$ are first observed, as well as the times t_1 they are marked as resolved. The plugin furthermore records the node identifier v_M of the measurement node and the identifier e_{next} of the channel the payment was forwarded over. That is, it records the tuple $(H(r), u_M, e_{next}, t_0, t_1)$ that is then ready to be used as input a payment destination estimator.

RETRIEVING LATENCIES FROM SOURCE Because the communication with the payment source is non-interactive, the adversary has to rely on observing retried payment attempts, as discussed above. To this end, the proof-of-concept implementation makes use of the `htlc_accepted` hook provided `c-lightning`'s plugin API in order to intercept incoming `update_add_htlc` messages. When a payment with a previously unobserved payment hash $H(r)$ is observed, the plugin records a corresponding timestamp t_0 and rejects the payment attempt. As the payment is is then retried, the timestamp t_1 of the second observation by an adversarial intermediary node is recorded. This hence allows the adversary to estimate the latency from the source and record the tuple $(H(r), u_M, e_{prev}, t_0, t_1)$.

While it is not guaranteed to observe the second payment, the proof-of-concept implementation is able to force the payment source to reuse the same payment path by exploiting a weakness in the interplay of Lightning's network protocol and `LND`-specific application behavior. That is, as channel updates may occur in the middle of a payment attempt, `LND` elects not to penalize intermediary nodes during route selection, if they report a channel policy failure, i. e., fail the payment with the failure codes `amount_below_minimum`, `fee_insufficient`, `incorrect_cltv_expiry`, or `channel_disabled` [@Dev20a; @LND20b]. Note that `LND` once a minute grants such nodes a "second chance", independently of whether the returned channel policies entail an actually meaningful update [@LND20a]. This allows our plugin to fail the first

---

[3] Proof-of-concept and simulator source codes, as well as utilized data sets, are publicly available in our companion repository at `https://git.tu-berlin.de/rohrer/cdt-data`.

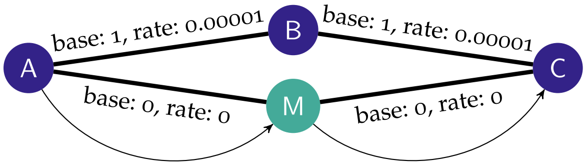


Figure 9.2: Experimental Testnet Setup

observed payment attempt with a corresponding `update_fail_htlc` failure message, which prompts the `LND` endpoint to immediately retry the payment over the same malicious intermediary node, effectively enabling reliable latency measurements.[4]

EXPERIMENTAL TESTNET SETUP In order to confirm the feasibility of retrieving the required time differences, we deployed an experimental setup on a segregated part of Lightning's `testnet` network. As shown in Figure 9.2, we deployed three nodes A, B, C running `LND` and one malicious node M running `c-lightning` with our proof-of-concept plugin. Between these nodes, channels were created so that the source node A would have two possible paths to send payments to destination node C: one over the benign node B, and one over M. While the channels between the benign nodes were configured with default fee settings (`base_fee` $= 1$ and `fee_rate` $= 0.00001$), the malicious M set its channel fees to 0 to increase its probability of payment path inclusion.

We then sent payments in one minute intervals from node A to node C. For all payments, node A chose the path (A, M, C), which proves M's strategy to be successful. Moreover, as discussed above, M would in each case reject the first payment attempt and only proceed on the second try, allowing it to retrieve latency measurements for both source node A as well as destination node C. It therefore confirms that we can retrieve the time differences that pose the basis for our timing attacks. As the next step, we can use the measurements to feed our estimators, which would infer source and destination.

9.3.3 *Network Simulations*

9.3.3.1 *Measuring Lightning's Peer-to-Peer Network*

In order to attain a reliable model for inter-peer connections, we in the following examine Lightning's public peer-to-peer network. To this end, we acquired a snapshot[5] of the network graph taken on March

---

[4] Note that as of this writing, a small change in the `c-lightning` source code is necessary to enable a plugin to return failure codes entailing a channel policy update. A corresponding patch can be found in our companion repository.

[5] https://git.tu-berlin.de/rohrer/discharged-pc-data/blob/master/snapshots/lngraph_2020_03_26__00_00.json.zst

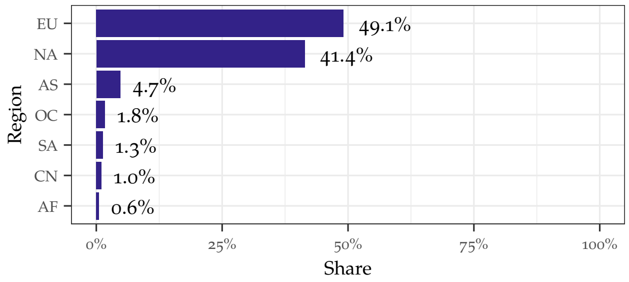


Figure 9.3: Geographical peer distribution of the Lightning Network

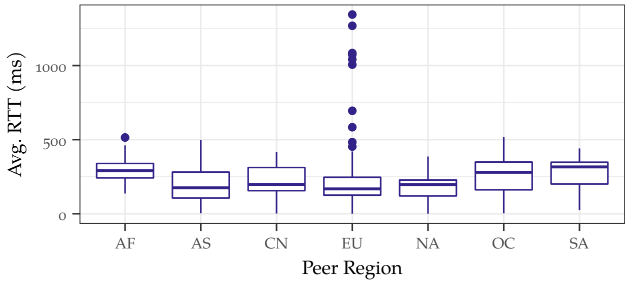


Figure 9.4: Latency distribution of Lightning's peer-to-peer network

26, 2020 00:00 UTC, extracted the $2,679$ public IPv4 addresses, and categorized them in regional clusters based on GeoLite2 [@Max21] geographic location database. As shown in Figure 9.3, the peer-to-peer network spans seven regions of the globe: Europe (EU), North America (NA), Asia (AS), Oceania (OC), South America (SA), China (CN), and Africa (AF). The data however also shows that the network is currently clearly dominated by the EU and NA regional clusters, which is in accordance with the regional distribution of Bitcoin's peer-to-peer network [@Bit21b].

Based on this data, we setup a measurement study to infer a suitable latency model for Lightning's peer-to-peer network. For this, we deployed seven measurement nodes as close as possible to the aforementioned regional clusters, i. e., in the following Amazon AWS regions: `us-west-1` (NA), `sa-east-1` (SA), `eu-central-1` (EU), `ap-southeast-2` (OC), `ap-south-1` (AS), `me-south-1` (AF), and `ap-east-1` (CN). After initialization, each measurement node starts collecting ICMP `ping` results to each of the public Lightning IP addresses. In particular, each measurement would send 100 `ping` requests to each Lightning node for 100 times, which allows to build a more reliable round-trip time (RTT) model by averaging over the results. Of the $2,679$ addresses, we found $1,297$ peers to be offline or not reachable via ICMP, which corresponds to around 48% of the network. The regional latency distribution for the remaining peers is shown in Figure 9.4: while there are some regional differences and outliers, the inter-peer latencies almost all fall below the 500ms mark, with the global median being located around 250ms.

9.3.3.2 *Simulator and Simulation Model*

In order to enable a larger-scale evaluation of the feasibility and impact of timing attacks on privacy, we developed a network simulator that allows to simulate payment routing in the Lightning Network based on real-world data.[3]

The simulator consists of around 3,000 lines of Rust code that implement the network model introduced in Section 9.1, as well as the logic to run time-discrete simulations of multi-hop payments. To this end, it recreates the multigraph of network nodes and edges as well as the necessary associated data (such as capacities, balances, time-lock deltas, etc.) from a network snapshot. Each node can queue events in simulation time, i. e., a monotonically increasing clock with a resolution of 1 ns. This allows to simulate message exchange according to times sampled from the underlying latency model, without introducing unnecessary side-effects, even when the events happen concurrently. The messaging logic mimics the Lightning payment protocol, making it possible to simulate and measure time differences in the message exchange, e. g., as depicted in Figure 6.2. In order to find payment paths, the simulator adopts the weight-based variant of Dijkstra's algorithm from the `LND` implementation (see Section 9.2.1).

The latency model is based on the measurement study presented in Section 9.3.3.1. While initializing the graph model, the simulator assigns a latency distribution to each edge based on the respective geographic regions of the connected nodes. In case a node only advertises a `.onion` address, i. e., is run behind a Tor hidden service, a random geographic location is assigned.

For the following analysis, the simulator was parametrized with a snapshot of the Lightning Network that was retrieved on May 1, 2020. Initial balance distributions between channel endpoints were assumed to be a 50/50 split of channel capacities. If not stated otherwise, in each simulated scenario 1,000 payments of varying amounts were sent between random network nodes, and each scenario was repeated 30 times with different seed values for the simulator's random number generator to ensure stastistical significance.

In the following, we are considering three main adversarial scenarios: `mcentral`, `mrandom`, and `lnbig`. While in the `mcentral` case the m highest ranked nodes with respect to their *betweenness centrality* are under control of the adversary, `mrandom` acts as a baseline in which she only controls m nodes chosen by uniform random sampling. A special case is the `lnbig` scenario, in which we study the potential capabilities of the 26 high-capacity nodes controlled by the single entity "LNBIG.com".

9.3.3.3 *Share of Compromised Paths*

In order to evaluate the relevance of the on-path adversary model, we analyze how likely it is that payments may be observed by adversaries of

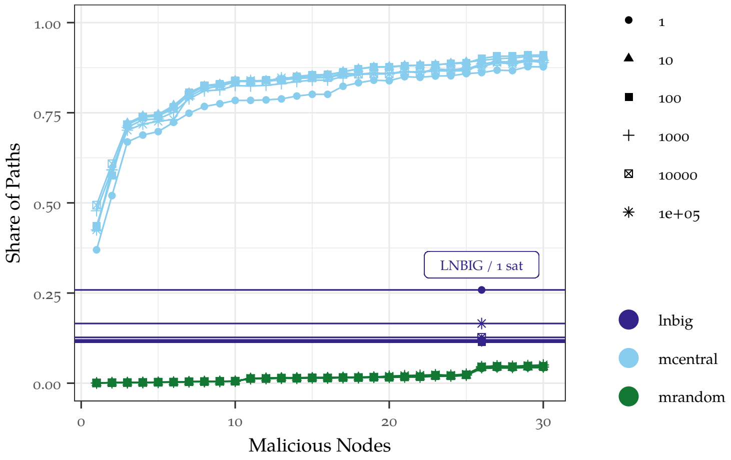


Figure 9.5: Share of compromised payment paths.

different magnitudes. In each network scenario, we simulated payments of different amounts (1, 10, 100, 1,000, 10,000, and 100,000 satoshis) between randomly chosen nodes and counted the times a malicious node was part of the path returned by the routing algorithm.

Figure 9.5 shows the share of compromised paths for network scenarios in which the adversary controls the $m \in \{1, .., 30\}$ most central or random nodes, as well as for the `lnbig` scenario. As this corresponds to the definition of betweenness centrality, it comes to no surprise that the most central nodes observe a high and increasing number of paths. However, it is noteworthy that the single most central node is included in 37% to around 49% of payment paths, depending on the chosen amount. Additionally, the share of compromised paths follows an initial steep increase, allowing an adversary in control of the four most central nodes to already observe an average of 72%, and one in control of 30 most central nodes to be included in 90% of payment paths.

In contrast, an adversary controlling randomly placed nodes may at best observe an average of 5% of payments. Moreover, an adversary controlling the 26 `lnbig` nodes can observe between 11% and 25% of payments, averaging at 15%.

Generally, the payments with the highest amount result in the highest shares of compromised paths. This is most likely the case since more central nodes tend to optimize their fee policies and are also well-connected capacity wise, i. e., are more likely part of the few paths that can route higher-amount payments. However, one exception to this rule can be observed in the case of `lnbig`, where the 1 satoshi case yields the highest chance of path inclusion at 25%. We assume this to be the case because of LNBIG's positioning in the network and since their nodes feature a high amount of channels with `base_fee` set to 0, making them more likely to be chose by the routing algorithm for low-amount payments.

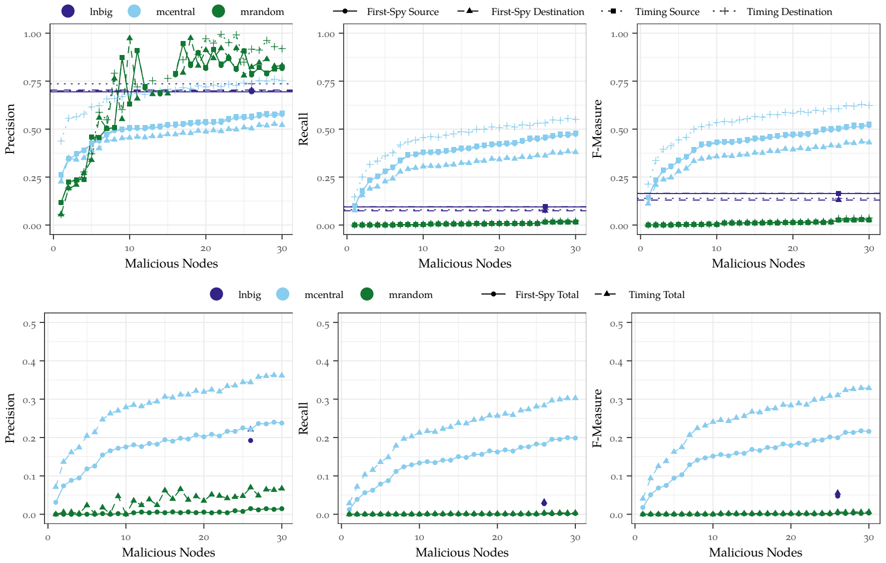


Figure 9.6: Adversarial success in dependence of number of malicious nodes.

9.3.3.4 *Adversarial Success*

In the case of an adversary aiming to deanonymize payments, the performance with which she can correctly guess the source or destination of a message is also a measure of (remaining) user privacy. We therefore analyze how successful adversaries of different magnitudes would be if they would run the proposed timing-based attacks by applying the source estimator $\mathfrak{M}_{s,\mathsf{T}}$ and destination estimator $\mathfrak{M}_{t,\mathsf{T}}$. As a baseline for comparison, we also implemented and simulated the First-Spy estimators $\mathfrak{M}_{s,\mathsf{FS}}$ and $\mathfrak{M}_{t,\mathsf{FS}}$ that respectively deem the predecessor of the first point of observation and the successor of the last point of observation to be source and destination of the observed payment.

The upper row of Figure 9.6 shows the success of the different estimators in dependence of the evaluated scenarios and number of malicious nodes. As can be seen in the top left plot, the precision with which the adversary estimates the correct sources or destinations is generally correlated with the number of controlled malicious nodes. In case of the `mcentral` scenario, the precision of each estimator roughly follows a logarithmic growth function, where the First-Spy destination estimator $\mathfrak{M}_{t,\mathsf{FS}}$ performs the worst ranging from 0.22 for a single controlled node to 0.52 for 30 malicious nodes. In contrast, the timing-based destination estimator $\mathfrak{M}_{t,\mathsf{T}}$ yields the highest accuracy that ranges from 0.45 to 0.75. Comparably, a potential adversary controlling the 26 `lnbig` nodes would be able to correctly identify senders or receivers with a precision ranging from 0.69 for $\mathfrak{M}_{s,\mathsf{FS}}$ to 0.73 for $\mathfrak{M}_{t,\mathsf{T}}$. These high estimation results can be attributed to the favorable positioning of LNBIG's nodes in the network topology. In similar vein, it can be observed that randomly placed malicious nodes, as in the `mrandom` scenrio, may actually guess the correct senders and receivers with quite high accuracy, as they cover the network graph more uniformly.

However, as shown in the top-middle plot of Figure 9.6, `lnbig` and `mrandom` nodes clearly do not perform as well in terms of recall. While the share of correctly attributed payments barely reaches 2% in the best case ($\mathfrak{M}_{t,\mathsf{T}}$, $m = 30$), `lnbig` is also only able to estimate the correct endpoints in 10% of all payments at best. Notably, the most central nodes have the highest recall, ranging between 7% ($\mathfrak{M}_{t,\mathsf{FS}}$, $m = 1$) and 55% of payments. Of course, the recall is highly correlated with the share of observed payments paths discussed in the previous subsection.

Therefore, in order to provide a unified measure that allows to analyze and compare the overall accuracy of estimators, the top-right plot of Figure 9.6 shows the corresponding F_1-Measure. In all cases, it shows that the First-Spy baseline is outperformed by the timing-based estimators, which reach up to an F_1 score of 0.62 for `mcentral`, $\mathfrak{M}_{t,\mathsf{T}}$, and $m = 30$. It is however noteworthy that while the timing-based source estimator always performs better than its First-Spy counterpart, it doesn't do so by a significant margin in some scenarios. This happens when the malicious nodes are placed close to the source node of the payment paths, which is generally the case for the high number of short payment paths and the more distributed node positioning of the `lnbig` scenario in particular. Interestingly, we also found that the weight-based routing algorithm (see Section 9.2.1) puts edges to more central (hence, in the `mcentral` case, more malicious) nodes at the beginning of payment paths, which increases the success cases of the First-Spy source estimator.

In order to give a overall comparison of timing-based attacks on privacy to the First-Spy approach, we analyzed the number of payments that were fully deanonymized, i. e., the number of payments for which the adversary was able to correctly identify source *and* destination. To this end, the bottom row of Figure 9.6 shows the precision, recall, and F_1-measure with which the adversary could totally deanonymize payments given the estimators $\mathfrak{M}_{\mathsf{FS}}$ and $\mathfrak{M}_{\mathsf{T}}$. Of course, as this considers a subset of the correct results of each individual estimator, all measures are lower. However, the results generally follow the same behavior as just discussed. Notably, the timing-based estimator outperforms the end-to-end deanonymization performance of the First-Spy approach in every case of every scenario and in precision, recall, as well as F_1-measure. It does so in particular in the `mcentral` scenarios, in which it attack success is reliably higher than the baseline by factor 1.5. Thereby, our simulation results confirm the feasibility and improved adversarial success of timing-based attacks on privacy in payment channel networks.

9.4 DISCUSSION

In the following, we discuss possible steps towards attack mitigation, the impact of upcoming changes to the Lightning protocol, as well as avenues of future research.

9.4.1 *Possible Countermeasures*

The feasibility of timing attacks relies on the possibility to build a reliable model of latencies, and on the adversary's capability of observing and correlating of suitable interactive multi-hop message exchanges, such as the current `update_add_htlc`, `update_fail_htlc`, and `update_fulfill_htlc` message payloads.

Therefore, in order to impair the retrieval of timing measurements, message replies could be delayed for a random amount of time by the Lightning nodes, along the lines of Bitcoin's transaction trickling scheme [FV17] or a timed mix network [KEB98; Pio+17]. However, this would of course significantly delay payment processing and therefore directly conflict with Lightning's goal of enabling quick payments. It would furthermore counteract recent efforts to reduce end-to-end payment latencies, such as Boomerang proposal [BNT20].

Moreover, the adversary's capability of correlating payment observations could be impaired, e. g., by introducing a payment scheme that does not leak identifying payment features, such as today's payment hash, such as anonymous multi-hop locks [Mal+19]. Note however, that even given such a scheme, payment observations may still be correlated through metadata analysis, as timing and payment amounts. Furthermore, as an individual node still needs to be able to match incoming and outgoing network messages, such decorrelation would only protect of re-identifying the same payment in the network, i. e., mitigate full deanonymization. Very likely, the individual source or destination estimators could still be applied.

9.4.2 *Impact of Protocol Changes*

The issue of *payment path distinguishability* based on time-lock deltas was identified by the developers of the Lightning protocol some time ago, which lead to the introduction of so-called *shadow routes* to the Lightning standard [@Dev20b]. The idea behind shadow routes is to add a random padding to the overall time-delta value of payments, so that the set of possible destinations would not be identified by a remaining lock time of 0. That said, different implementations of Lightning handle the random padding differently, and, to the best of our knowledge `LND` currently does not implement shadow routes [@LD18] at all. In order to estimate what impact shadow routes would have on the accuracy of timing-based destination estimators $\mathfrak{M}_{t,\mathrm{T}}$, we re-evaluated the scenarios discussed in

Section 9.3.3 while disabling timelock-based anonymity set reduction. Even though this corresponds to a worst-case estimation, we found the decrease in precision and recall of the estimator to be only 2-3%, indicating that most of its performance is based on the timing-based maximum likelihood estimation.

The currently discussed proposal for *Rendez-Vous Routing* [@Dec20] would allow for the creation of partial onion messages that include the payloads for only a suffix of the payment path. These partial onion messages could then be handed to an untrusted party, which would be able to complete the payment path by supplying a suitable prefix to the *rendez-vous point*. This construction bears resemblance to Tor's hidden services and would allow for receiver-anonymous payments, i. e., would allow users to send payments whose location in the network they are not aware of. While the implementation of this proposal would therefore generally improve Lightning Network's privacy, it would likely not interfere with the feasibility of timing attacks.

