Security and Anonymity Aspects of the Network Layer of Permissionless Blockchains

zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Till Neudecker

aus Kassel

Tag der mündlichen Prüfung:	30. November 2018
Erster Gutachter:	Prof. Dr. rer.nat. Hannes Hartenstein Karlsruher Institut für Technologie
Zweiter Gutachter:	Prof. Dr. Jordi Herrera-Joancomartí Universitat Autònoma de Barcelona

This work was supported by the German Federal Ministry of Education and Research within the projects *KASTEL_IoE* and *KASTEL_ISE* in the Competence Center for Applied Security Technology (*KASTEL*) and by the state of Baden-Württemberg through bwHPC, bwFileStorage, and LSDF Online Storage.

Zusammenfassung

Permissionless Blockchains sind dezentrale Systeme, die Konsens erzielen. Das prominenteste Beispiel einer Permissionless Blockchain ist das elektronische Zahlungssystem Bitcoin, welches Konsens über die von Teilnehmern des Systems erzeugten Finanztransaktionen erzielt. Während verteilter Konsens seit Jahrzehnten Gegenstand zahlreicher Forschungsarbeiten ist, ist Bitcoin das erste bekannte System, welches Konsens im sog. *permissionless*-Modell erzielt, d.h. ohne die vorausgehende Feststellung der Identitäten der Teilnehmer des Systems.

Die Teilnehmer von Permissionless Blockchains kommunizieren über ein unstrukturiertes Peer-to-Peer (P2P) Netzwerk miteinander. Da das Verfahren zur Konsensbildung von Permissionless Blockchains auf Daten basiert, die über dieses P2P-Netzwerk übertragen werden, können Sicherheitslücken in der Netzwerkschicht auch die Konsensbildung und damit die angestrebte Funktion des Systems beeinflussen. Während unstrukturierte P2P-Netzwerke in der Vergangenheit umfassend analysiert wurden, führt ihr Einsatz in Permissionless Blockchains zu Sicherheitsanforderungen und Angreifermodellen, die bisher noch nicht berücksichtigt wurden. Obwohl einzelne Angriffe auf die Netzwerkschicht von Permissionless Blockchains analysiert wurden, ist unklar, welche Sicherheitseigenschaften die Netzwerkschicht von Permissionless Blockchains haben sollte. Diese Unklarheit motiviert die erste in dieser Dissertation behandelte Forschungsfrage: *Wie können Anforderungen und Zielkonflikte, die in den Mechanismen der Netzwerkschicht von Permissionless Blockchains vorhanden sind, untersucht werden?*

In dieser Dissertation wird eine Systematisierung von Angriffen auf die Netzwerkschicht von Bitcoin vorgestellt, in der Angriffe hinsichtlich der angegriffenen Mechanismen und der Auswirkungen der Angriffe auf höhere Schichten des Systems kategorisiert werden. Basierend auf der Systematisierung werden fünf Anforderungen für die Netzwerkschicht von Permissionless Blockchains abgeleitet: Leistung, niedrige Beteiligungskosten, Anonymität, Robustheit gegen Denial-of-Service Angriffe sowie Topologieverschleierung. Darüber hinaus