As sending large payments given the current channel capacities is often unsuccessful, schemes allowing to split payments and route them over different paths, such as the recently implemented *multi-part payments*, have been discussed for some time in the Lightning community. As each individual payment carries only part of the overall amount, they provide increased value privacy, since the adversary is less likely to observe all payments and cannot infer the actual transaction volume. However, as this results in a higher number of closely correlated payments, an adversary has a higher probability to observe such payments, whereby the sender/receiver anonymity is decreased.

9.4.3 *Future Research*

Our analysis of timing attacks is based on the model of public Lightning nodes, as introduced in Section 9.1. However, the Lightning protocol also allows for the establishment of *hidden* payment channels that are only known to the adjacent neighbors and are not broadcasted in the public peer-to-peer network. As the estimators of course presuppose the knowledge of the underlying channel graph to be able to return the candidate endpoint of maximum likelihood, they are bound to fail in these circumstances. Therefore, applying methods from the research area of *topology inference* [DRT19; NAH16] in order to detect hidden channels would be an interesting avenue for future research.

In our network simulations, we furthermore observed cases in which the timing estimators wrongly identified the endpoints of unusually long payment paths as the candidates with maximum likelihood. This is often the case when these paths consist of many edges with small mean latencies, which then results in an aggregated distribution that is closer to the measured time difference than the correct candidate. Recently, Kappos et al. [Kap+20] proposed a model for endpoint deanonymization based on a probability distribution over payment path lengths. We think that integrating such an approach could help to exclude such unusually long paths and hence further improve the results of timing-based attacks on privacy.

10

ATTACHMENT STRATEGIES FOR PAYMENT CHANNEL NETWORKS

This chapter is based on the publication: Kimberly Lange, Elias Rohrer, and Florian Tschorsch. "On the Impact of Attachment Strategies for Payment Channel Networks." In: ICBC '21: Proceedings of the third International Conference on Blockchain and Cryptocurrency. 2021 [LRT21]. While this was a collaborative effort, the design and implementation of the simulation environment and the attachment strategies should be attributed to fellow co-authors.

As we discussed in previous chapters, the Lightning Network currently exhibits a high degree of centralization, which has been shown to be detrimental with respect to security and privacy. Since it furthermore utilizes a source-routed best-effort routing protocol to conduct multihop payments, payment reliability is highly dependent on the connectivity of involved nodes. Likewise, the position of routing nodes in the network topology is highly correlated with their fee revenue. Therefore, the question arises which connection points are preferable for nodes joining the network with respect to their connectivity or revenue.

In the following, we present an empirical analysis on the local and global impact of a variety of *attachment strategies* for payment channel networks. To this end, we survey the field of graph theory for strategies that aim to increase the joining node's connectivity and routing revenue and analyze their short-term and long-term impact on the network's efficiency and performance.

10.1 PRELIMINARIES

By the beginning of May 2020, the Lightning Network consisted of more than 4,300 nodes and around 25,000 payment channels exhibiting a combined capacity of more than 785 bitcoins (more than USD 8 million). We now introduce the necessary models and notations on which we base the further analysis presented in this chapter.

10.1.1 *Network Model*

We model the Lightning Network as a *directed multigraph* $G = (V, E)$, where the vertex set V constitutes the Lightning nodes and the multiset E the payment channels. Every bidirectional channel is represented by two directed edges in order to separately store the individual capacities and channel policies of both channel endpoints. Accordingly, the edge (u, v) stores how high u's share of the total channel balance is and which settings u chose for the channel. As each channel locks funds and each on-chain transaction involves costly transaction fees, opening many channels on the Lightning Network can be expensive. In order to reduce the number of required channels, the Lightning Network offers multihop routing, which enables the sending of payments to non-adjacent nodes in the network. In this case, the payment is routed over intermediate nodes along the payment path, which is determined by the payment's sender and secured by Hashed Time-Locked Contract (HTLC) protocols, as discussed in Section 6.1.

The payment's sender typically selects the most suitable route by running an adapted version of Dijkstra's shortest path algorithm [Dij59] that considers channel capacities, fees and locking duration in the edge weight calculation. That is, the algorithm first discards all candidate edges with insufficient capacities and then selects the path with minimal aggregated edge weights based on the intermediate nodes' fee policies and maximum lock-time. Such a weight-based algorithm is for example utilized by the popular `LND` implementation, which accounts for more than 90% of today's network nodes [TSZ19]. For the calculation of the respective transaction fees, each edge in the public network graph stores the routing fee policies, which are composed of a base fee f^B and a proportional fee f^P. The base fee f^B is a fixed amount that has to be paid to the routing node for every forwarded payment; the default value is 1 satoshi ($= 1 \cdot 10^{-8}$ BTC). The default value of proportional fee f^P is $1 \cdot 10^{-6}$ satoshi, which is multiplied with the transaction amount $|tx|$ of each payment. Therefore, routing higher value payments generates higher fees for the routing nodes. Concisely, the fee $f_u(v, |tx|)$ that has to be paid to the routing node u for forwarding a transaction with amount $|tx|$ to v can be calculated accordingly as

$$f_u(v, |tx|) = f_u^B(v) + f_u^P(v) \cdot |tx|.$$

In order to account for the weight-based routing algorithm, parts of our graph analysis is based on the *fee graph* $G_{F,|tx|}$, which we obtain through a transformation on G. This transformation allows the network analysis to account for Lightning's routing behavior in an approximative fashion, even when applying standard weight-based graph algorithms. In particular, G is reduced to $G_{F,|tx|}$ by excluding all edges of insufficient capacities with respect to a transaction amount $|tx|$. The weights for each edge (u, v) in $G_{F,|tx|}$ are set to $f_u(v, |tx|)$, i. e., they denominate the routing fees that would arise from transferring $|tx|$ through this channel.[1]

10.1.2 *Joining the Network*

Due to the costs associated with channel establishment, a node joining the network should follow a certain set of rules for choosing its initial connection points according to an optimization goal. We call such an algorithm returning a candidate node set $C \subseteq V$ an *attachment strategy*

$$\mathscr{S}(G, k, \texttt{cap}) \to C,$$

---

[1] Note that this approximative approach is only applied when necessary for general graph analysis or as part of the attachment algorithms. In contrast, the simulation framework used for the evaluation of the proposed strategies follows a payment protocol that closely resembles the real-world behavior, as will be discussed in Section 10.3.1.

which takes as parameters the public network graph G, the number of channels to be opened $k = |C|$, and the capacity `cap` (in satoshi) that each of the channels should hold.

The respective optimization goal depends on the motivation for joining the network. We consider the attachment strategies from the point-of-view of three distinct perspectives:

- *End-users* join the network to conduct cheap, reliable, and fast payments and therefore are interested in strategies that improve their local connectivity to the network.
- *Service providers* participate as routing nodes in the network in order to earn transaction fees. They are therefore interested in optimizing their local node's channel selection in order to receive maximal profit.
- *The network* perspective regards the global impact of a particular strategy and considers its impact on the network's overall connectivity and reliability over time.

As these view points follow partly conflicting interests, they may not easily be reconciled, but expose a fundamental trade-off between short-term egoistical efficiency and the long-term development of the network (cf. [WH20]). However, as different attachment strategies fall on different points in the spectrum of this trade-off, we empirically investigate their usefulness regarding these three view points.

The performance of each strategy of course highly depends on the user's behavior: if we for example assume an end-user would conduct frequent payments to only a single service provider, the optimal connectivity-oriented strategy would be to establish a direct payment channel to it. However, so far no reliable data source on user behavior in payment channel networks is publicly available to the research community, which necessitates the introduction of a number of assumptions with regard to the payment model. To this end, we refrain from introducing overly complex assumptions that may act as confounding factors to our analysis. In particular, we assume for the sake of simplicity that the user plans to send payments to destinations all over the network. Moreover, we assume that the capacity `cap` is the same for all k channels and that initial balances are split equally between the channel endpoints. We also assume that every node in the network agrees to open a channel, which may not be the case in the real network, in particular since recent research found such optimistic behavior to entail security risks [HZ20]. Finally, we assume new channels to be established with the default fee settings. Note that in current Lightning implementations attachment strategies are used in the so-called *autopilot* feature that allows the client software to automatically choose and establish new channels.

10.2 NETWORK ATTACHMENT STRATEGIES

In the following, we introduce candidate strategies for nodes joining payment channel networks. We also provide a first assessment of their applicability as well as their complexity in dependence of the number of nodes $n = |V|$ and number of edges $m = |E|$.

10.2.1 *Random*

The Random strategy is the simplest attachment strategy, in which the attachment points are determined by uniform random sampling from the node set V. This strategy can be quickly computed in $\mathcal{O}(n)$ and, while it mainly serves as a baseline for comparison, it counteracts centralizing tendencies since it does not prefer any particular connection point.

10.2.2 *Highest Degree*

The Highest Degree strategy sorts all nodes V according to their degree, and returns the k nodes with the highest degrees. As the number of different neighbors is presumably more meaningful than the total number of channels a node v has, its degree $\deg(v)$ is determined in the fee graph G_F, since it disregards multi-edges. The candidate set can be computed quickly with this strategy because $\deg(v)$ can be retrieved from the adjacency lists and sorting can be done in $\mathcal{O}(n \log n)$.

Connecting to nodes with highest degrees is an extreme form of *preferential attachment* which is known to induce a "rich-gets-richer" effect that yields scale-free networks [BA99], and is likely responsible for the highly centralized substructures found in the Lightning Network today. In fact, highest-degree attachment strategies were deployed in prior versions of `LND`'s autopilot feature and have been critically discussed in the community [@Pic].

10.2.3 *Betweenness Centrality*

The notion of *betweenness centrality* [Fre77b] indicates how many shortest paths in the network graph G a node v is part of. More specifically, $bc(v) = \sum_{\substack{s,t \in V \\ s \neq v \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$, where σ_{st} is the total number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of shortest paths from s to t via v.

In context of Lightning, nodes exhibiting a high betweenness centrality implies that they are often chosen by the weight-based routing algorithm and therefore are part of many payment paths. Since a large share of the network can be reached via these nodes with minimal distance in terms of fees, they are in return promising candidates for node attachment. Note that this often also corresponds to overall shorter payment paths, which improves reliability.

Consequently, the Betweenness attachment strategy elects the k nodes with the highest betweenness centrality values, which are calculated via the weighted Brandes' algorithm [Bra08] based on the fee graph G_F. As the weighted version of the algorithm has a runtime complexity in $\mathcal{O}(nm + n^2 \log n)$, our implementation additionally employs the optimizations from [Bag+12], which speed up the calculation of betweenness centralities (but do not change the algorithmic complexity). Connecting to nodes with the highest betweenness centralities is another form of preferential attachment and likely results in further network centralization.

10.2.4 *k-Center*

The k-Center strategy is based on the assumption that the joining node can improve its overall connectivity to the network by establishing channels to k nodes such that the highest distances between them and any other node in the network are minimized. Ideally, this would lead to nodes in different parts of the network being chosen as the k new neighbors in order to minimize the length of the longest shortest payment path. This likely results in faster and cheaper transactions due to fewer nodes being part of the routes. Reducing the number of nodes and channels contained in a payment route can also decrease the risk that a transaction fails as there are less points of failure.

The idea for this strategy is based on the k-center problem [HS85], which is defined as follows.

Definition 9. *Given a complete undirected graph $G = (V, E)$ in a metric space and an integer k, a k-center is a subset of nodes $C \subseteq V$ with $|C| \leq k$ such that $max_{v \in V} d(v, C)$ is minimized, with $d(v, C)$ being the shortest distance of v to the closest node in C.*

It was previously proven that this problem is NP-complete and that it is NP-hard even for an ϵ-approximation with $\epsilon < 2$ [HS85]. This means that 2-approximation algorithms, which return a solution that is within twice the optimal solution value in polynomial time, are the best possible algorithms for the k-center problem, unless $P = NP$. Due to the fact that distances in the fee graph G_F are not necessarily symmetric, the Lightning Network unfortunately cannot be modeled as a weighted fee graph in metric space. We therefore use the greedy k-center algorithm introduced in [Gon85] on a generated complete *distance graph* to minimize the number of hops on the longest shortest path and disregard fees or channel capacities. To this end, the joining node first establishes a connection to the network's highest degree node and then executes a single-source shortest path (SSSP) search to retrieve the distances for the k-center algorithm. This results in a total time complexity of $\mathcal{O}(k(m+n))$.

As the k-Center strategy aims to interconnect the network centers, it should improve the network's robustness and facilitate decentralization.

10.2.5 *k-Median*

Besides looking at the longest shortest path to any other node in the network, a promising strategy is to minimize the average shortest path distance to all other nodes. Assuming that the joining node sends a transaction to any other node with the same probability, it is very likely favorable to require a minimal average number of hops to any other node in order to reduce transaction fees, latencies, and failures. Hence, we have to solve a problem that is known as the *Single-Source Average Shortest Path Distance Minimization* (SS-ASPDM) problem [MT09]. It was previously proven that an optimal solution to the SS-ASPDM problem for a node v can be found by only adding edges incident to v [MT09]. Thus, the problem can be utilized in our use case of the Lightning Network since a joining node may only influence the opening of channels which are incident to itself. Adopting the approach to only add edges incident to the source node v, the SS-ASPDM problem corresponds to the k-median problem [MT09].

In a graph context, the k-median problem can be formulated as follows.

Definition 10. *Given a complete undirected graph $G = (V, E)$ in a metric space and an integer k, the k-median problem strives to find a subset of nodes $C \subseteq V$ with $|C| \leq k$ such that $\sum_{v \in V} d(v, C)$ is minimized, with $d(v, C)$ being the shortest distance of v to the nearest node in C.*

Again, the problem is NP-hard [CKY05] and only an approximate solution can be found within polynomial time, unless $P = NP$. For solving the k-median problem in a distance graph, we establish an initial connection to the highest degree node and then utilize the "forward" greedy algorithm presented in [CKY05], which results in an overall time complexity of $\mathcal{O}(kn(n+m)\log n)$ when applied to the weighted fee graph.

Similarly to the k-Center approach, the k-Median strategy promises to improve network robustness and reduce centralization.

10.2.6 *Maximum Betweenness Improvement (MBI)*

A node that joins the network with the intent to act as a service provider or routing node strives for financial profit from participating in the Lightning Network. To this end, a routing node v should rather focus on optimizing its own betweenness centrality $bc(v)$ than connecting to central nodes.

Therefore, it has to solve a problem known as Maximum Betweenness Improvement (MBI) [Ber+18b], which is defined as follows.

Definition 11 (MBI). *Given a directed graph G, a node v, and an integer k, which set of edges S incident to v, with $|S| \leq k$, should be added to G in order to maximize $bc(v)$?*

The MBI problem has been proven to be NP-hard, but a greedy algorithm that provides an approximate solution exists [Ber+18b].

This MBI strategy temporarily opens any channel that node v could set up, calculates $bc(v)$, and closes the channel again. This is repeated for all possible channels and in the end the channel generating the highest betweenness improvement for the joining node is elected. This channel is then established and the procedure is repeated until all k candidates are found, leading to an overall high time complexity in $\mathcal{O}(kn^3)$.

Note that this strategy is similar to the approach found in [ERE20b], which however also optimizes the node's fee settings. As this results in a further increased computational complexity over the already high resource requirements of Bergamini et al.'s algorithm, we in lieu of these optimizations follow the more feasible MBI strategy.

10.3 EMPIRICAL ANALYSIS

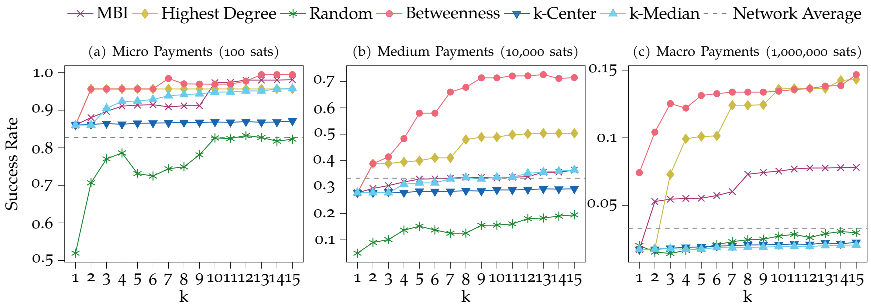
In the following, we empirically analyze the performance of attachment strategies for payment channel networks from a local perspective, i. e., from the view of a single end-user or service provider aiming to join the network.

10.3.1 *Network Simulator, Setup, and Methodology*

As a basis for the empirical analysis, we developed a time-discrete event simulator that implements the network multigraph model (cf. Section 10.1.1) and allows to simulate payment processing as well as nodes joining the network according to a given attachment strategy.[2] The simulator initially reads the network graph from a snapshot of the Lightning Network and simulates path finding through a weight-based route selection algorithm similar to the one found in `LND`. While some aspects of the real-world payment procedure—such as the HTLC protocol negotiations—are omitted by our simulation model for the sake of simplicity, the simulator was carefully implemented to approximate the real-world behavior. To this end, transaction processing is simulated by checking and adjusting the available balances along the payment path. During this phase, the arising fee revenues are calculated based on the provided fee policies and the remaining transaction value for

---

[2] The simulator code base is publicly available in our companion repository at `https://git.tu-berlin.de/rohrer/pcn-attachment-data`.



(a) Micro Payments (100 sats)

(b) Medium Payments (10,000 sats)

(c) Macro Payments (1,000,000 sats)

Figure 10.1: Transaction success rates in dependence of chosen attachment strategy and transaction amounts.

each hop along the way. Note that consequentially and just as in the real network, transaction success is not guaranteed even if a path is found, as the path finding algorithm does not operate on the private balances, but the public capacities. We base our further analysis on a snapshot of the Lightning Network from May 1, 2020 at 10am that was taken from the dataset [@Roh] provided by [RMT19]. At this time, the largest connected component of the network consisted of more than 4,300 nodes connected by nearly 25,000 channels, which held an overall capacity of more than 785 BTC.

In order to analyze their performance, we simulated the joining of individual nodes according to the given attachment strategy, every time establishing $k \in \{1, \ldots, 15\}$ channels with sufficient capacity and default fee settings. We then evaluated the connectivity and fee revenue of the joined node through two sets of simulated payments: one set of 1,000 transactions with the joined node as a fixed source and the destination selected by uniform random sampling, and another set of 1,000 transactions for which both source and destination were chosen randomly. The simulations were conducted under the assumption of three different transaction volumes: micro payments of 100 sats, medium payments of 10,000 sats, and macro payments of 1,000,000 sats (see also [ERE20b]). If not stated otherwise, our analysis is based on the most relaxed assumption of 100 sats. For every strategy, transaction value, and every value of k, the simulations were furthermore repeated 30 times with different seed value inputs for the utilized random number generator. This results in a five-digit sample size ensuring the statistical significance of the results.

10.3.2 *Transaction Success*

In order to assess the impact of the attachment strategies on the connectivity of the joining node, we analyzed the average transaction success rate, i. e., the share of all transaction that actually succeeded. In Figure 10.1, the average success rate is shown in dependence of the number of transaction amounts and channels k that were established corresponding to the respective strategies. Moreover, the a priori network-wide average success rate is shown for comparison, which was determined by simulating 10,000 transactions with randomly chosen sources and destinations in the initial graph configuration.

We observe that generally node connectivity improves with the number of established channels and that all but the Random strategy tend to result in an average success rate higher than the network average. Moreover, strategies that prefer central connection points, such as the Betweenness strategy, fare better than strategies that connect the periphery of the network, such as k-Center. This is likely the case because connecting to very central points in the network reduces the average path length and thereby also the probability of routing failures due to unavailable balances.

This is supported by the fact that the assumed transaction volume has a big impact on the average success rate: while the network average for micro payments is around 83% (Figure 10.1a), it drops below 34% for medium payments (Figure 10.1b), and even to less than 4% for macro payments (Figure 10.1c). This observation is of course in line with previous literature, in which Lightning's limited available capacity and the resulting low success rates for higher-volume payments have been discussed for some time [BSB19b; RMT19; WH20]. Our results underline that currently only a small number of central nodes hold enough capacity to be able to route any high-volume payments. While the heavily skewed capacity distribution results in overall very low transaction success rates, we observe that strategies that preferably connect to these few central nodes—such as Betweenness, Highest Degree, and MBI—can increase their lead in such high-volume payment scenarios. However, in order to limit the impact the current capacity constraints found in the Lightning Network have on our results, we continue our further analysis of attachment strategies under the most relaxed assumption of micro payments.

10.3.3 *Transaction Fees*

End-users joining the Lightning Network likely want to optimize their connection point with regards to the result fees that arise from sending payments. In Figure 10.2a the fees paid by the connecting node are shown in dependence of the number of channels and with respect to the chosen strategy. Again, generally all strategies result in fee costs lower than the network average, which is even true for Random for more than $k = 5$ channels. As the fees in most cases improve linearly with the number of established channels, it can be concluded that overall better connectivity and the resulting increased routing opportunities help to reduce the cost associated with sending Lightning payments.

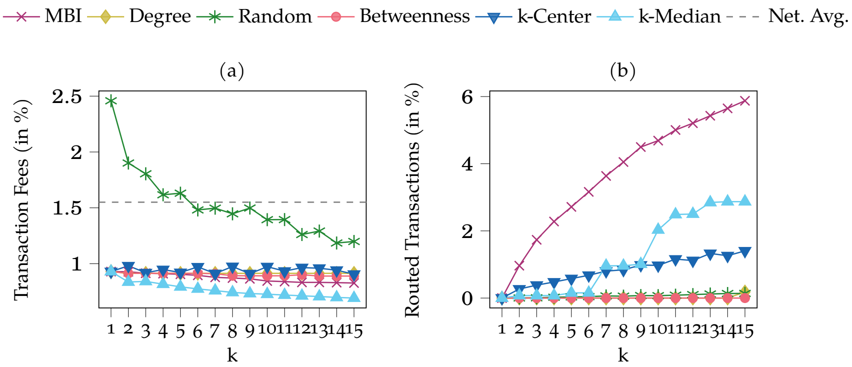


Figure 10.2: Transaction fees and share of routed transactions in dependence of chosen attachment strategy.

Interestingly, the k-Median strategy is the clear favorite with regards to fee saving, likely as it helps to increase connectivity between network clusters and connects these as well as more centralized nodes.

10.3.4 *Service Provider Revenue*

Service providers join the network with the intend to earn the maximum amount of profit. To this end, we analyze which attachment strategy can help to improve their fee revenue. The share of routed transactions (which in our case directly corresponds to the fee revenue) is shown in Figure 10.2b.[3]

Independently of the strategy, the share of routed transactions improves with the overall connectivity of the joining nodes, i. e., it increases with the number of established payment channels k, but tends to favor strategies that improve path diversity. However, the MBI strategy is clearly superior in this regard, allowing the joining node even to route close to 6% of all payments conducted in the network by establishing $k = 15$ channels. This comes to no surprise as this strategy is specifically focused on maximizing the number of payment paths routed through the joining node, and previous work showed the benefits of such an approach [ERE20b]. Apart from this, the k-Median strategy is a promising candidate, as it able to secure the service provider a routing share of close to 3% of all payments in the case of $k = 15$.

Table 10.1: Algorithm runtimes in dependence of chosen attachment strategy and number k of established channels (in sec.).

k	Highest Degree	Betweenness	k-Median	k-Center	MBI
1	0.25	440.00	2.70	0.51	2,784.00
2	0.25	440.00	4.70	0.59	4,834.00
3	0.25	440.00	6.40	0.63	6,965.00
4	0.25	440.00	8.50	0.66	9,232.00
5	0.25	440.00	9.70	0.67	11,429.00
6	0.25	440.00	11.30	0.72	13,938.00
7	0.25	440.00	13.30	0.75	16,478.00
8	0.25	440.00	15.00	0.81	19,371.00
9	0.25	440.00	17.10	0.88	22,294.00
10	0.25	440.00	18.20	0.81	24,634.00

10.3.5 *Runtime Analysis*

In order for a attachment strategy to be an actual candidate to be implemented in the autopilot functionality of a Lightning client implementation, it should deliver its results in a viable amount of time. Therefore, we measured the run times of discussed strategies under real-world conditions. To this end, we deployed our strategy implementations on an `t2.xlarge` instance (4 vCPUs based on Intel Xeon 3.3 GHz, 16 GB memory) on Amazon Elastic Compute Cloud (EC2) running Ubuntu Server 18.04. We then measured the execution time that it took the algorithms to return the respective candidate sets.

The results shown in Table 10.1 generally concur with our complexity analysis given in Section 10.2: while the Highest Degree, Betweenness and k-Center strategies remain roughly constant runtimes, k-Median and especially MBI grow in a linear fashion with the number of established payment channels k.

This is of particular significance, since it takes MBI between 2,000 and 2,500 seconds longer to finish for each additional channel. As this amounts to an overall runtime of around seven hours for $k = 10$, the practicability of this strategy is heavily put under question, potentially even given its performance benefits in terms of fee revenue.

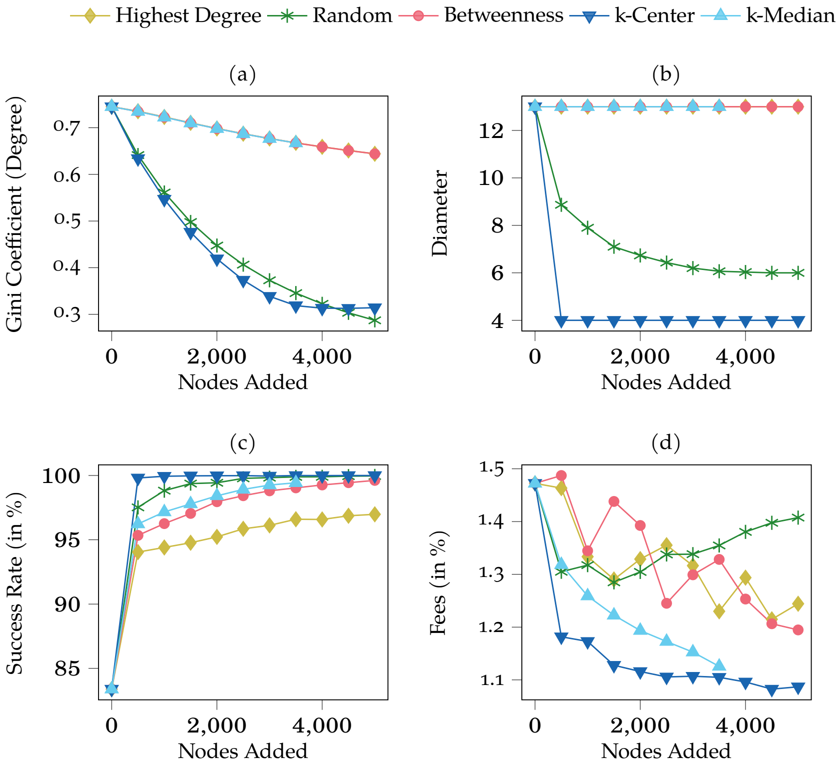


Figure 10.3: Long-term impact of attachment strategies on the network.

10.4 EVALUATING THE LONG-TERM IMPACT

So far, we analyzed the discussed attachment strategies with respect to their local short-term impact, i. e., from the point of view of an egoistical node joining the network. In the following, we assess the global long-term impact of the discussed attachment strategies for payment channel networks.

10.4.1 *Simulation Setup*

In order to evaluate the global long-term impact, we utilized the time-discrete event-based network simulator from Section 10.3.1 to model the process of 5,000 nodes sequentially joining the Lightning Network, which corresponds to more than doubling the network size. Each node joins the network with $k = 10$ channels that are established according to the given strategy, which is roughly the network average node degree. While the future network development will probably not exactly follow these assumptions, this approach allows us to compare the advantages

---

[3] Note that our analysis compares the proportional fee revenues gained from routing in the Lightning Network and does not consider any costs for running a routing node, such as the on-chain fees associated with channel establishment. In order to estimate the net. profit of a node operator, such cost would have to be known and subtracted from the revenue.

and drawbacks of each strategy without considering additionally interfering and confounding factors. This simulation-based analysis was conducted for all but the MBI strategy. Due to MBI's significantly higher computational requirements (cf. Section 10.3.5), we had to refrain from including it in the long-term evaluation. As before, all randomized transactions were repeated 1,000 and all simulations 30 times to ensure statistically significant results.

10.4.2 *Impact on the Network's Topology*

In order to analyze the impact each attachment strategy has on network centralization over time, we analyzed the network topology in intervals of 500 joining nodes and recorded essential network metrics. Figure 10.3a shows the Gini coefficient of the node degree, which quantifies the inequality of the degree distribution for an increasing number of added nodes. As expected, the network exhibits initially a high Gini value of nearly 0.75, which underlines the high degree of inequality currently exhibited by Lightning's network topology. Furthermore, the results show that strategies following a preferential attachment pattern, such as the Highest Degree or Betweenness strategies only marginally decrease the centralization over time, while strategies that also connect the fringes of the network, such as k-Center and Random have a strong positive impact on centralization. Interestingly, we observe that k-Median tends to elect the same set of k nodes over time. While these k nodes increase their connectivity, it does not result in a significant improvement with respect to the degree inequality.

Figure 10.3b shows the average network diameter, i. e., the longest shortest path in the network, which is an indicator for the worst-case routing complexity. Again, Random and k-Center perform best and are able to immensely reduce the initial network diameter of 13 already after attaching 500 nodes. Notably, the k-Center strategy quickly allows all network nodes to reach all other nodes in just four hops. In order to get an understanding of how the participation in routing is impacted over time, we analyzed the inequality of betweenness centralities and the central point dominance. The results generally concur with our observations for node degrees. They also show that our current choice of establishing the initial connection of the k-Center and k-Median strategies to the single highest degree node results in an increased central point dominance. While this is an implementation detail, its impact requires further investigation in the future.

10.4.3 *Impact on the Network's Performance*

In order to evaluate the performance of the network in dependence of each attachment strategy, we analyzed the average success rate and the arising fees by regularly simulating transactions in the network. To this end, we executed batches of 1,000 micro transactions with randomly chosen sources and destinations after the addition of every 500 nodes.

As can be seen in Figure 10.3c, the average network success rate generally improves with an increasing number of nodes and the additionally provided routing capacity. The evaluation moreover shows that again the decentralizing strategies Random and especially k-Center benefit the overall network connectivity the most, letting the success rate quickly rise to close to 100%.

While this pattern is generally also reflected in the average paid transaction fees, as shown in Figure 10.3d, the results highlight that a high degree of centralization can be beneficial for fee costs. In particular, while the Highest Degree strategy does generally not offer many benefits, it does result in rather low average fee costs. This is likely due to the short average path lengths and high efficiency of star sub-structures (cf. [Ava+20; SZ20]). However, again the k-Center strategy proves to be the most promising candidate to minimize fee costs for the end-user in the long term, with k-Median being a close second.

10.4.4 *Discussion*

Throughout our analysis, it became apparent that the Lightning Network currently is heavily restricted by its overall limited capacity and its concentration on a few central service providers. We therefore found that the provided quality of service and user experience would immensely benefit from any kind of higher-volume and higher-connectivity adoption.

We also found that from an egoistical perspective, strategies selecting central attachment points seem to provide the best short-term performance, with the exception of transaction fees, in which case the k-Median strategy showed to be the most promising candidate. From the global point of view, however, decentralizing strategies proved to provide the best long-term benefits for the network overall. With regard to this conflict of interest, we empirically confirm the trade-off between efficiency and decentralization [Ava+20; WH20].

However, our analysis showed two strategies to be feasible and potentially capable of combining local short-term and global long-term interests: k-Center and k-Median. While these strategies may not be the absolute optimum from the egoistical point of view, they benefit the long-term network development the most. It therefore remains an open question whether users would accept non-optimal short-term strategies, if they benefit them and the whole network in the long-term.

In order to balance this trade-off, real-world implementations should consider to employ a set of different well-chosen strategies to establish their channels. However, the exact choices and the share of connections established through a particular strategy are up to further analysis. Our implementation of the k-Center and k-Median strategies currently builds on an initial centralized connection. We therefore also deem the potential of such “mixed” strategies, i. e., strategies that further randomize and distribute these connection types, a promising subject for future research. Furthermore, while we generally hold the inherent conflicts of interest to be hard to reconcile, we think they should further be discussed and addressed in the community.

Part III

BEYOND CRYPTOCURRENCIES

11

WEBCHAIN: ENABLING REFERENCE VERIFIABILITY FOR THE WORLD WIDE WEB

While currently the main fields of application for blockchain technologies are of financial nature, many of the underlying foundational principles can be utilized in order to build resilient distributed systems beyond cryptocurrencies. In this regard, we in the following present Webchain, a decentralized system inspired by blockchains and distributed timestamping schemes that enables source and reference verifiability for resources on the World Wide Web.

This chapter is based on the publications: Elias Rohrer, Steffen Heidel, and Florian Tschorsch. "Webchain: Verifiable Citations and References for the World Wide Web." In: BLOCKCHAIN '18: Proceedings of the IEEE International Conference on Blockchain. July 2018 [RHT18] and Elias Rohrer, Steffen Heidel, and Florian Tschorsch. "Enabling Reference Verifiability for the World Wide Web with Webchain." In: ACM Trans. Internet Techn. 20.4 (2020) [RHT20]. While they were collaborative efforts, the implementation of the prototype and the evaluation should be attributed to fellow co-authors.

11.1 REFERENCE ROT AND SOURCE VERIFIABILITY IN ACCELERATED TIMES

In recent years, online source material and references have become more and more prevalent, not only in journalistic writing, but also in scientific literature. However, given the ephemeral nature of the World Wide Web, the so-called 'reference rot' is at the same time becoming a more serious issue.

The availability and verifiability of internet references have been studied in quite diverse fields of scientific literature, and most studies found the current state to be vastly unreliable. For example, Aronsky et al. [Aro+07] studied the biomedical literature published in PubMed [@Med19] in 2006 and found that, while online references were not prevalent in this field at the time, more than 10% of references were unavailable within two days after publication. Similarly, Dellavalle et al. [Del+03] examined three top-tier medical journals and also found high unavailability rates soon after publication (3.8% at 3 months, 10% at 15 months, and 13% at 27 months after publication). Furthermore, Bugeja and Dimitrova [BD05] uncovered that 40% of online references were unavailable a year after publication, when they studied the state of online references in journalism conference papers. As it is based on the then increasingly defunct principles of replicability and reproducibility, the vanishing of such large amounts of online references damages the feasibility of the scientific method.

This is also observed by Karpf in [Kar12], as he notes that the ephemeral nature of the World Wide Web is a big obstacle in the way of current social science research. While Karpf acknowledges the existence of the Wayback Machine of the Internet Archive [@Mac19] and deems it an *exceptional research resource*, he also recognizes its limits and explicitly calls for more such "lobster traps", i. e., public archival systems of similar nature.

Rogers [Rog15] discusses *digital methods* and the "messiness" of online material with respect to the consequences that may arise for research in the humanities. In particular, the author raises the question on which kinds of facts research may be grounded and arrives at the conclusion that, in absence of source and reference verifiability, digital methods and online observations have to be grounded on offline data. However, he also observes a trend in the opposite direction: more and more research methods and journalistic investigations are conducted based on online data.

This trend goes hand in hand with new forms of journalism, such as citizen, data, and networked journalism. While the former two describe that more independent citizens conduct investigations of journalistic nature for their own merit and that this research is more often data-driven, the latter notion tries to capture the current amalgamation of such concepts: in the emerging networked journalism [VPC12], journalists work together with independent data sources, citizen investigators, and even base their coverage in part on crowdsourced and user-generated materials, such as photographies, videos, and textual reports. Of course, this entails data collection from social media and other publically available resources, also known as open-source intelligence (OSINT) [Gib04].

Given such a diverse set of sources of varying reliability, new technologies can assist with the tedious fact-checking process and allow to automatize the formal verification of the provenance and integrity of online sources. This kind of automatization is especially important when the information flow is speeding up, because then journalists are left with less time to check (online) sources, as Van der Haak et al. stress:

> "While working at Internet speed does not change the basic principles of journalism, it does make the reflective practice more difficult. The greater the volume of information to be scrutinized and the faster its input is demanded for news production, the less time is left for analytical treatment and storytelling." [VPC12]

The authors therefore arrive at the conclusion that source verification and archival technologies allow to increase the level of automatization in contemporary journalism while the journalists "will concentrate on the interpretation, analysis, and storytelling of the slower and more fundamental changes in society." [VPC12]

In this chapter, we present Webchain, a system that aims to address the discussed challenges: as it provides the functionality to securely verify the formal authenticity, integrity, and provenance of online sources and references, it is suited to aid a more automatized fact-checking process. Moreover, as it enables the verification of citation provenance even in absence of the original source, it could be an effective tool to mitigate the negative effects on source verifiability introduced by reference rot.

11.2 RELATED WORK

The challenging goal of digital preservation is defined as the archival of authenticated content over time. To this end, web cache and archival systems, such as the Wayback Machine at the Internet Archive [@Mac19], the LOCKSS system [Man+05], Perma.cc [@Per19], and WebCite [ET05] have evolved to ensure the preservation of digital content. While these approaches sustain availability of web pages, they do not necessarily protect from manipulation of the archive's contents [ANW17].

To some extent, content-addressable networks [Jac+12] in general and the InterPlanetary File System (IPFS) [Ben14] in particular address these issues by offering a storage layer in which data is addressed and secured by cryptographic hashes. It requires the users, though, to employ these systems as immediate medium of communication. For example, the recently proposed DClaims system [SSD19] leverages IPFS storage in combination with Ethereum smart contracts to enable the decentralized publication and retrieval of web annotations [@W3C20b]. While this system allows to annotate articles (e. g., for the sake of fact-checking), it does not secure the article and reference data.

On the other hand, approaches like the trusty URI scheme introduced by Kuhn and Dumontier [KD15] aim to improve the verifiability of web artifacts by appending hashes to the artifact URI. As the data needed for verification is included in the URI, users receiving trusty URIs can independently verify that the retrieved data has not been tampered with. If an artifact secured in this way is referencing others via trusty URIs, this approach secures the integrity of the entire reference tree, an effect similar to the secured citation graphs of Webchain. It however does neither ensure availability, nor non-repudiable authenticity of the secured reference data: when the author decides to delete an artifact, the chain of references breaks, potentially leaving previously secured subtrees in a unverifiable state. In contrast, as Webchain is an independent distributed system, it allows to verify the citation graphs even if the original artifacts become inaccessible.

The field of data provenance has been explored extensively in recent years, and standards like W3C's `PROV` have been developed [@W3C20a]. However, while recent work [Mor17] provided the necessary groundwork in order to enable verifiable `PROV` documents through the application of digital signatures, no procedures have been standardized to this end as of yet. In [HSW09], Hasan et al. introduce the Sprov system which provides a secure provenance scheme based on hashed and interlinked provenance records. Therefore, their notion of provenance chains bears some resemblance to Webchain's citation graphs. However, while Sprov is meant to run at the lower layers of a single machine's operating system, Webchain offers secure citation provenance of web artifacts and offers a decentralized trust model by applying a distributing timestamping scheme. Moreover, it could be extended for compatibility with the `PROV` data model, once the necessary procedures have been standardized.

To the best of our knowledge, the Webchain system is the first approach that combines content authenticity and integrity with citation provenance and non-repudiation provided by secure timestamping. While approaches which require complete trust in a centralized authority exist [Fac+09; HM14], Webchain's architecture distributes the necessary trust. To this end, Webchain constructs a distributed ledger, which borrows concepts from blockchain-based approaches such as Bitcoin [Nak08] and directed acyclic graphs (DAGs) of blocks. However, in contrast to most blockchain systems, Webchain follows the separation of concerns principle and distributes the data over multiple independent ledgers.

Systems like OpenTimestamps [@Ope19] and ProvChain [Lia+17] utilize the Bitcoin blockchain as a back-end for a secure time attestation service. However, while the former allows for the attestation of arbitrary file data, the latter is a system that records provenance data for files in cloud storages. To this end, it records file operations, encodes them in JSON format, and publishes a hash digest to the Bitcoin blockchain via a third-party API provider in order to retrieve a secured timestamp. In difference to these approaches, Webchain allows for the authenticated publication and referencing of article data on the World Wide Web.

A central component of Webchain's architecture is the timestamping service. In order to distribute the trust, we integrate Haber and Stornetta's distributed timestamping scheme [HS91]. While they assume a static network and do not specify node addressing, we fill the gap and develop a protocol for a dynamic overlay network, which includes maintaining network state in a distributed environment.

11.3 SYSTEM OVERVIEW

Webchain enables secure author attribution and verifiable citation of articles published on the World Wide Web, such as news articles or blog posts.[1] In the following, we give an overview of Webchain's core architecture, before discussing individual system components.

11.3.1 *The Webchain Architecture*

The backbone protocol of the Webchain architecture is inspired by blockchain-based systems, such as Bitcoin [Nak08]. Like these systems, Webchain draws its security and accountability from replicated data

---

[1] Note the Webchain architecture could generally be adapted to support other means of communication (e. g., IPFS [Ben14]) and additional data formats, such as PDF files. This would however require specific adjustments for article normalization and the addressing of data. We therefore focus our account on the primary use-case of articles published on the World Wide Web.

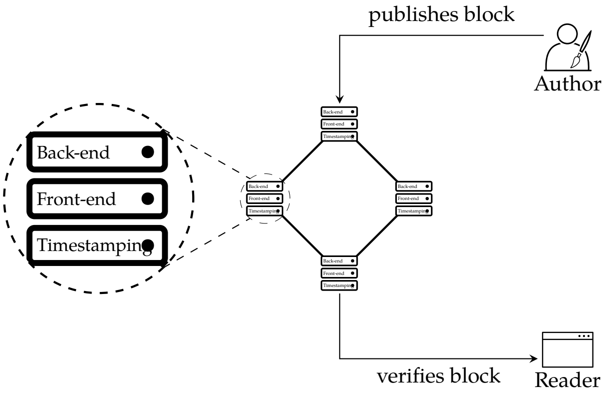


Figure 11.1: Webchain architecture.

structures consisting of cryptographically interlinked blocks, i. e., distributed ledgers. In Webchain however, every block represents an article published on the WWW. Each of these blocks is linked to every block corresponding to a cited article. Updates to a distributed ledger of blocks are broadcast in a network of *infrastructure nodes* that verify and apply them to their local ledger state. Infrastructure nodes consist of three components (see Figure 11.1): a back-end component that manages and provides access to the distributed ledger data, a front-end component that integrates an author-facing content management system, and a timestamping component that securely keeps track of block creation (see Section 11.4).

The Webchain system knows two types of users: *authors* and *readers*. Authors register at an infrastructure node to create and publish articles, which may cite other articles. The Webchain front-end provides an interface which authors use to add or edit articles. The front-end also takes care of all Webchain-related procedures. In particular, it verifies the validity of citation chains, creates a new block, and embeds the Webchain metadata into the article, which provides a verifiable reference to the respective block. In addition, articles are signed by the author. We envisage that readers use a lightweight *client* software, implemented as a browser plugin. The client scrapes visited websites for embedded Webchain metadata, queries Webchain infrastructure nodes for the block data, and verifies the citation chain. Therefore, the client does not need additional external knowledge to verify block integrity locally and in particular is not required to trust a specific infrastructure node to be trustworthy.

11.3.2 *Block Layout*

A Webchain block B is defined as the 6-tuple $B = (id, f, t, m, \sigma_{id}, D)$, which is also illustrated in Figure 11.2. A block's main purpose is to hold and secure a set of data entries D. In the Webchain system, we

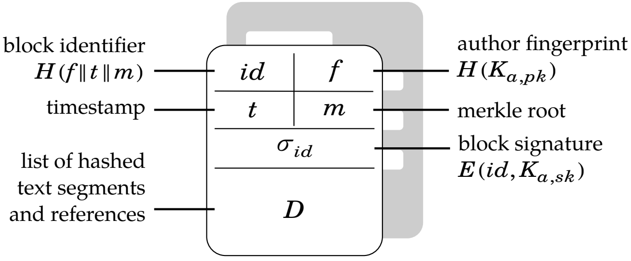


Figure 11.2: Webchain block layout.

differentiate two types of data entries: (i) hash values of text segments, which are part of an article and (ii) references to other blocks' data entries. These entry types are stored as a list, discriminated by a preceding type header. In order to secure data entries, they are used to construct a Merkle tree [Mer87]. As all values are incorporated in the root value m, the *Merkle root*, the value m secures the integrity of all block data. Furthermore, Webchain adds a POSIX timestamp t and the block author's fingerprint f, which is given by applying a cryptographically secure hash function H to the author's public key $K_{a,pk}$, i. e., $H(K_{a,pk})$. A block is uniquely identified by the block identifier $id = H(f\|t\|m)$, which is also recorded in a block. Finally, the author adds her digital signature $\sigma_{id} = sign(id, K_{a,sk})$, where $K_{a,sk}$ denotes her private key. This cryptographic block construction ensures that block identifiers depend on the block content as well as on the time it was created. Therefore, the construction allows to check block integrity by reproducing and verifying the block identifier based on the article data itself. Since a cryptographic signature of the identifier is provided by the author, a block's authenticity can also be validated. We assume that the respective key material is known by the validating party. This can be implemented by employing a variety of well-known identity management techniques which we discuss in Section 11.5.1.

11.3.3 *Block Creation*

From an author's perspective creating a new article feels very much the same as before: she signs in to her respective infrastructure node's CMS and uses the web interface to edit an article. After saving the article, however, the infrastructure node prepares the article by subdividing the text into text segments. The reason for this kind of segmentation is to provide other authors the ability to cite individual text segments rather then requiring to cite complete articles only. The segmentation can be achieved in different ways, e. g., by subdividing HTML elements. Alternatively, the text could be segmented based on sentence boundaries or semantically coherent text spans. Both are well-studied areas in the field of natural language processing [GT94; PH97; RR97]. An addi-

```
<blockquote data-webchain-id="f115e016daad472...">
Citing an author whose ideas or information you used is paying a
    debt.
-- Umberto Eco, How to Write a Thesis.
</blockquote>
```

Figure 11.3: Example of an embedded citation code.

tional normalization step might be necessary to yield an deterministic segmentation.

After preprocessing, the infrastructure node applies the hash function H to each of the article's segments and appends the results to the data section D of the new block B. Given D, the node derives the Merkle root m, adds f and t, and calculates the block identifier id, which is signed by the author. The block identifier is then embedded in the web page as a corresponding HTML tag.

If an author cites an article which is part of the Webchain ecosystem, a respective reference has to be added to D. In this case, the client software automatically detects and includes Webchain metadata when copy-pasting from a web site that is part of the Webchain system. The reference, in form of a block identifier, is also included as an HTML attribute. An example can be seen in Figure 11.3.

When citations are translated to block references, so-called *citation chains* emerge. Moreover, since various blocks can reference the same block, multiple citation chains can form a *citation graph* (see Figure 11.4), which is basically a directed acyclic graph (DAG). Of course, this graph does not need to be fully connected, and single articles which do not cite others even yield connected components of size one. Also note that many independent citation graphs can coexist as blocks are not guaranteed to reference prior blocks. This is especially noteworthy, as it differentiates the Webchain design clearly from other blockchain-based approaches: the Webchain system does not host a single ledger of blocks, but many ledgers that only have to be stored by the involved parties, which follows the separation of concerns principle.

Blocks are announced in a peer-to-peer network of infrastructure nodes. Every node receiving a new block verifies its validity before forwarding the block to its neighbors. If one of the node's registered authors is associated with the block, i. e., a path between a block of the author and the new block exists, the block is replicated by the infrastructure node. Otherwise, the block is discarded after forwarding.

11.3.4 *Block Verification*

When infrastructure nodes receive a previously unknown block $B_0 = (id_0, f_0, t_0, m_0, \sigma_{id_0}, D_0)$, they perform a number of checks to verify its *validity*:

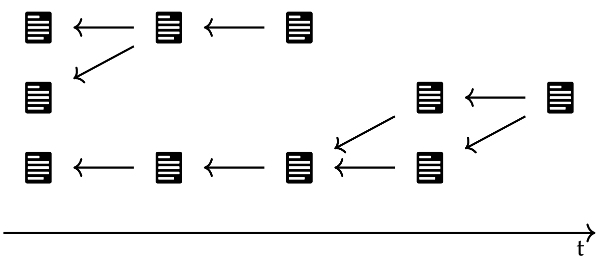


Figure 11.4: Citation graphs over time.

1. Construct a Merkle tree from D_0 with its root m'_0.
2. Check that the Merkle roots match: $m_0 = m'_0$.
3. Check the block signature: $id_0 = \mathit{verify}(\sigma_{id_0}, K_{a_0,pk})$.

Additionally, when the nodes are involved with the block's citation graph, they inspect D_0 and recursively retrieve all referenced blocks B_i, $i \in \{1 \dots N\}$, i. e., they traverse the citation graph. This enables them to verify the *citation provenance*, by running additional checks:

1. Follow all references $d_j \in D_i$, $i \in \{0 \dots N\}$ and assert that d_j is indeed included in the referenced block.
2. Assert that all referenced blocks are *valid* by running the validity checks described before.

Therefore, a block is considered *valid*, when the integrity of the data is confirmed through use of hashing and the Merkle tree construction. Furthermore, *citation provenance* is ensured by following the cryptographically secure links between blocks and verifying the existence of data items in the cited sources.

The client software checks the accessed websites for Webchain metadata. When the client detects such metadata, it extracts the block identifier and retrieves the corresponding block B_0 and referenced blocks B_i from an infrastructure node in the background. The client applies the segmentation and normalization algorithms to the article and applies the hash function to the text segments. If included, it also extracts the citation information. Based on this data, the client constructs its own local data set D'_0. In order to verify the article's content, the client proceeds with the following steps:

1. Calculate Merkle root m'_0 from D'_0.
2. Assert that m'_0 and the retrieved block's m_0 match.
3. Assert the *validity* and *citation provenance* of the retrieved blocks B_i, as described before.

Given that blocks can only reference other blocks that already exist, it should be the case that timestamps in each individual citation chain follow a certain pattern. In particular, when following an individual link from a citing block B_i to a referenced block B_j, it should hold that $t_j < t_i$. However, this property can only be guaranteed to hold, if nodes have a meaningful way to verify timestamps for a specific block, which then is approved or discarded accordingly.

So far, we neglected to describe how infrastructure nodes are able to verify the timestamp information provided in the blocks. As a naive sanity check, every node receiving a new block could verify that the given timestamp is close to its system time and that it is larger than the timestamps of all referenced blocks. However, this method has several drawbacks. For once, it highly depends on the correctness of the local clock and could thereby lead to unnecessary divergent ledger states in the network. This verification method is also only viable for recently authored blocks, making it unclear how to handle timestamps of retroactively retrieved blocks. Moreover, this scheme would not prevent an author from manipulating the timestamps and embezzling blocks. Webchain addresses this issue by implementing a distributed timestamping scheme, which is described in the following section.

11.4 DISTRIBUTED TIMESTAMPING

Timestamping services offer their users to issue a proof of existence, which states that a certain data item existed at a specific point in time. The service is most commonly realized by a centralized trusted timestamping authority that issues such an attestation. However, we consider this model to be unsuited for the otherwise distributed design of the Webchain system. Therefore, timestamping in Webchain builds upon the distributed timestamping scheme introduced by Haber and Stornetta in [HS91].

Distributed timestamping is based on the idea that a user should—rather than placing trust in a single centralized authority—distribute the trust to a whole network of timestamping authorities, which we call *timestamping validators*. By cryptographically signing hashed data together with a timestamp, these validators attest that they have seen the data at a specific point in time. Of course, it would immensely reduce the security properties of the timestamping protocol, if an attestation would no longer be required to come from a specific validator, but could stem from any of the validators. Therefore, Haber and Stornetta introduced the idea of deterministically electing k designated timestamping validators based on the content of the data itself, k being a global constant. This is done by using the hash y of the data (which is in our case $y = id$) as seed for a pseudorandom number generator

(PRNG) function G, whose output sequence directly determines the k designated validator nodes as

$$V_y = \bigcup_{i=1\ldots k} G(y, i)$$

The hash y is sent to each of the selected validator nodes V_y, which respond with a digital signature, effectively creating a timestamp.

Verification of these timestamps is done as follows: the verifying party hashes the data to gain y and calculates k executions of G to reconstruct the set of designated validators. The data is considered valid, if all of the determined validators have signed the input. Thereby, the security of this approach does not directly come from increasing the number of validators (this only distributes the trust), but from the deterministic validator election. For this, it is assumed that a cryptographically secure hash function is used (i. e., the probability of collisions can be neglected), and that it is hard to spawn validator nodes with arbitrary identifiers.

In order to adapt this scheme for the Webchain system, we assume that every infrastructure node also functions as a timestamping validator and holds a public-private key pair specifically for attestation purposes. Validators are addressed by its fingerprint $v_i = H(pk_i)$ with pk_i being their public key. When new blocks are distributed in the network, every node, including the validators, are able to calculate the set of designated validators V_{id} for a block identifier id. This of course also applies to the designated validators, which issue a corresponding attestation upon reception of a block they are responsible for. The validator's attestation is again broadcast in the network, and stored replicas of the blocks are updated by appending the additional signatures to the field σ_{id}. Blocks are considered *valid*, if they not only feature the author's signature, but also the k validator signatures.

However, deploying the scheme found in [HS91] is not entirely feasible for the Webchain system: based on the construction, it can be assumed that the authors required a network of static nodes, which however does not fit the dynamic design of Webchain's network. In Webchain, we therefore propose an extension that manages the network state in a dynamic overlay of validators.

11.4.1 *Validator Membership Management*

While we can assume that Webchain infrastructure nodes are relatively stable, the network still is of dynamic and open nature, which requires a flexible approach for addressing and determining the designated validators. We address this issue by—instead of directly using the output sequence of the PRNG—introducing an addressing scheme based on the XOR metric known from Kademlia [MM02]. This non-Euclidean metric assigns a distance to two binary values, by applying the XOR operation

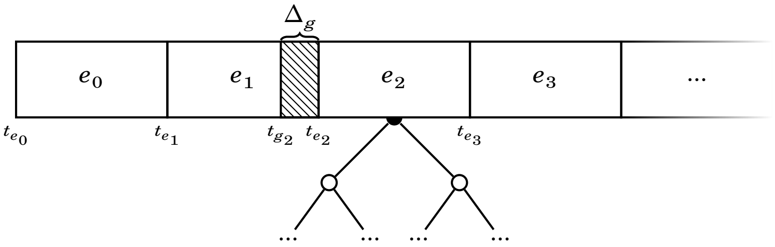


Figure 11.5: Epochs define time intervals in which a specific validator set is considered active. The validator set is secured by a Merkle root. The grace period Δ_g leaves room for data propagation but also implies, for example, that valid announcements for e_2 need a timestamp dated before t_{g_2}.

and interpreting the result as a integer number, i. e., $d(x, y) = (x \oplus y)_{10}$. By using the XOR metric, Webchain can assign a well-defined distance between each value from the sequence obtained by G and all existing validator nodes. This allows to determine the k designated validators, i. e., V_y, by selecting the k node identifiers *closest* to the values obtained from G. However, since the deterministic validator election scheme not only depends on the data input, but also on the validator set, a node verifying a block has to know the state of the validator network *at the time of the block's creation*.

In order to manage the network state, we divide time into *epochs*. In each particular epoch, the state of validators s builds the basis for the respective validator election. We denote such an epoch as $e_i = (i, s)$, including a continuous epoch number $i \in \mathbb{N}$ and a value representing the state s. When joining the network, every validator starts announcing her participation for a future epoch. Every infrastructure node keeps track of these *announcements* and thereby keeps a list of candidate validators active for a specific epoch i. All infrastructure nodes should reach consensus on which validators are considered active in a specific epoch. However, depending on a variety of factors (e. g., network performance), an announcement could reach some nodes in time to register for a specific epoch, while it may reach others too late. To mitigate this consistency problem, we use the timestamping solution itself: other validator nodes sign announcements and broadcast the attestations in the network. Similar to blocks, announcements are only considered valid if they are signed by k designated validators and received before the epoch starts. This is, announcing nodes are then deemed active in the respective epoch.[2]

In order to deal with network delays, infrastructure nodes require that announcements have a timestamp t_a that is dated at least some time Δ_g earlier than the start time t_{e_i} of the respective epoch e_i. Therefore,

---

[2] Note, that this introduces a bootstrapping problem: the announcements for the initial validator set cannot get attested by a previous set of validator nodes. Therefore, we assume a set of bootstrapping nodes for e_0 to be hard-coded in the Webchain software.

$\Delta_g = |t_{e_i} - t_{g_i}|$ defines a grace period, which is illustrated in Figure 11.5. The grace period is introduced to ensure that "fresh" announcements and the related attestations reach all nodes in the network, before the start of the new epoch. Therefore, we think t_{g_i} should be chosen so that Δ_g is at least double the maximum expected round-trip time (RTT) in the network, i. e.,

$$\Delta_g > 2 \cdot RTT_{max}.$$

For practical considerations, a Δ_g of around 500 ms should suffice for most network topologies. From the list of active validators, every node builds a sorted Merkle tree, yielding a root value which succinctly and securely describes the membership state of this epoch, denoted as s. When new blocks are created, e_i is appended to the block and secured by the identifier hash value, now respectively denoted as $B = (id, f, t, m, \sigma_{id}, e_i, D)$ and $id = H(f \| t \| m \| e_i)$. The introduction of epochs allows to retroactively verify that a block was signed by the correct set of validators. As a consequence, joining nodes have to retrieve the historical epoch data needed for verifying a particular block. As the introduction of epochs limits the time a specific set of validators is designated for a given input, it also increases the system security (see Section 11.6).

11.4.2 *Validator Redundancy*

The described timestamping scheme allows to determine which validators are responsible for signing a specific data item in any given epoch. However, what happens if a designated validator becomes unavailable or maliciously refuses to issue attestations? As infrastructure nodes only accept new blocks and announcements when they are confirmed by k designated nodes, refusing to attest could be an easy way to run a Denial-of-Service (DoS) attack on the Webchain system. Fortunately, there is an effective way to mitigate such attacks in form of validator redundancy: infrastructure nodes do no longer check for k attestations of designated validators, but they check for a number

$$k' < k.$$

This solution was already proposed by Haber and Stornetta and in our case also helps to handle validator node failure: when a validator announces its participation in an epoch, but then fails to attest a block or announcement, the block is still valid, if at least k' validators are still online.

Leaving a certain degree of freedom and accepting a smaller number of attestations, though, inevitably reduces the security level. As we will see in our analysis, it increases an attacker's chances to contribute the set of designated validators, i. e., to maintain the security level, a higher k should be chosen.

11.4.3 *Validator Replication*

As described before, infrastructure nodes replicate all citations graphs which are connected to any articles created by their authors. But, this also implies that only parties with some interest in the cited articles are storing the data. In particular, since they would be the only replicating nodes, this would enable malicious infrastructure nodes to remove old uncited articles from the network. To mitigate such an after-the-fact manipulation and to guarantee *non-repudiation* for articles in the Webchain system, we introduce validator replication. This is, in addition to the respective infrastructure nodes, the set of designated validators also replicate the attested blocks. Therefore, the block data is replicated k more times in the network, depending on its content. This ensures that an article that is once inserted into the Webchain system cannot be removed afterwards.

11.5 SYSTEM EXTENSIONS

In the following, we discuss a number of extensions to the basic Webchain system that are indispensable for a practical implementation of its architecture.

11.5.1 *Identity Management*

The security properties of the Webchain design heavily rely on the ability to retrieve and validate authentic key material for a given node identity. Yet, authors may register and publish under a pseudonym, as nodes are responsible for handling their associated author identities.

In the following, we briefly discuss candidates for node identity management in Webchain:

HIERARCHICAL PUBLIC KEY INFRASTRUCTURE (PKI) The hierarchical PKI is a time-tested concept known all too well from the X.509 Internet Public Key Infrastructure [Coo+08]. However, utilizing a hierarchical PKI scheme would require Webchain to rely on a rather centralized and out-of-bound infrastructure that follows its own trust model. While it may be a robust choice for the identity management in Webchain, it contradicts Webchain's distributed approach.

WEB OF TRUST (WOT) In contrast to the hierarchical structure of a typical PKI, the concept of a Web of Trust allows users to infer trust in an identity-to-key binding in a completely decentralized manner: users validate and sign the identities of other users and thereby publicly state their trust in an identity binding. This statement creates an edge in a graph of trust relationships between users themselves. This graph can then be used to create a trust relationship regarding a specific identity binding by traversing over multiple edges in the trust graph. Therefore, the web of trust allows to validate identities by inferring trust from other trusted users' verdict. While the fully decentralized nature of this model seems promising, it has proven to be a rather inconvenient and hardly ever used solution in practice (e. g., in the web of trust of OpenPGP [Cal+07]).

DNS-BASED AUTHENTICATION OF NAMED ENTITIES (DANE) Another well-known method for managing the mapping of public keys to identities is DANE [HS12]. In DANE, X.509 certificates are attached to respective entries in the domain name system (DNS) and secured via DNSSEC [Are+05]. This allows to distribute certificate information without the need for a central certificate authority (CA). Additionally, bindings for storing user public keys are currently explored for the use-case of OpenPGP public keys [Wou16]. When Webchain infrastructure nodes—as it is usually the case on the World Wide Web—are already addressed through DNS records, we consider DANE an appropriate candidate for identity management that can be easily integrated in the Webchain architecture.

DECENTRALIZED IDENTITY MANAGEMENT SYSTEMS In recent years, a number of identity management systems emerged that foster a more decentralized trust model. For example, the CONIKS [Mel+15] system provides end-user verifiable identity-key bindings and could be nicely deployed in conjunction with Webchain infrastructure. However, other proposals, such as Namecoin [@Nam19], Certcoin [FVY14], and Decentralized Anonymous Credentials [GGM14] build on existing blockchain technologies to create a distributed ledger of identity mappings. While generally suitable in regard to its decentralized trust assumptions, utilizing these systems would introduce a lot of additional complexity and external dependencies to the Webchain design.

11.5.2 *Versioning*

Given the Webchain design, we also support the ability to publish revised versions of an article right out of the box. To this end, we extend the block layout by a new field p, which points to the previous block version. This block reference p either can be left blank if the block is the first instantiation of this article, or set to the block identifier of the previous block. When propagating this new version of a block in the network, infrastructure nodes need to check that the new block is signed by the same author as previous versions, but otherwise treat the block as usual. Since blocks are broadcast, all infrastructure nodes replicating the previous version will eventually also get and replicate the new block. They may choose to notify citing authors that a new version exists.

11.5.3 *Peer-to-Peer Network*

As similar distributed ledger technologies, Webchain is built on a broadcast mechanism: every block, attestation, and join announcement is propagated to every infrastructure node, which then verifies the item according to its capabilities. This approach typically assumes an unstructured peer-to-peer network model in which every participating node establishes connections to a random subset of neighbor nodes. While such networks are known to be relatively robust, broadcast is an inefficient operation in this kind of networks. In proof-of-work blockchain networks such as Bitcoin, delayed and inefficient block propagation can have severe consequences on consensus stability [Ger+16]. However, as Webchain uses a deterministic validator election, it does not face the same challenge. In order to further reduce the effect of network-induced delays, Webchain also introduces the aforementioned grace period Δ_g that allows for network synchronization. Our basic prototype design therefore implements an unstructured peer-to-peer network for block propagation, while the Webchain systems may overall profit from a more efficient and tunable approach, such as introduced in Chapter 5.

Moreover, the combination of deterministic validator election and validator replication allows for a trivially improved procedure for looking up specific data items in the peer-to-peer network: assuming every infrastructure node does not only keep track of the validator identifiers, but also of the respective IP addresses, it directly knows which node to query for a specific data item. This method is in line with the idea of distributed hash tables (DHTs), like Chord [Sto+01] or Kademlia [MM02]. By trading space complexity for lookup complexity, the algorithm allows for a $\mathcal{O}(1)$ lookup operation. While out of scope for the present work, we think this extension to the Webchain protocol is a promising direction for future research.

11.5.4 *Steps Towards Deployment*

As in recent years deliberate spreading of disinformation and reference rot became more prevalent issues on the Web, authors and publishers have an increasing incentive to increase readers' trust in online publications. To this end, Webchain is a useful tool that provides individual authors (e. g., bloggers and citizen journalists) as well as institutions (such as news agencies and news sites) with the capabilities to publish independently verifiable and authenticatable articles on the Web.

The Webchain prototype already offers a rudimentary web front-end allowing for the editing and publication of articles. The interface is based on Vue.js [@Vue20] and serves as an initial proof-of-concept. It particularly shows that the Webchain architecture is able to integrate with basic web technologies and content management systems, such as Wordpress [@Wor19]. Many pre-existing publishing platforms and news outlets could therefore introduce support for the Webchain system without much overhead, especially since the multi-ledger design does not require all participants to process and store every piece of data. Hence these platforms have an incentive to run Webchain nodes themselves, which is why we envisage that the core network deployment would be made up by such hosted infrastructure nodes. However, since Webchain is conceptualized as an open network and does not require trust in particular nodes, enthusiasts could also opt to host their own nodes, thereby contributing to the network's resources and further decentralization.[3]

In the following, we discuss the security properties of the Webchain protocol and investigate what magnitude of deployment is necessary to ensure secure timestamping even in the face of powerful adversaries.

11.6 WEBCHAIN SECURITY

In the following, we discuss security-related properties of the Webchain system and analyze the resilience of the timestamping algorithm in the face of an attack.

11.6.1 *Security Properties*

Webchain provides the following security properties:

CONFIDENTIALITY, INTEGRITY Article content is not stored in the Webchain system and only included as hashes, thereby keeping it *confidential*. Hashing is also used for the construction of a block's identifier and Merkle root, which ensure the *integrity* of the given information.

AVAILABILITY, AUTHENTICITY The *availability* of blocks and state data is ensured through the validator replication scheme. The use of cryptographic signatures makes sure that the *authenticity* of infrastructure nodes and authors can be validated. As discussed, this also depends on the presence of trustworthy identity management scheme.

NON-REPUDIATION, CITATION PROVENANCE Furthermore, the combination of signatures and append-only replication allows everyone to prove that a certain block was previously created by a specific author. This proof of existence does not only yield *non-repudiation* of blocks, but also *citation provenance*.

---

[3] Note that while still capable of being run side-by-side with traditional Web server infrastructure, the current Webchain prototype implementation (see Section 11.7.1) already goes one step further and is able to run as a independent infrastructure node following peer-to-peer principles.

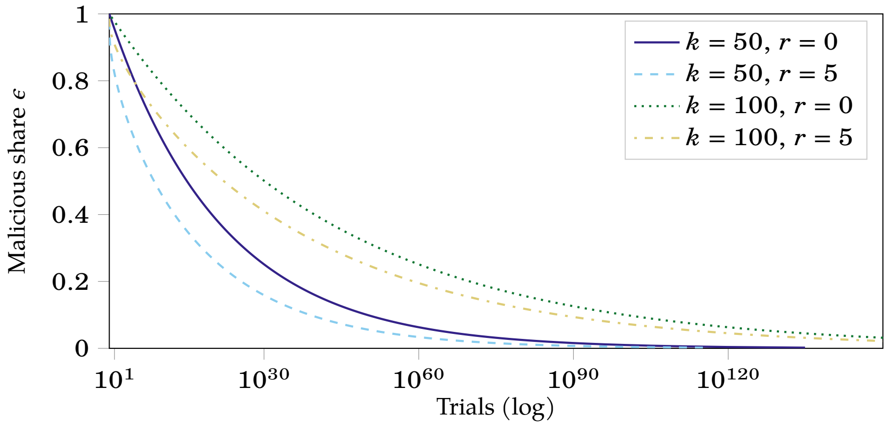


Figure 11.6: Number of trials needed for an adversary to find an input for the PRNG that yields a designated validator set contained in her fraction ϵ of controlled nodes.

11.6.2 *Timestamping Security Analysis*

In the following, we analyze the properties of the timestamping algorithm in case of an attack. We assume an adversary trying to construct a block with a forged timestamp. The adversary can spawn additional infrastructure nodes and hence controls a fraction ϵ of all active validator nodes in a given epoch.[4] In order to succeed with forging a timestamp, the adversary can only try different combinations of input data, e. g., by adding or varying certain parts of the content, until the deterministic validator election yields k validators which are exclusively controlled by herself. As every run of the pseudorandom generator function is statistically independent, the probability of choosing k malicious validators is $p_k = \epsilon^k$. Assuming a uniform distribution of hash values, the number of trials before succeeding can be estimated as p_k^{-1}. However, the distribution changes when we introduce validator redundancy r: in case of allowing for a subset of at least $k' = k - r$ attestations, the probability for a successful attack can be calculated as

$$p_{k'} = \sum_{i=k'}^{k} P(X = i) = \sum_{i=k'}^{k} \binom{k}{i} \epsilon^i (1-\epsilon)^{k-i}.$$

Section 11.6.2 shows the expected number of necessary trials for a varying fraction of malicious active validators ϵ and $k \in \{50, 100\}$ designated validators. For example, even when controlling a fraction $\epsilon = 0.5$ of all validators, an attack is only possible after approximately $1.13 \cdot 10^{15}$ and $1.26 \cdot 10^{30}$ trials for $k = 50$ and $k = 100$, respectively. Moreover, it is shown that redundancy significantly weakens security:

---

[4] Note that depending on the identity management system, rate limiting node registrations could mitigate spawning a large number of nodes.

when allowing $r = 5$ redundant validators (i. e., $k' = k - 5$), the number of trials is approximately $4.75 \cdot 10^8$ for $k = 50$ and $1.59 \cdot 10^{22}$ for $k = 100$. As each trial involves calculating a number of hashes, these numbers may be compared to a proof-of-work process, e. g., mining in Bitcoin [Nak08]. As of March 2018, the biggest mining pool, BTC.com has an aggregated hash rate of around $6 \cdot 10^{18} H/s$ [@BTC18]. When we assume these efforts to be equivalent, this would mean that the biggest mining pool in Bitcoin could instantly run an attack on Webchain with $k = 50$, given that it controls more than 50% of infrastructure nodes. However, for the case of $k = 100$, it would take such an powerful adversary a much longer period of time, that is, approximately $2.10 \cdot 10^{11} s$ or around $6,660$ years. Then again, when redundancy is introduced, this shrinks to around 45 minutes.

Clearly, there is a trade-off between the security and the reliability of the system. Therefore, we further investigate this trade-off to infer parameters rendering the forging attack infeasible, while still providing an adequate level of reliability. Since the attacker has to find an appropriate input value before the underlying state changes at the start of the next epoch, the time for running the attack is limited by the epoch length. Therefore, by choosing an epoch length that is sufficiently lower than the expected time needed for the attack, based on a given network size k and a desired redundancy factor r, this attack can be mitigated.

However, the shorter the epoch, the more epochs are created, hence the higher the overhead in terms of space and message complexity. Therefore, we propose to fixate the epoch duration at a reasonable default value, e. g., in the order of minutes to hours, and adjust r with respect to a given network size k. In the worst case, assume an adversary with comparable computational capabilities as BTC.com and in control of 50% of active validator nodes. Based on this adversary model, Section 11.6.2 shows the average time needed for a successful attack, depending on the parameters k and r. Again, it can be seen that the redundancy factor has a big impact on how fast the attacker can conduct the attack. For example, if we assume a Webchain network of just $k = 200$ infrastructure nodes and a redundancy factor of $r = 20$, an attack would still take more than 4,685,000 years, well above any reasonable epoch value. If we assume a network of $k = 150$ nodes and $r = 10$, timestamps would be still secure, since an attack would take around 5,996 years. And, as mentioned before, for $k = 100$ and $r = 5$, the system would still hold for approximately 44 minutes, which may be considered insecure for an epoch length of 1 hour. Then again, given we are considering the worst-case, five redundant validators would probably still be fine for an epoch length of 10 minutes.

While the epoch length and r should be adapted based on the requirements experienced in practice, it is expected that—if gaining traction—the Webchain network would easily reach 100 nodes before drawing the

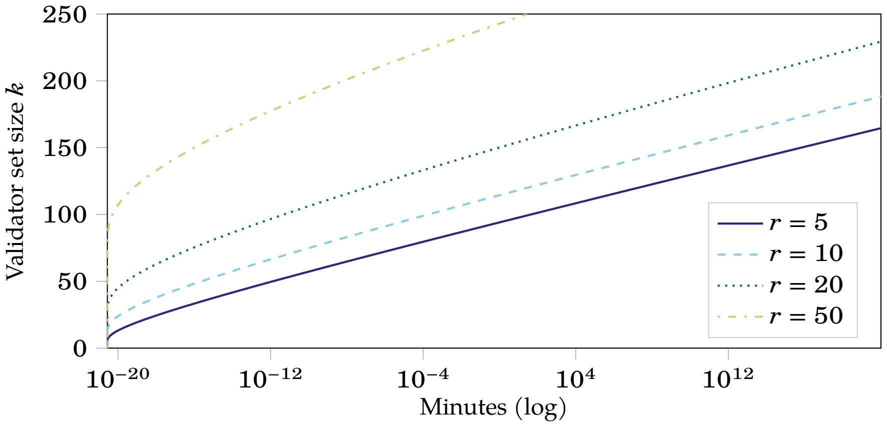


Figure 11.7: Average time needed for a successful attack run by an adversary comparable to the largest Bitcoin pool, depending on replication parameter k and redundancy parameter r, assuming $\epsilon = 0.5$.

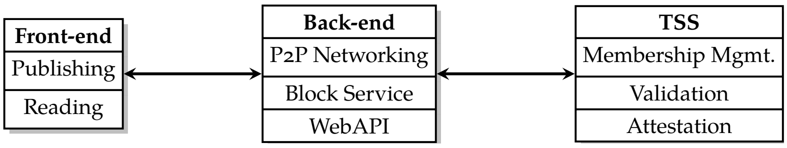


Figure 11.8: Overview of implementation building blocks.

interest of a powerful attacker. Therefore, we propose an epoch length of 10 minutes and $r = 5$ as default parameters for the Webchain system.

11.7 EVALUATION

In the following, we introduce a prototype implementation and evaluate Webchain's properties regarding resilience to node failures and attestation delay.

11.7.1 *Prototype Implementation*

In order to proof the feasibility of the Webchain architecture, we implemented a fully working prototype application consisting of approximately 6,000 lines of source code.[5]

The back-end and timestamping components are implemented in the Go programming language [@Lan19] and utilize the LibP2P [@Sta19] networking stack for basic peer communications. The author-facing front-end is implemented as a browser-based web application that communicates and interacts with the distributed ledgers via an RESTful API

---

[5] The source code and evaluation data are publically accessible at `https://git.tu-berlin.de/rohrer/webchain-public`.

provided by the back-end component. Section 11.7.1 gives an overview of the implementation building blocks.

The back-end's block service provides ledger-related functionality, for which ECDSA cryptography based on Curve25519 [Ber06] and SHA2-256 [FIP15] are employed. It can retrieve and verify blocks and timestamps from the peer-to-peer network, as well as create new blocks based on input from the WebAPI.

Timestamps are generated and validated by the timestamping service (TSS). In order to get timestamps for new blocks, designated validators are chosen according to a random number generator based on the ChaCha20 [NL15] algorithm. Note that the TSS itself is generally independent of the block service implementation and can therefore issue timestamps for arbitrary data. This allows for extended modularity, as it only requires an adequate peer-to-peer module for membership management functions, as well as the transmission and retrieval of timestamping requests.

Users interact with the system via the web front-end, which interacts with the WebAPI provided by the back-end component. The WebAPI delegates all requests to the appropriate component and does not implement any business logic on its own. In addition to the interaction with the citation system, it also exposes information about the underlying peer-to-peer network. Users can view the address and public key of the infrastructure node they are connected to, as well as information on the neighbor nodes in the network.

The peer-to-peer module implements the aforementioned unstructured peer-to-peer overlay network to enable communication between infrastructure nodes. In order to bootstrap the network, a list of initial seed nodes are deployed with the prototype software. After connecting to one or more seed nodes, joining nodes broadcast their own address as well as a *getPeers* message. As an answer, they receive a list of known nodes from the seed nodes. From this list, they randomly select 8 nodes as their neighbors and store the other addresses for later use. If message delivery to one of the neighbor nodes fails, nodes remove this neighbor and try to connect to a new one, either from the stored addresses or from new *getPeers* broadcast. To be able to quickly replace inactive neighbors, the nodes broadcast *getPeers* messages in regular intervals and store addresses they received over the network.

Based on this overlay network, the prototype implements a basic broadcasting mechanism: in general, the network nodes forward all received broadcast messages to each of their neighbors. However, they keep track of seen messages and avoid duplicate transmissions. As each the broadcast is meant to reach all nodes in the network and should do so as fast as possible, we did not employ a time-to-live (TTL) counter or similar limiting mechanism for the propagation of information. Nevertheless, the prototype enforces a random delay for announcement messages to reduce the load at epoch boundaries. To this end, instead

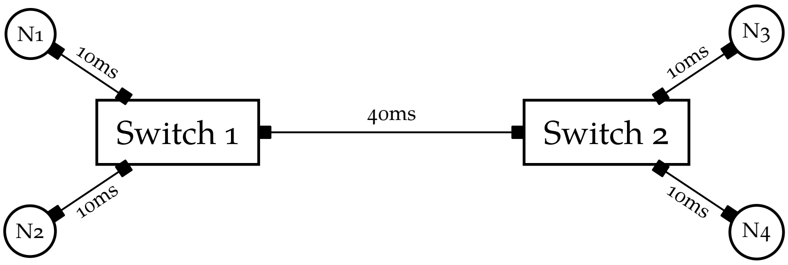


Figure 11.9: Example Mininet topology with four infrastructure nodes N1,..., N4.

of immediately broadcasting their announcements at the start of an new epoch, nodes choose a random delay bounded at the *maximum announcement delay*. This is necessary, because otherwise all participating nodes would broadcast their announcement at the same time, causing an high peak load. This would be especially detrimental in an emulated environment, where clocks are fully synchronized.

As described before, infrastructure nodes store a public-private key pair used in the timestamping component. However, in our research prototype implementation we opted not to predetermine a specific identity management solution and hence chose to address and identify validators by their full public key. While this does not provide an immediate trust relationship, it follows a trust-on-first-use model and allows us to verify timestamping signatures without the need to resolve identifiers to public keys.

The prototype implementation furthermore requires consistent epoch length values in the whole network and assumes that the clocks of all nodes are roughly synchronized in order not to be marked as failed during validator election. As mentioned before, Webchain has a bootstrapping problem: if no epoch state is available, no set of validators can be chosen to sign and confirm timestamps. Because the announcements for participation also have to be timestamped, no valid announcements are possible in this state. This is the case for the first epoch, but also when no, or less than k', nodes announced their participation for any epoch. Therefore, the prototype implementation provides an initial set of fallback validators. If no valid announcements for an epoch are available, the epoch state is build from this set of bootstrapping peers, as if they had announced their participation for the bootstrapping epoch. From this point on, we require all bootstrapping peers to announce their participation for the next epoch.

11.7.2 *Evaluation Setup*

In order to evaluate the Webchain prototype implementation, we created experimentation scenarios using the Mininet [LHM10] network emulator. Mininet allows for the creation of virtual network hosts and switches based on the standard Linux networking stack and utilizes the OpenFlow [McK+08] software-defined networking (SDN) protocol, resulting in highly realistic networking environments.

In order to recreate a networking setup capturing long distance (e. g., transatlantic) connections, we created a network topology following the dumbbell model: it features two interconnected switches which are themselves connected to a number of associated nodes that run the Webchain prototype implementation. Latency-wise, we provision this dumbbell topology with a large link delay between the two central switches (40 ms) and a relatively small delay (10 ms) set for the links connecting the infrastructure nodes themselves. Section 11.7.2 shows an example of the topology with four nodes. This setup results in round-trip times (RTTs) of approximately 120 ms for long distance connections and roughly 40 ms for intracontinental connections.

Since Mininet emulates the network based on the actual application code, all our measurements are conducted based on real time. Therefore it is of utmost importance to not introduce additional processing and network delays by overloading the host machine running the evaluation. Since both, the Webchain protocol and the underlying peer-to-peer network stack utilize timeouts in various parts, artificial delays would lead to unpredictable errors in our measurements, which could possible distort the outcome of the evaluation. To avoid overloading, we never fully utilize the capacity of the host machine. We furthermore limit the CPU capacity every node is allowed to use and monitor the load while running the evaluation.

Apart from the total number of nodes, the number of required validators for each timestamp is a major limiting factor for our evaluation, because it directly increases the number of created and verified signatures, as well as the amount of necessary network traffic. While we do not evaluate the validator performance in detail, this is backed by two observations. First, while the cryptography used in the implementation is considered fast (see [Ber08; Ber+11]), profiling shows, that nodes spend a significant amount of time creating and verifying signatures. Second, the monitored total load of the computer increases with the number of required signatures and the total number of nodes. In a real world deployment, those calculations would be done in parallel on a large number of infrastructure nodes. While in our case a new process is started for every node that allows for a certain degree of parallelism, the extend of parallelism cannot easily be emulated on a single computer, especially considering Mininet's limited worker distribution capabilities. Given these limitations, running on an Intel Xeon Processor 5130 with 32GB RAM, we can safely evaluate the Webchain system running 20 nodes with $k = 10$ validators.

Since it is mandatory to sign timestamps by a large number of validators, we need the Webchain system to be deployable in large scale. Therefore, the system must be resilient against failing or malicious

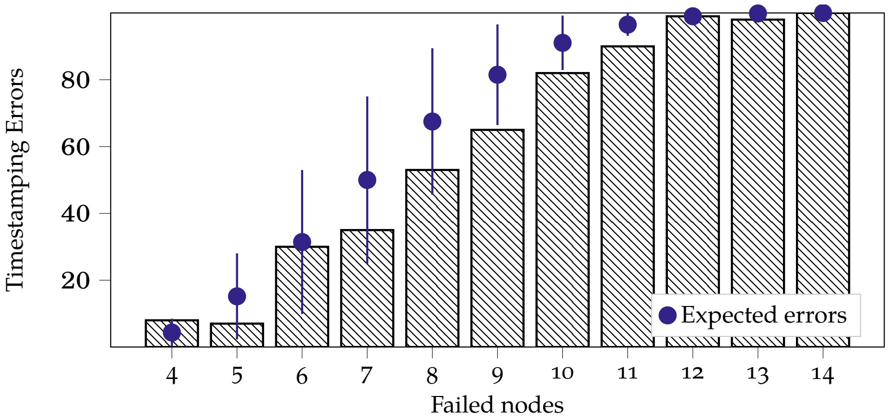


Figure 11.10: Timestamping errors in dependence of the number of failed nodes. For comparison, error bars show the expected number of errors and the associated variance.

nodes and at the same time be fast enough to be usable by a broad user base. In the following, we therefore analyze how Webchain deals with failing nodes given a configurable validator redundancy and what magnitude of attestation delay we can expect from the timestamping component for different values of required validator signatures k.

11.7.3 *Resilience to Node Failures*

In order to secure the Webchain system against unavailable nodes and Denial-of-Service attacks, we introduced the concept of validator redundancy. Without redundancy, a single unavailable or malicious validator could cause an otherwise valid timestamp to fail, resulting in a timestamping error. With validator redundancy, a timestamp is valid if it was signed by $k' = k - r$ instead of k validator nodes, which allows for r failed validators. Note however, that even without redundancy unavailable nodes do not necessarily cause a timestamping error in every case, but only for the timestamps for which the unavailable nodes are chosen as designated validators.

To evaluate whether the prototype implementation matches the expected behavior, we parametrize the system with an increasing number of failing nodes and compare the number of failing timestamps to the expected failure rate. To simulate a node failure, we deactivate the Mininet link between the node and its switch, thereby removing it from the topology. In order to determine the expected number of timestamping errors, we calculate the probability of a failed timestamp as well as the corresponding expected value and variance for the total number of timestamps tried. With redundancy, a timestamping error is only possible if the number of chosen but failed validator nodes exceeds r. Defining X as the number of chosen but failed validator nodes, the

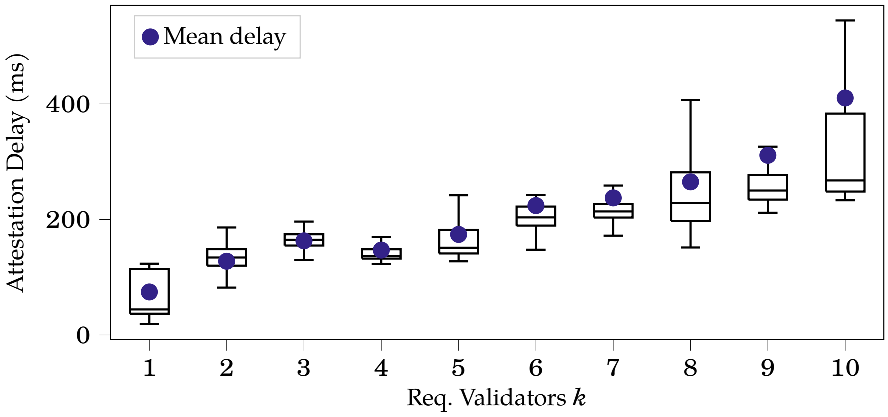


Figure 11.11: Attestation delay depending on the number of required validator signatures k.

probability of a failure $P(X > r)$ is following a hypergeometric distribution, i. e., $X \sim Hyp(N, F, k)$ with N being the number of participating infrastructure nodes and F number of failed validator nodes.

Section 11.7.3 shows the number of timestamping errors for an increasing number of failed nodes compared to the number of expected errors.

For every number of failed nodes, we timestamped $n = 100$ random block identifiers with $k = 10$ and $r = 3$. We therefore only show results for $F \geq 4$ and until every timestamp fails with high probability ($F = 14$). Furthermore, an error is recorded if not all required signatures are received by our measurement node within a timeout period of five seconds, which is of course significantly larger than any expected and observed delays in the network.

In Section 11.7.3, the number of errors is charted as bars, while the expected errors and their variances for n timestamps are shown as dots and whiskers. We can see that the implementation behaves as expected, as the number of errors never exceeds the bounds of the error distribution variance. Moreover, the number of errors is lower than expected in almost every case, indicating that Webchain's resilience to node failures is exceeding the expectation.

11.7.4 *Attestation Delay*

Every new block in the Webchain system has to be timestamped before it is accepted by other nodes. The delay introduced by the attestation process is therefore a major factor for the time it takes to publish new blocks, which is an important part of the scalability and usability of the system. It also determines the possible resolution of the timestamping system, since a long delay limits the time interval in which a timestamped block can be accepted and signed by the chosen validator nodes. The attestation delay depends on the propagation delay and the time of cryptographic operations. Both of these factors are however determined by the number of required validators k. Also, choosing a large k value allows for longer epoch times, since the epoch time limits the time available for an brute-force attack on the system, as discussed in Section 11.6.

We therefore evaluate the attestation delay in dependence of different values of k. In order to avoid interference from other timestamps and validator announcements, we base the delay measurements off a single epoch. Because the implementation returns a timestamp as soon as $k - r$ signatures are available and we are interested in the time we need for k signatures, we disable validator redundancy, i. e., $r = 0$. For every value of k, $n = 100$ random block identifiers were timestamped and propagated in the network. We verified that every timestamp was successful by validating them after the measurements were finished. Since we evaluate with $r = 0$, real-world delays are expected to be lower in practice.

Because every time the block identifier and k signatures are disseminated in the network, $k + 1$ broadcast operations are expected. of which k broadcasts can be done concurrently after receiving the block identifier. Because of the concurrency of the broadcast, we expect the increase in delay per additionally required validator to be subadditive. Note however, that in contrast to a real-world deployment, the evaluation setup cannot provide the same level of concurrency since it is limited by the number of provisioned CPU cores and hence limits the CPU time for each Webchain instance. This is especially important for calculating and verifying signatures. For the evaluation we can therefore expect an steeper increase in delay when the number of required signatures exceeds the number of CPU cores available.

Section 11.7.4 shows the evaluation results for $k \in [1, 10]$ required validators. The boxplot depicts the median value and the interquartile range of measured delays, as well as maxima and minima. For comparison, mean values are added as dots. Note that the measurements confirm our expectations: for values of k smaller than the number of available CPU cores, the additional delay increases slowly. This is followed by a higher increase in delay for every k after that. This can also explain the much wider range of delays we observed for values of k exceeding the number of CPU cores: if the CPU load gets too high, there is a higher probability for a node to get stalled, resulting in higher response times.

As discussed in Section 11.6, in order to resist large-scale forging attacks, Webchain should be able to scale to $k \geq 100$. Therefore, the observed subadditivity property is an important property of the system. While our evaluation confirms this property for small values of k, it also shows the limits of our evaluation setup. In order to circumvent such effects, future experiments should be deployed on a testbed of peer-to-peer nodes.

12

CONCLUSION

In this thesis, we studied blockchain scalability from a computer networking perspective.

In the first part, we researched the performance and reliability of the networking layer and showed that it is deeply interconnected with the stability and security of the consensus layer. In Chapter 3 and Chapter 4, we conducted longitudinal measurement studies on real-world peer-to-peer networks and created simulation models that enable the in-depth analysis of these complex infrastructures. Moreover, we proposed improvements to the way blockchain networks propagate block and transaction data in Chapter 5. To this end, we could show how a decentralized but structured approach to data dissemination can help to increase the efficiency and performance of the networking layer.

In the second part, we studied payment channel networks and showed that their performance, security, and privacy properties are closely correlated with the characteristics of the emerging channel graph. In Chapter 7, we studied the properties of the Lighting Network's topology and analyzed its resilience towards random failures and targeted attacks. We found the network to be potentially susceptible to partitioning attacks by an adversary strategically targeting failure points in the topology. Moreover, we explored the potential benefits of multi-path route selection in Chapter 8. In Chapter 9, we studied the possibility of timing attacks on privacy in PCNs. We showed that an on-path attacker may exploit the timing behavior of multi-hop message exchanges in order to infer payment endpoints with high precision and recall. This once more emphasizes the importance of maintaining a decentralized channel topology. To this end, we studied different attachment strategies in Chapter 10. We empirically analyzed their benefits for end users and service providers, and showed that there is an inherent conflict of interest between the egoistical short-term view of node operators and the overall positive long-term impact of attachment strategies that champion decentralization.

In the third part, we explored how the foundational principles of blockchain networks can be applied to build more resilient distributed systems beyond the cryptocurrency use case. To this end, we showed in Chapter 11 how a decentralized system based on peer-to-peer network technology can be utilized to render the authenticity and provenance of public data verifiable.

The inherent trade-offs between the scalability, resilience, privacy, and decentralization blockchain networks have been re-emerging focus points throughout our research. While we have seen them unfold in different forms and magnitudes, the consequences of these trade-offs are often difficult to capture and determine. In this regard, the value of decentralization seems to be particularly hard to quantify, which might be the reason why this aspect is always in danger of being traded in for the sake of scalability and performance. However, as we have seen time and time again that the absence of sufficient decentralization can have severe consequences regarding resilience and privacy, we conclude that furthering the decentralization of distributed systems remains an ongoing challenge for the research community.

BIBLIOGRAPHY

PUBLICATIONS BY THE AUTHOR

[DRT19] Erik Daniel, Elias Rohrer, and Florian Tschorsch. “Map-Z: Exposing the Zcash Network in Times of Transition.” In: *LCN ’19: Proceedings of the 44th IEEE International Conference on Local Computer Networks*. Oct. 2019.

[LRT21] Kimberly Lange, Elias Rohrer, and Florian Tschorsch. “On the Impact of Attachment Strategies for Payment Channel Networks.” In: *ICBC ’21: Proceedings of the third International Conference on Blockchain and Cryptocurrency*. 2021.

[NRT19] Vinothkumar Nagasayanan, Elias Rohrer, and Florian Tschorsch. “ICRC: Instant Certificate Revocation Checking using Blockchain-backed Bloom Filters.” In: *Proceedings of the 30th Krypto-Tag*. 2019.

[RHT18] Elias Rohrer, Steffen Heidel, and Florian Tschorsch. “Webchain: Verifiable Citations and References for the World Wide Web.” In: *BLOCKCHAIN ’18: Proceedings of the IEEE International Conference on Blockchain*. July 2018.

[RHT20] Elias Rohrer, Steffen Heidel, and Florian Tschorsch. “Enabling Reference Verifiability for the World Wide Web with Webchain.” In: *ACM Trans. Internet Techn.* 20.4 (2020).

[RLT17] Elias Rohrer, Jann-Frederik Lass, and Florian Tschorsch. “Towards a Concurrent and Distributed Route Selection for Payment Channel Networks.” In: *CBT ’17: Proceedings of the 1st International Workshop on Cryptocurrencies and Blockchain Technology*. Sept. 2017.

[RMT19] Elias Rohrer, Julian Malliaris, and Florian Tschorsch. “Discharged Payment Channels: Quantifying the Lightning Network’s Resilience to Topology-Based Attacks.” In: *S&B ’19: Proceedings of IEEE Security & Privacy on the Blockchain*. June 2019.

[RT19] Elias Rohrer and Florian Tschorsch. “Kadcast: A Structured Approach to Broadcast in Blockchain Networks.” In: *AFT ’19: Proceedings of the first ACM conference on Advances in Financial Technologies*. Oct. 2019.

[RT20] Elias Rohrer and Florian Tschorsch. “Counting Down Thunder: Timing Attacks on Privacy in Payment Channel Networks.” In: *AFT ’20: Proceedings of the second ACM conference on Advances in Financial Technologies*. Oct. 2020.

[RT21a] Elias Rohrer and Florian Tschorsch. “Blockchain Layer Zero: Characterizing the Bitcoin Network through Measurements, Models, and Simulations.” In: *LCN ’21: Proceedings of the 46th IEEE International Conference on Local Computer Networks*. 2021.

[RT21b] Elias Rohrer and Florian Tschorsch. *Kadcast-NG: A Structured Broadcast Protocol for Blockchain Networks*. Submitted and currently under peer review. 2021.

OTHER PUBLICATIONS

[AM18] Maher Alharby and Aad van Moorsel. “BlockSim: A Simulation Framework for Blockchain Systems.” In: *SIGMETRICS Performance Evaluation Review* 46.3 (2018).

[ABP14] Jeff Alstott, Ed Bullmore, and Dietmar Plenz. “powerlaw: a Python package for analysis of heavy-tailed distributions.” In: *PloS one* 9.1 (2014).

[Aok+19] Yusuke Aoki, Kai Otsuki, Takeshi Kaneko, Ryohei Banno, and Kazuyuki Shudo. “SimBlock: A Blockchain Network Simulator.” In: *INFOCOM '19: Proceedings of the 2019 Conference on Computer Communications Workshops*. 2019.

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. “Hijacking Bitcoin: Routing Attacks on Cryptocurrencies.” In: *SP '17: Proceedings of the 38th IEEE Symposium on Security and Privacy*. 2017.

[Are+05] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. *DNS Security Introduction and Requirements*. RFC 4033 (Proposed Standard). RFC. RFC Editor, Mar. 2005.

[Aro+07] Dominik Aronsky, Sina Madani, Randy J Carnevale, Stephany Duda, and Michael T Feyder. “The prevalence and inaccessibility of Internet references in the biomedical literature at the time of publication.” In: *Journal of the American Medical Informatics Association* 14.2 (2007).

[ANW17] Mohamed Aturban, Michael L. Nelson, and Michele C. Weigle. “Difficulties of Timestamping Archived Web Pages.” In: *CoRR* abs/1712.03140 (2017). arXiv: `1712.03140`.

[Aum+20] Lukas Aumayr, Esra Ceylan, Matteo Maffei, Pedro Moreno-Sanchez, Iosif Salem, and Stefan Schmid. “Demand-Aware Payment Channel Networks.” In: *CoRR* abs/2011.14341 (2020). arXiv: `2011.14341`.

[Ava+20] Zeta Avarikioti, Lioba Heimbach, Yuyi Wang, and Roger Wattenhofer. “Ride the Lightning: The Game Theory of Payment Channels.” In: *FC '20: Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. 2020.

[AL94] Baruch Awerbuch and Tom Leighton. “Improved approximation algorithms for the multi-commodity flow problem and local competitive routing in dynamic networks.” In: *STOC '94: Proceedings of the 26th Annual ACM Symposium on Theory of Computing*. 1994.

[Bac02] Adam Back. *Hashcash-A Denial of Service Counter-Measure*. Aug. 2002. URL: `http://www.hashcash.org/papers/hashcash.pdf`.

[BNT20] Vivek Kumar Bagaria, Joachim Neu, and David Tse. "Boomerang: Redundancy Improves Latency and Throughput in Payment-Channel Networks." In: *FC '20: Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. 2020.

[Bag+12] Miriam Baglioni, Filippo Geraci, Marco Pellegrini, and Ernesto Lastres. "Fast Exact Computation of betweenness Centrality in Social Networks." In: *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2012, Istanbul, Turkey, 26-29 August 2012*. IEEE Computer Society, 2012.

[Bai+12] Michael Bailey, David Dittrich, Erin Kenneally, and Douglas Maughan. "The Menlo Report." In: *IEEE Secur. Priv.* 10.2 (2012).

[BS19] Ryohei Banno and Kazuyuki Shudo. "Simulating a Blockchain Network with SimBlock." In: *ICBC '19: Proceedings of the 1st International Conference on Blockchain and Cryptocurrency*. 2019.

[BA99] Albert-László Barabási and Réka Albert. "Emergence of scaling in random networks." In: *science* 286.5439 (1999).

[BM07] Ingmar Baumgart and Sebastian Mies. "S/Kademlia: A practicable approach towards secure key-based routing." In: *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems*. Dec. 2007.

[Ben+14a] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. "Zerocash: Decentralized anonymous payments from Bitcoin." In: *SP '14: Proceedings of the 35th IEEE Symposium on Security and Privacy*. May 2014.

[Ben+14b] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. "Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture." In: *USENIX Security '14: Proceedings of the 23rd USENIX Security Symposium*. 2014.

[Ben14] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System." In: *CoRR* abs/1407.3561 (2014).

[BSB19a] Ferenc Béres, István András Seres, and András A. Benczúr. "A Cryptoeconomic Traffic Analysis of Bitcoins Lightning Network." In: *CoRR* abs/1911.09432 (2019). arXiv: `1911.09432`.

[BSB19b] Ferenc Béres, István András Seres, and András A. Benczúr. "A Cryptoeconomic Traffic Analysis of Bitcoins Lightning Network." In: *CoRR* abs/1911.09432 (2019). arXiv: `1911.09432`. URL: `http://arxiv.org/abs/1911.09432`.

[Ber+18a] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'Angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. "Improving the Betweenness Centrality of a Node by Adding Links." In: *ACM Journal of Experimental Algorithmics* 23 (2018).

[Ber+18b] Elisabetta Bergamini, Pierluigi Crescenzi, Gianlorenzo D'Angelo, Henning Meyerhenke, Lorenzo Severini, and Yllka Velaj. "Improving the Betweenness Centrality of a Node by Adding Links." In: *ACM Journal of Experimental Algorithmics* 23 (2018).

[Ber06] Daniel J. Bernstein. "Curve25519: New Diffie-Hellman Speed Records." In: *PKC '06: Proceedings of the 9th International Conference on Theory and Practice of Public-Key Cryptography*. 2006.

[Ber08] Daniel J. Bernstein. "The Salsa20 Family of Stream Ciphers." In: *New Stream Cipher Designs - The eSTREAM Finalists*. 2008.

[Ber+11] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. "High-speed high-security signatures." In: *IACR Cryptology ePrint Archive* 2011 (2011).

[BYZ18] Wei Bi, Huawei Yang, and Maolin Zheng. "An Accelerated Method for Message Propagation in Blockchain Networks." In: *CoRR* abs/1809.00455 (2018). arXiv: `1809.00455`. URL: `http://arxiv.org/abs/1809.00455`.

[BKP14] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. "Deanonymisation of Clients in Bitcoin P2P Network." In: *CCS '14: Proceedings of the 21st ACM Conference on Computer and Communications Security*. Nov. 2014.

[Boc+06] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. "Complex networks: Structure and dynamics." In: *Physics reports* 424.4-5 (2006).

[Bor06] Nikita Borisov. "Computational Puzzles as Sybil Defenses." In: *P2P '06: Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*. 2006.

[Bra08] Ulrik Brandes. "On variants of shortest-path betweenness centrality and their generic computation." In: *Social Networks* 30.2 (2008).

[Bro10] Daniel R. L. Brown. *SEC 2: Recommended Elliptic Curve Domain Parameters*. Certicom Research, 2010. URL: `http://www.secg.org/sec2-v2.pdf`.

[BD05] Michael J Bugeja and Daniela Dimitrova. "Exploring the half-life of Internet footnotes." In: *Iowa Journal of Communication* 37.1 (2005).

[Cal+07] J. Callas, L. Donnerhacke, H. Finney, D. Shaw, and R. Thayer. *OpenPGP Message Format*. RFC 4880 (Proposed Standard). RFC. RFC Editor, Nov. 2007.

[Cao+20] Tong Cao, Jiangshan Yu, Jérémie Decouchant, Xiapu Luo, and Paulo Veríssimo. "Exploring the Monero Peer-to-Peer Network." In: *FC '20: Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. 2020.

[CL99] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance." In: *OSDI '99: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*. Feb. 1999.

[Cha85] David Chaum. "Security Without Identification: Transaction Systems to Make Big Brother Obsolete." In: *Commun. ACM* 28.10 (1985).

[CFN88] David Chaum, Amos Fiat, and Moni Naor. "Untraceable Electronic Cash." In: *CRYPTO '88: Proceedings of the 8th Conference on Advances in Cryptology*. Aug. 1988.

[Cha+19] Nakul Chawla, Hans Walter Behrens, Darren Tapp, Dragan Boscovic, and K Selçuk Candan. "Velocity: Scalability Improvements in Block Propagation Through Rateless Erasure Coding." In: *ICBC '19: Proceedings of the 1st International Conference on Blockchain and Cryptocurrency*. May 2019.

[CCF09] Thibault Cholez, Isabelle Chrisment, and Olivier Festor. "Evaluation of Sybil Attacks Protection Schemes in KAD." In: *AIMS '09: Proceedings of the 3rd International Conference on Autonomous Infrastructure, Management and Security*. 2009.

[CKY05] Marek Chrobak, Claire Kenyon, and Neal E. Young. "The Reverse Greedy Algorithm for the Metric *K*-Median Problem." In: *COCOON '05: Proceedings of the 11th Annual Conference on Computing and Combinatorics*. 2005.

[CSN09] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. "Power-law distributions in empirical data." In: *SIAM review* 51.4 (2009).

[Coo+08] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280 (Proposed Standard). RFC. RFC Editor, May 2008.

[Cro+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. "On Scaling Decentralized Blockchains - (A Position Paper)." In: *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*. Vol. 9604. Lecture Notes in Computer Science. Springer, 2016.

[CH13] Zoltán Czirkos and Gábor Hosszú. "Solution for the broadcasting in the Kademlia peer-to-peer overlay." In: *Computer Networks* 57.8 (2013).

[Dai98] Wei Dai. *B-Money*. 1998. URL: `http://www.weidai.com/bmoney.txt`.

[Dam+20] Gijs van Dam, Rabiah Abdul Kadir, Puteri N. E. Nohuddin, and Halimah Badioze Zaman. "Improvements of the Balance Discovery Attack on Lightning Network Payment Channels." In: *SEC '20: Proceedings of the 35th International Conference on Systems Security and Privacy Protection*. 2020.

[DG09] George Danezis and Ian Goldberg. “Sphinx: A Compact and Provably Secure Mix Format.” In: *SP '09: Proceedings of the 30th IEEE Symposium on Security and Privacy*. 2009.

[DRO18] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. “eltoo: A Simple Layer2 Protocol for Bitcoin.” In: *White paper: https://blockstream.com/eltoo.pdf* (2018).

[DW13] Christian Decker and Roger Wattenhofer. “Information propagation in the bitcoin network.” In: *P2P '13: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*. Sept. 2013.

[DW15] Christian Decker and Roger Wattenhofer. “A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels.” In: *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*. Vol. 9212. Lecture Notes in Computer Science. Springer, 2015.

[Del+03] Robert P Dellavalle, Eric J Hester, Lauren F Heilig, Amanda L Drake, Jeff W Kuntzman, Marla Graber, and Lisa M Schilling. “Going, Going, Gone: Lost Internet References.” In: *Science* 302.5646 (2003).

[Dij59] Edsger W. Dijkstra. “A note on two problems in connexion with graphs.” In: *Numerische Mathematik* 1 (1959).

[DPH14] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartı. “The Bitcoin P2P network.” In: *BITCOIN '14: Proceedings of the 1st Workshop on Bitcoin Research*. Mar. 2014.

[Dou02] John R. Douceur. “The Sybil Attack.” In: *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. 2002.

[Eck+20] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. “Splitting Payments Locally While Routing Interdimensionally.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020).

[ER59] Paul Erdős and Alfréd Rényi. “On random graphs I.” In: *Publ. Math. Debrecen* 6 (1959).

[ERE20a] Oguzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. “How to Profit from Payments Channels.” In: *FC '20: Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. 2020.

[ERE20b] Oguzhan Ersoy, Stefanie Roos, and Zekeriya Erkin. “How to Profit from Payments Channels.” In: *Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers*. Vol. 12059. Lecture Notes in Computer Science. Springer, 2020.

[Eya+16] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol.” In: *NSDI '16: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*. Mar. 2016.

[ES14] Ittay Eyal and Emin Gün Sirer. “Majority is not enough: Bitcoin mining is vulnerable.” In: *FC ’14: Proceedings of the 18th International Conference on Financial Cryptography and Data Security*. Mar. 2014.

[ET05] Gunther Eysenbach and Mathieu Trudel. “Going, Going, Still There: Using the WebCite Service to Permanently Archive Cited Web Pages.” In: *JMIR* 7.5 (Dec. 2005).

[Fac+09] Michael Factor, Ealan Henis, Dalit Naor, Simona Rabinovici-Cohen, Petra Reshef, Shahar Ronen, Giovanni Michetti, and Maria Guercio. “Authenticity and Provenance in Long Term Digital Preservation: Modeling and Implementation in Preservation Aware Storage.” In: *TaPP’09: In Proceedings of the 1st Workshop on the Theory and Practice of Provenance*. Feb. 2009.

[Fan+18] Giulia C. Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. “Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees.” In: *POMACS* 2.2 (2018).

[FV17] Giulia C. Fanti and Pramod Viswanath. “Deanonymization in the Bitcoin P2P Network.” In: *NIPS ’17: Proceedings of 30th Annual Conference on Neural Information Processing Systems*. Dec. 2017.

[Fei73] Horst Feistel. “Cryptography and Computer Privacy.” In: *Scientific American* 228.5 (1973).

[FSW14] Sebastian Feld, Mirco Schönfeld, and Martin Werner. “Analyzing the Deployment of Bitcoin’s P2P Network under an AS-level Perspective.” In: *ANT ’14: Proceedings of the 5th International Conference on Ambient Systems, Networks and Technologies*. June 2014.

[FR03] Faith E. Fich and Eric Ruppert. “Hundreds of impossibility results for distributed computing.” In: *Distributed Comput.* 16.2-3 (2003).

[Fin04] Hal Finney. *RPOW*. 2004. URL: `http://cryptome.org/rpow.htm`.

[FIP15] Federal Inf. Process. Stds. (NIST FIPS). “FIPS PUB 180-4. Secure Hash Standard.” In: (2015).

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. “Impossibility of distributed consensus with one faulty process.” In: *Journal of the ACM* (*JACM*) 32.2 (1985).

[FF56] Lester Randolph Ford and Delbert R Fulkerson. “Maximal flow through a network.” In: *Canadian journal of Mathematics* 8 (1956).

[Fre77a] Linton Freeman. “A set of measures of centrality based on betweenness.” In: *Sociometry* (1977).

[Fre77b] Linton C Freeman. “A set of measures of centrality based on betweenness.” In: *Sociometry* (1977).

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. "A Decentralized Public Key Infrastructure with Identity Retention." In: *IACR Cryptology ePrint Archive* 2014 (2014).

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "The Bitcoin Backbone Protocol: Analysis and Applications." In: *EUROCRYPT '15: Proceedings of the 34th International Conference on the Theory and Applications of Cryptographic Techniques*. Apr. 2015.

[GKL20] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. "Full Analysis of Nakamoto Consensus in Bounded-Delay Networks." In: *IACR Cryptol. ePrint Arch.* 2020 (2020). URL: `https://eprint.iacr.org/2020/277`.

[GGM14] Christina Garman, Matthew Green, and Ian Miers. "Decentralized Anonymous Credentials." In: *NDSS '14: Proceedings of the Network and Distributed System Security Symposium*. Feb. 2014.

[Gen+18] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. "Decentralization in Bitcoin and Ethereum Networks." In: *FC '18: Proceedings of the 22nd International Conference on Financial Cryptography and Data Security*. Feb. 2018.

[Ger+14] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. "Is Bitcoin a Decentralized Currency?" In: *SP '14: Proceedings of the 35th IEEE Symposium on Security and Privacy*. 2014.

[Ger+16] Arthur Gervais, Ghassan Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. "On the Security and Performance of Proof of Work Blockchains." In: *CCS '16: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*. Oct. 2016.

[Gib04] Stevyn Gibson. "Open source intelligence: An intelligence lifeline." In: *The RUSI Journal* 149.1 (2004).

[GT88a] Andrew V Goldberg and Robert E Tarjan. "A new approach to the maximum-flow problem." In: *Journal of the ACM (JACM)* 35.4 (1988).

[GT88b] Andrew V. Goldberg and Robert Endre Tarjan. "A new approach to the maximum-flow problem." In: *J. ACM* 35.4 (1988).

[Gon85] Teofilo F. Gonzalez. "Clustering to Minimize the Maximum Intercluster Distance." In: *Theoretical Computer Science* 38 (1985).

[Gou67] Peter R Gould. "On the geographical interpretation of eigenvalues." In: *Transactions of the Institute of British Geographers* (1967).

[GM17] Matthew Green and Ian Miers. "Bolt: Anonymous Payment Channels for Decentralized Currencies." In: *CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[GT94] Gregory Grefenstette and Pasi Tapanainen. "What is a word, what is a sentence?: problems of Tokenisation." In: (1994).

[GNH18] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. “Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin.” In: *BITCOIN '18: Proceedings of the 5th Workshop on Bitcoin Research*. Feb. 2018.

[Gud+20] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. “SoK: Layer-Two Blockchain Protocols.” In: *FC '20: Proceedings of the 24th International Conference on Financial Cryptography and Data Security*. 2020.

[HS91] Stuart Haber and W. Scott Stornetta. “How to Time-Stamp a Digital Document.” In: *CRYPTO '90: Proceedings of the 10th Conference on Advances in Cryptology*. Ed. by Alfred Menezes and Scott A. Vanstone. 1991.

[HSS08] Aric Hagberg, Pieter Swart, and Daniel S Chult. *Exploring network structure, dynamics, and function using NetworkX*. Tech. rep. Los Alamos National Lab (LANL). Los Alamos, NM, USA, 2008.

[Han+20] Yilin Han, Chenxing Li, Peilun Li, Ming Wu, Dong Zhou, and Fan Long. “Shrec: bandwidth-efficient transaction relay in high-throughput blockchain systems.” In: *SoCC '20: Proceedings of the 2020 ACM Symposium on Cloud Computing*. Oct. 2020.

[HZ20] Jona Harris and Aviv Zohar. “Flood & Loot: A Systemic Attack on The Lightning Network.” In: *AFT '20: Proceedings of the second ACM conference on Advances in Financial Technologies*. 2020.

[HSW09] Ragib Hasan, Radu Sion, and Marianne Winslett. “Preventing history forgery with secure provenance.” In: *TOS* 5.4 (2009).

[Hei+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. “TumbleBit: An Untrusted Bitcoin-Compatible Anonymous Payment Hub.” In: *NDSS '17: Proceedings of the 24th Annual Network and Distributed System Security Symposium*. 2017.

[Hei+15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. “Eclipse Attacks on Bitcoin’s Peer-to-Peer Network.” In: *USENIX Security '15: Proceedings of the 24th USENIX Security Symposium*. Aug. 2015.

[Hen+08] Thomas R Henderson, Mathieu Lacage, George F Riley, Craig Dowell, and Joseph Kopena. “Network simulations with the ns-3 simulator.” In: *SIGCOMM Demonstration* 14.14 (2008).

[Hen+19] Sebastian A. Henningsen, Daniel Teunis, Martin Florian, and Björn Scheuermann. “Eclipsing Ethereum Peers with False Friends.” In: *S&B '19: Proceedings of IEEE Security & Privacy on the Blockchain*. 2019.

[Her+19] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal Pedrosa, Cristina Pérez-Solà, and Joaquín García-Alfaro. “On the Difficulty of Hiding the Balance of Lightning Network Channels.” In: *AsiaCCS '19: Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 2019.

[HS85] Dorit S. Hochbaum and David B. Shmoys. "A Best Possible Heuristic for the *k*-Center Problem." In: *Mathematics of Operations Research* 10.2 (1985).

[HS12] P. Hoffman and J. Schlyter. *The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA*. RFC 6698 (Proposed Standard). RFC. RFC Editor, Aug. 2012.

[HG08] Mark D Humphries and Kevin Gurney. "Network small-worldness: a quantitative method for determining canonical network equivalence." In: *PloS one* 3.4 (2008).

[HM14] Trung Dong Huynh and Luc Moreau. "ProvStore: A Public Provenance Repository." In: *IPAW '14: Proceedings of the 5th International Provenance and Annotation Workshop*. June 2014.

[Imt+19] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis. "Churn in the Bitcoin Network: Characterization and Impact." In: *ICBC '19: Proceedings of the 1st International Conference on Blockchain and Cryptocurrency*. 2019.

[Jac+12] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nick Briggs, and Rebecca Braynard. "Networking named content." In: *Commun. ACM* 55.1 (2012).

[JH12] Rob Jansen and Nicholas Hopper. "Shadow: Running Tor in a Box for Accurate and Efficient Experimentation." In: *NDSS '12: Proceedings of the Network and Distributed System Security Symposium*. 2012.

[Joh+13] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. "Users Get Routed: Traffic Correlation on Tor by Realistic Adversaries." In: *CCS '13: Proceedings of the 20th ACM Conference on Computer and Communications Security*. Oct. 2013.

[KJL18] Ben Kaiser, Mireya Jurado, and Alex Ledger. "The Looming Threat of China: An Analysis of Chinese Influence on Bitcoin." In: *CoRR* abs/1810.02466 (2018).

[Kap+18] George Kappos, Haaroon Yousaf, Mary Maller, and Sarah Meiklejohn. "An Empirical Analysis of Anonymity in Zcash." In: *arXiv preprint arXiv:1805.03180* (2018).

[Kap+20] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. "An Empirical Analysis of Privacy in the Lightning Network." In: *arXiv preprint arXiv:2003.12470* (2020).

[KAC12] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. "Double-spending Fast Payments in Bitcoin." In: *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security*. Oct. 2012.

[Kar12] David Karpf. "Social science research methods in Internet time." In: *Information, communication & society* 15.5 (2012).

[KEB98] Dogan Kesdogan, Jan Egner, and Roland Büschkes. “Stop-and-Go-MIXes Providing Probabilistic Anonymity in an Open System.” In: *IH ’98: Proceedings of the Second International Workshop on Information Hiding*. 1998.

[KNW20] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. *Timelocked Bribing*. Cryptology ePrint Archive, Report 2020/774. `https://eprint.iacr.org/2020/774`. 2020.

[KG17] Rami Khalil and Arthur Gervais. “Revive: Rebalancing Off-Blockchain Payment Networks.” In: *CCS ’17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[KR21] Julia Khamis and Ori Rottenstreich. “Demand-aware Channel Topologies for Off-chain Payments.” In: *13th International Conference on COMmunication Systems & NETworkS, COMSNETS 2021, Bangalore, India, January 5-9, 2021*. Jan. 2021.

[KRS18] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. “A Better Method to Analyze Blockchain Consistency.” In: *CCS ’18: Proceedings of the 25nd ACM SIGSAC Conference on Computer and Communications Security*. Oct. 2018.

[Kim+18] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. “Measuring Ethereum Network Peers.” In: *IMC ’18: Proceedings of the Internet Measurement Conference*. Oct. 2018.

[Kla+] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. *bloXroute: A Scalable Trustless Blockchain Distribution Network WHITEPAPER*.

[KLR09] Michael Kohnen, Mike Leske, and Erwin P. Rathgeb. “Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network.” In: *NETWORKING ’09: Proceedings of the 8th International IFIP-TC 6 Networking Conference*. 2009.

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. “An Analysis of Anonymity in Bitcoin Using P2P Network Traffic.” In: *FC ’14: Proceedings of the 18th International Conference on Financial Cryptography and Data Security*. Mar. 2014.

[KD15] Tobias Kuhn and Michel Dumontier. “Making Digital Artifacts on the Web Verifiable and Reliable.” In: *IEEE Trans. Knowl. Data Eng.* 27.9 (2015).

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine generals problem.” In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982).

[Lan+17] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan R. Iyengar, et al. “The QUIC Transport Protocol: Design and Internet-Scale Deployment.” In: *SIGCOMM ’17: Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication*. 2017.

[LHM10] Bob Lantz, Brandon Heller, and Nick McKeown. "A network in a laptop: rapid prototyping for software-defined networks." In: *HotNets '10: Proceedings of the 9th ACM Workshop on Hot Topics in Networks*. Oct. 2010.

[LMZ20] Peng Li, Toshiaki Miyazaki, and Wanlei Zhou. "Secure Balance Planning of Off-blockchain Payment Channel Networks." In: *INFOCOM '20: Proceedings of the 39th Conference on Computer Communications*. 2020.

[Lia+17] Xueping Liang, Sachin Shetty, Deepak K. Tosh, Charles A. Kamhoua, Kevin A. Kwiat, and Laurent Njilla. "ProvChain: A Blockchain-based Data Provenance Architecture in Cloud Environment with Enhanced Privacy and Availability." In: *CCGRID '17: Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. May 2017.

[Lin+20] Jian-Hong Lin, Kevin Primicerio, Tiziano Squartini, Christian Decker, and Claudio J. Tessone. "Lightning Network: a second path towards centralisation of the Bitcoin economy." In: *CoRR* abs/2002.02819 (2020). arXiv: `2002.02819`.

[Loc+10] Thomas Locher, David Mysicka, Stefan Schmid, and Roger Wattenhofer. "Poisoning the Kad Network." In: *ICDCN '10: Proceedings of the 11th International Conference on Distributed Computing and Networking*. 2010.

[Lub+11] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. *RaptorQ Forward Error Correction Scheme for Object Delivery*. RFC 6330 (Proposed Standard). RFC. RFC Editor, Aug. 2011.

[Mal+17a] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. "SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks." In: *NDSS '17: Proceedings of the 24th Annual Network and Distributed System Security Symposium*. Feb. 2017.

[Mal+17b] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. "Concurrency and Privacy with Payment-Channel Networks." In: *CCS '17: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017.

[Mal+19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability." In: *NDSS '19: Proceedings of the 26th Annual Network and Distributed System Security Symposium*. 2019.

[Man+05] Petros Maniatis, Mema Roussopoulos, Thomas J. Giuli, David S. H. Rosenthal, and Mary Baker. "The LOCKSS peer-to-peer digital preservation system." In: *ACM Trans. Comput. Syst.* 23.1 (2005).

[Mao+20] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. "Perigee: Efficient Peer-to-Peer Network Design for Blockchains." In: *PODC '20: Proceedings of the 2020 ACM Symposium on Principles of Distributed Computing*. Aug. 2020.

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. "Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network." In: *IACR Cryptology ePrint Archive* 2018 (2018).

[MF20] Stefano Martinazzi and Andrea Flori. "The evolving topology of the Lightning Network: Centralization, efficiency, robustness, synchronization, and anonymity." In: *PloS one* 15.1 (2020).

[Mas51] Frank J Massey Jr. "The Kolmogorov-Smirnov test for goodness of fit." In: *Journal of the American statistical Association* 46.253 (1951).

[MAQ99] H. Massias, X. Serret Avila, and J.-J. Quisquater. "Design Of A Secure Timestamping Service With Minimal Trust Requirement." In: *SITB'99: Proceedings of the 20th Symposium on Information Theory in the Benelux*. May 1999.

[MM02] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric." In: *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*. 2002.

[Maz+20] Subhra Mazumdar, Sushmita Ruj, Ram Govind Singh, and Arindam Pal. "HushRelay: A Privacy-Preserving, Efficient, and Scalable Routing Algorithm for Off-Chain Payments." In: *ICBC '20: Proceedings of the second International Conference on Blockchain and Cryptocurrency*. 2020.

[McK+08] Nick McKeown, Thomas E. Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker, and Jonathan S. Turner. "OpenFlow: Enabling Innovation in Campus Networks." In: *Computer Communication Review* 38.2 (2008).

[Mei+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. "A fistful of bitcoins: characterizing payments among men with no names." In: *IMC '13: Proceedings of the 13th ACM SIGCOMM Conference on Internet Measurement*. Oct. 2013.

[Mel+15] Marcela S. Melara, Aaron Blankstein, Joseph Bonneau, Edward W. Felten, and Michael J. Freedman. "CONIKS: Bringing Key Transparency to End Users." In: *USENIX Security '15: Proceedings of the 24th USENIX Security Symposium*. Aug. 2015.

[Mer87] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function." In: *CRYPTO '87: Proceedings of the 7th Conference on Advances in Cryptology*. Aug. 1987.

[MT09] Adam Meyerson and Brian Tagiku. "Minimizing Average Shortest Path Distances via Shortcut Edge Addition." In: *APPROX '09: Proceedings of the 12th International Workshop on Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. 2009.

[Mil+19] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. "Sprites and State Channels: Payment Networks that Go Faster Than Lightning." In: *FC '19: Proceedings of the 23rd International Conference on Financial Cryptography and Data Security*. 2019.

[MJ15] Andrew Miller and Rob Jansen. "Shadow-Bitcoin: Scalable Simulation via Direct Execution of Multi-Threaded Applications." In: *CSET '15: Proceedings of the 8th Workshop on Cyber Security Experimentation and Test*. Aug. 2015.

[Mil+15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. *Discovering bitcoin's public topology and influential nodes*. May 2015.

[MZ20] Ayelet Mizrahi and Aviv Zohar. "Congestion Attacks in Payment Channel Networks." In: *CoRR* abs/2002.06564 (2020). arXiv: `2002.06564`.

[Mor17] Luc Moreau. "A Canonical Form for PROV Documents and Its Application to Equality, Signature, and Validation." In: *ACM Trans. Internet Techn.* 17.4 (2017).

[NBS20] Ryunosuke Nagayama, Ryohei Banno, and Kazuyuki Shudo. "Identifying Impacts of Protocol and Internet Development on the Bitcoin Network." In: *ISCC '20: Proceedings of the 2020 Symposium on Computers and Communications*. 2020.

[Nak08] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[Nau+19] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. "Erlay: Efficient Transaction Relay for Bitcoin." In: *CCS '19: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. Nov. 2019.

[Neu19] Till Neudecker. *Characterization of the Bitcoin Peer-to-Peer Network (2015-2018)*. Tech. rep. 1. Karlsruhe, 2019. 29 pp.

[NAH16] Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein. "Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network." In: *UIC '16: Proceedings of the 2016 International Conference on Ubiquitous Intelligence & Computing*. July 2016.

[NH19] Till Neudecker and Hannes Hartenstein. "Network Layer Aspects of Permissionless Blockchains." In: *IEEE Communications Surveys and Tutorials* 21.1 (2019).

[NL15] Y. Nir and A. Langley. *ChaCha20 and Poly1305 for IETF Protocols*. RFC 7539 (Informational). RFC. RFC Editor, May 2015.

[Nis+20] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. “Toward Active and Passive Confidentiality Attacks On Cryptocurrency Off-Chain Networks.” In: *ICISSP '20: Proceedings of the 6th International Conference on Information Systems Security and Privacy*. 2020.

[Ozi+19] A. Pinar Ozisik, Gavin Andresen, Brian Neil Levine, Darren Tapp, George Bissias, and Sunny Katkuri. “Graphene: efficient interactive set reconciliation applied to blockchain propagation.” In: *SIGCOMM '19: Proceedings of the 2019 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2019.

[PH97] David D. Palmer and Marti A. Hearst. “Adaptive Multilingual Sentence Boundary Disambiguation.” In: *Computational Linguistics* 23.2 (1997).

[Pap+18] Giuseppe Pappalardo, Tiziana Di Matteo, Guido Caldarelli, and Tomaso Aste. “Blockchain inefficiency in the Bitcoin peers network.” In: *EPJ Data Science* 7.1 (2018).

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. “Analysis of the Blockchain Protocol in Asynchronous Networks.” In: *EUROCRYPT 17: Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2017.

[Pax+11] V. Paxson, M. Allman, J. Chu, and M. Sargent. *Computing TCP's Retransmission Timer*. RFC 6298 (Proposed Standard). RFC. RFC Editor, June 2011.

[PK00] Andreas Pfitzmann and Marit Köhntopp. “Anonymity, Unobservability, and Pseudonymity - A Proposal for Terminology.” In: *PET '00: Proceedings of the International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*. 2000.

[PN18] Dmytro Piatkivskyi and Mariusz Nowostawski. “Split Payments in Payment Networks.” In: *CBT '18: Proceedings of the 2nd International Workshop on Cryptocurrencies and Blockchain Technology*. 2018.

[PN20] Rene Pickhardt and Mariusz Nowostawski. “Imbalance measure and proactive channel rebalancing algorithm for the Lightning Network.” In: *ICBC '20: Proceedings of the second International Conference on Blockchain and Cryptocurrency*. 2020.

[Pio+17] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. “The Loopix Anonymity System.” In: *USENIX Security '17: Proceedings of the 26th USENIX Security Symposium*. 2017.

[PD16] Joseph Poon and Thaddeus Dryja. “The bitcoin lightning network: Scalable off-chain instant payments.” In: (Jan. 2016).

[Pri+16] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. “Flare: An Approach to Routing in Lightning Network.” In: (2016).

[Que17] Jeffrey Quesnelle. “On the linkability of Zcash transactions.” In: *arXiv preprint arXiv:1712.01210* (2017).

[RH13] Fergal Reid and Martin Harrigan. “An analysis of anonymity in the bitcoin system.” In: *Security and Privacy in Social Networks*. Springer, 2013.

[RM12] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347 (Proposed Standard). RFC. RFC Editor, Jan. 2012.

[RR97] Jeffrey C. Reynar and Adwait Ratnaparkhi. “A Maximum Entropy Approach to Identifying Sentence Boundaries.” In: *ANLP '97: Proceedings of the 5th Applied Natural Language Processing Conference*. Mar. 1997.

[Rin+20] Daniel Rincon, Eva Yiwei Wu, Sofia Dewar, and Daniel Zhu. “Identifying Beneficial Connection Types in Payment Channel Networks: The Case of Lightning.” In: *University of California Berkeley* (2020).

[Rob19] Daniel Robinson. “HTLCs Considered Harmful.” In: *Stanford Blockchain Conference*. Jan. 2019.

[Rog15] Richard Rogers. “Digital methods for web research.” In: *Emerging trends in the social and behavioral sciences: An interdisciplinary, searchable, and linkable resource* (2015).

[RS13] Dorit Ron and Adi Shamir. “Quantitative Analysis of the Full Bitcoin Transaction Graph.” In: *FC '13: Proceedings of the 17th International Conference on Financial Cryptography and Data Security*. Apr. 2013.

[Roo+18] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. “Settling Payments Fast and Private: Efficient Decentralized Routing for Path-Based Transactions.” In: *NDSS '18: Proceedings of the 25th Symposium on Network and Distributed System Security*. Feb. 2018.

[Ros11] Meni Rosenfeld. *Analysis of Bitcoin Pooled Mining Reward Systems*. Computing Research Repository, Report abs/1112.4980. 2011.

[Ros12] Meni Rosenfeld. *Analysis of hashrate-based double spending*. 2012. URL: `https://bitcoil.co.il/Doublespend.pdf`.

[SZ20] Yotam Sali and Aviv Zohar. “Optimizing Off-Chain Payment Networks in Cryptocurrencies.” In: *CoRR* abs/2007.09410 (2020). arXiv: `2007.09410`. URL: `https://arxiv.org/abs/2007.09410`.

[SOA16] Muntadher Fadhil Sallal, Gareth Owenson, and Mo Adda. “A Bitcoin Model for Evaluation of Clustering to Improve Propagation Delay in Bitcoin Network.” In: *CSE '16: Proceedings of the 2016 IEEE International Conference on Computational Science and Engineering*. Aug. 2016.

[SOA17a] Muntadher Fadhil Sallal, Gareth Owenson, and Mo Adda. “Locality based approach to improve propagation delay on the Bitcoin peer-to-peer network.” In: *IM ’17: Proceedings of the IFIP/IEEE International Symposium on Integrated Network and Service Management*. May 2017.

[SOA17b] Muntadher Fadhil Sallal, Gareth Owenson, and Mo Adda. “Proximity Awareness Approach to Enhance Propagation Delay on the Bitcoin Peer-to-Peer Network.” In: *ICDCS ’17: Proceedings of the 37th International Conference on Distributed Computing Systems*. June 2017.

[SSD19] João Santos, Nuno Santos, and David Dias. “DClaims: A Censorship Resistant Web Annotations System using IPFS and Ethereum.” In: *CoRR* abs/1912.03388 (2019). arXiv: `1912.03388`.

[Ser+19] István András Seres, László Gulyás, Dániel A. Nagy, and Péter Burcsi. “Topological Analysis of Bitcoin’s Lightning Network.” In: *MARBLE ’19: Proceedings of the first International Conference on Mathematical Research for Blockchain Economy*. 2019.

[Sil+20] Paulo Silva, David Vavricka, João Barreto, and Miguel Matos. “Impact of Geo-Distribution and Mining Pools on Blockchains: A Study of Ethereum.” In: *DSN ’20: Proceedings of the 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 2020.

[Siv+18] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Mohammad Alizadeh, Giulia C. Fanti, and Pramod Viswanath. “Routing Cryptocurrency with the Spider Network.” In: *HotNets ’18: Proceedings of the 17th ACM Workshop on Hot Topics in Networks*. 2018.

[SZ15] Yonatan Sompolinsky and Aviv Zohar. “Secure High-Rate Transaction Processing in Bitcoin.” In: *FC ’15: Proceedings of the 19th International Conference on Financial Cryptography and Data Security*. Jan. 2015.

[Spe17] Giorgio Alfredo Spedicato. “Discrete Time Markov Chains with R.” In: *The R Journal* 9.2 (2017).

[SEB07] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. “Exploiting KAD: possible uses and misuses.” In: *Computer Communication Review* 37.5 (2007).

[Sto+01] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. “Chord: A scalable peer-to-peer lookup service for internet applications.” In: *SIGCOMM ’01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. 2001.

[SR06] Daniel Stutzbach and Reza Rejaie. “Understanding churn in peer-to-peer networks.” In: *IMC ’06: Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*. Oct. 2006.

[Sza05] Nick Szabo. *Bit Gold*. 2005. URL: `https://nakamotoinstitute.org/bit-gold/`.

[Tan+20] Weizhao Tang, Weina Wang, Giulia C. Fanti, and Sewoong Oh. "Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks." In: *Proc. ACM Meas. Anal. Comput. Syst.* 4.2 (2020).

[TMM20] Sergei Tikhomirov, Pedro Moreno-Sanchez, and Matteo Maffei. "A Quantitative Analysis of Security, Anonymity and Scalability for the Lightning Network." In: *S&B '20: Proceedings of IEEE Security & Privacy on the Blockchain*. 2020.

[Tik+20] Sergei Tikhomirov, Rene Pickhardt, Alex Biryukov, and Mariusz Nowostawski. "Probing Channel Balances in the Lightning Network." In: *CoRR* abs/2004.00333 (2020). arXiv: 2004.00333. URL: https://arxiv.org/abs/2004.00333.

[TSZ19] Saar Tochner, Stefan Schmid, and Aviv Zohar. "Hijacking Routes in Payment Channel Networks: A Predictability Tradeoff." In: *CoRR* abs/1909.06890 (2019). arXiv: 1909.06890. URL: http://arxiv.org/abs/1909.06890.

[TBP20] Florian Tramèr, Dan Boneh, and Kenny Paterson. "Remote Side-Channel Attacks on Anonymous Transactions." In: *USENIX Security '20: Proceedings of the 29th USENIX Security Symposium*. 2020.

[Tro+17] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. "Systematizing Decentralization and Privacy: Lessons from 15 Years of Research and Deployments." In: *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017.4 (2017).

[TYE20] Itay Tsabary, Matan Yechieli, and Ittay Eyal. "MAD-HTLC: Because HTLC is Crazy-Cheap to Attack." In: *CoRR* abs/2006.12031 (2020). arXiv: 2006.12031. URL: https://arxiv.org/abs/2006.12031.

[UPS11] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. "A survey of DHT security techniques." In: *ACM Comput. Surv.* 43.2 (2011).

[VPC12] Bregtje Van der Haak, Michael Parks, and Manuel Castells. "The future of journalism: Networked journalism." In: *International journal of communication* 6 (2012).

[VTM14] Marie Vasek, Micah Thornton, and Tyler Moore. "Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem." In: *BITCOIN '14: Proceedings of the 1st Workshop on Bitcoin Research*. Mar. 2014.

[VFV17] Shaileshh Bojja Venkatakrishnan, Giulia C. Fanti, and Pramod Viswanath. "Dandelion: Redesigning the Bitcoin Network for Anonymity." In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)* (2017).

[VCS03] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. "Karma: A secure economic framework for peer-to-peer resource sharing." In: *P2PEcon '03: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems*. June 2003.

[WCY19] Canhui Wang, Xiaowen Chu, and Qin Yang. "Measurement and Analysis of the Bitcoin Networks: A View from Mining Pools." In: *CoRR* abs/1902.07549 (2019). arXiv: `1902.07549`.

[WL15] Luqin Wang and Yong Liu. "Exploring Miner Evolution in Bitcoin Network." In: *PAM '15: Proceedings of the 16th International Conference on Passive and Active Measurement*. Mar. 2015.

[Wan+08] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. "Attacking the Kad network." In: *4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008, Istanbul, Turkey, September 22-25, 2008*. 2008.

[Wan+13] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. "Attacking the kad network - real world evaluation and high fidelity simulation using DVN." In: *Security and Communication Networks* 6.12 (2013).

[Wan+19] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. "Flash: efficient dynamic routing for offchain networks." In: *CoNEXT '19: Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. Dec. 2019.

[WS98] Duncan J Watts and Steven H Strogatz. "Collective dynamics of 'small-world'networks." In: *nature* 393.6684 (1998).

[WH20] Finnegan Waugh and Ralph Holz. "An empirical study of availability and reliability properties of the Bitcoin Lightning Network." In: *CoRR* abs/2006.14358 (2020). arXiv: `2006.14358`. URL: `https://arxiv.org/abs/2006.14358`.

[Wou16] P. Wouters. *DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP*. RFC 7929 (Experimental). RFC. RFC Editor, Aug. 2016.

[YG03] Beverly Yang and Hector Garcia-Molina. "Designing a Super-Peer Network." In: *ICDE '03: Proceedings of the 19th International Conference on Data Engineering*. 2003.

[Yu+18] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. "CoinExpress: A Fast Payment Routing Mechanism in Blockchain-Based Payment Channel Networks." In: *ICCCN '18: Proceedings of the 27th International Conference on Computer Communication and Networks*. July 2018.

[ZWL21] Lihao Zhang, Taotao Wang, and Soung Chang Liew. "Speeding up Block Propagation in Blockchain Network: Uncoded and Coded Designs." In: *CoRR* abs/2101.00378 (2021). URL: `https://arxiv.org/abs/2101.00378`.

WEBPAGES

[@FIB21] Fast Internet Bitcoin Relay Engine (FIBRE). *Homepage*. 2021. URL: http://bitcoinfibre.org (Retrieved Apr. 17, 2021).

[@ns-21] ns-3 Network Simulator. *Homepage*. 2021. URL: https://www.nsnam.org (Retrieved Apr. 17, 2021).

[@Bit19] Bitcoin Core. *Homepage*. 2019. URL: https://bitcoincore.org (Retrieved Apr. 19, 2019).

[@Bit21a] Bitcoin Project. *Bitcoin RPC API*. 2021. URL: https://developer.bitcoin.org/reference/rpc/index.html (Retrieved July 1, 2021).

[@Bit21b] Bitnodes. *Homepage*. 2021. URL: https://bitnodes.io (Retrieved Apr. 22, 2021).

[@Blo19] Blockchain.com. *Hashrate Distribution*. 2019. URL: https://blockchain.com/pools?timespan=4days (Retrieved Dec. 10, 2019).

[@Blo21a] Blockchain.com. *Average Confirmation Time of Bitcoin Transactions*. 2021. URL: https://www.blockchain.com/charts/avg-confirmation-time (Retrieved Apr. 13, 2021).

[@Blo21b] Blockchain.com. *Average fees per Bitcoin transaction*. 2021. URL: https://www.blockchain.com/charts/fees-usd-per-transaction (Retrieved Apr. 13, 2021).

[@Blo21c] Blockchain.com. *Bitcoin. Average Transactions Per Block*. 2021. URL: https://www.blockchain.com/charts/n-transactions-per-block (Retrieved July 1, 2021).

[@BTC18] BTC.com. *Pool Distribution*. 2018. URL: https://btc.com/stats/pool?pool_mode=week (Retrieved Sept. 23, 2019).

[@But14] Vitalik Buterin. *A next-generation smart contract and decentralized application platform*. 2014. URL: https://ethereum.org/en/whitepaper/.

[@c-l20] c-lightning Project. *Github*. 2020. URL: https://github.com/ElementsProject/lightning (Retrieved May 8, 2020).

[@Cor16] Matt Corallo. *BIP 152: Compact Block Relay*. Apr. 2016. URL: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[@Daf15] Suhas Daftuar. *BIP 130: sendheaders message*. May 2015. URL: https://github.com/bitcoin/bips/blob/master/bip-0130.mediawiki.

[@Dec20] Christian Decker. *Rendez-Vous Routing Proposal*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc/blob/rendez-vous/proposals/0001-rendez-vous.md (Retrieved Apr. 20, 2021).

[@Dev20a] Lightning Network Developers. *BOLT #4: Onion Routing Protocol*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/04-onion-routing.md (Retrieved Apr. 20, 2021).

[@Dev20b] Lightning Network Developers. *BOLT #7: P2P Node and Channel Discovery*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc/blob/master/07-routing-gossip.md (Retrieved Apr. 20, 2021).

[@Dev20c] Lightning Network Developers. *BOLT In-Progress Specifications*. 2020. URL: https://github.com/lightningnetwork/lightning-rfc (Retrieved Apr. 20, 2021).

[@Dir21] Jean-Christophe Begue Dirkjan Ochtman Benjamin Saunders. *Quinn. Pure-rust QUIC protocol implementation*. 2021. URL: https://github.com/quinn-rs/quinn (Retrieved July 1, 2021).

[@Ele19] Electric Coin Company. *Zcash Network Information*. 2019. URL: https://z.cash/upgrade/ (Retrieved May 2019).

[@Ele21] Electric Coin Company. *Zcash Homepage*. 2021. URL: https://z.cash/ (Retrieved Apr. 19, 2021).

[@Eth19a] Ethereum Project. *devp2p Network Protocols*. 2019. URL: https://github.com/ethereum/devp2p (Retrieved Dec. 2, 2019).

[@Eth21] Etherscan. *Ethereum Network Utilization*. 2021. URL: https://etherscan.io/chart/networkutilization (Retrieved Apr. 6, 2021).

[@Eth19b] Etherscan.io. *Ethereum Block Size History*. 2019. URL: https://etherscan.io/chart/blocksize (Retrieved May 24, 2019).

[@Fou] Interledger Foundation. *ILP-RFC 0018: Interleger Risk Mitigations - Payment Griefing*. URL: https://interledger.org/rfcs/0018-connector-risk-mitigations/ (Retrieved Feb. 12, 2019).

[@Gro20] The Tcpdump Group. *TCPDUMP/LIBPCAP public repository*. 2020. URL: https://www.tcpdump.org/ (Retrieved Sept. 29, 2020).

[@iPl19] iPlane. *An Information Plane for Distributed Services*. 2019. URL: http://web.eecs.umich.edu/~harshavm/iplane/ (Retrieved May 21, 2019).

[@Jer19] Andrew Lumsdaine Jeremy Siek Lie-Quan Lee. *Boost Graph Library*. 2019. URL: https://www.boost.org/doc/libs/1_66_0/libs/graph/doc/ (Retrieved Apr. 2019).

[@LD18] Lightning Labs and The Lightning Network Developers. *LND Shadow Route Github Issue*. 2018. URL: https://github.com/lightningnetwork/lnd/issues/1222 (Retrieved Apr. 20, 2021).

[@Lan19] The Go Programming Language. *Website*. 2019. URL: https://golang.org (Retrieved Sept. 23, 2019).

[@LND20a] LND. *Github Commit: Move Second Chance Logic*. 2020. URL: https://github.com/lightningnetwork/lnd/commit/dc13da5abbfa429273b516abd566f6c6fa5bb200 (Retrieved May 8, 2020).

[@LND20b] LND. *Github: Policy Failure Logic*. 2020. URL: https://github.com/lightningnetwork/lnd/blob/1354a461701b9396f0b4a35b01d308c5fcc0dbd2/routing/result_interpretation.go#L343 (Retrieved May 8, 2020).

[@Mac19] Internet Archive: Wayback Machine. *Website*. 2019. URL: https://archive.org/web/ (Retrieved Sept. 23, 2019).

[@Max21] MaxMind, Inc. *GeoIP GeoLite2 database*. 2021. URL: https://dev.maxmind.com/geoip/geoip2/geolite2/ (Retrieved Apr. 22, 2021).

[@Med19] US National Library of Medicine - National Institutes of Health. *PubMed Website*. 2019. URL: https://www.ncbi.nlm.nih.gov/pubmed/ (Retrieved Sept. 23, 2019).

[@Nam19] Namecoin. *Website*. 2019. URL: https://namecoin.info (Retrieved Sept. 23, 2019).

[@Ope19] OpenTimestamps. *Website*. 2019. URL: https://opentimestamps.org (Retrieved Sept. 23, 2019).

[@Per19] Perma.cc. *Website*. 2019. URL: https://perma.cc (Retrieved Sept. 23, 2019).

[@Per18] Trevor Perrin. *The Noise Protocol Framework*. 2018. URL: https://noiseprotocol.org/noise.html (Retrieved Apr. 20, 2021).

[@Pic] René Pickhardt. *Is the Barabási-Albert model a reasonable choice for the autopilot?* URL: https://github.com/lightningnetwork/lnd/issues/677 (Retrieved June 14, 2020).

[@Pro] XRP Ledger Project. *Ripple Data API v2*. URL: https://developers.ripple.com/data-api.html (Retrieved Feb. 13, 2019).

[@Rai20] Raiden Network. *Homepage*. 2020. URL: https://raiden.network/ (Retrieved Apr. 20, 2021).

[@Roh] Elias Rohrer. *Snapshots of the Lightning Network*. URL: https://git.tu-berlin.de/rohrer/discharged-pc-data/ (Retrieved July 12, 2021).

[@Rus21] Rust Bitcoin Developers. *Rust Bitcoin Library*. 2021. URL: https://github.com/rust-bitcoin/rust-bitcoin (Retrieved July 1, 2021).

[@Sta19] LibP2P. A Modular Networking Stack. *Website*. 2019. URL: https://libp2p.io (Retrieved Sept. 23, 2019).

[@Tea21] Rust Team. *Rust. A language empowering everyone to build reliable and efficient software*. 2021. URL: https://www.rust-lang.org/ (Retrieved July 1, 2021).

[@Tec20] Karlsruhe Institute of Technology DSN. *Bitcoin Monitoring*. 2020. URL: `https://dsn.tm.kit.edu/bitcoin/` (Retrieved Nov. 30, 2020).

[@The21] The ZeroMQ Authors. *ZeroMQ. An open-source universal messaging library.* 2021. URL: `https://zeromq.org/` (Retrieved July 1, 2021).

[@Tru] Trustnodes.com. *Lightning Network DDoS Sends 20% of Nodes Down*. URL: `https://www.trustnodes.com/2018/03/21/lightning-network-ddos-sends-20-nodes` (Retrieved Feb. 6, 2019).

[@Val20] Henry de Valence. *A New Network Stack For Zcash*. 2020. URL: `https://www.zfnd.org/blog/a-new-network-stack-for-zcash/` (Retrieved Apr. 17, 2021).

[@Vue20] Vue.js. *Website*. 2020. URL: `https://vuejs.org/` (Retrieved Feb. 1, 2020).

[@W3C20a] W3C. *PROV-Overview*. 2020. URL: `https://www.w3.org/TR/prov-overview/` (Retrieved Jan. 23, 2020).

[@W3C20b] W3C. *Web Annotation Working Group*. 2020. URL: `https://www.w3.org/annotation/` (Retrieved Jan. 23, 2020).

[@Wor19] WordPress. *Website*. 2019. URL: `https://wordpress.org` (Retrieved Sept. 23, 2019).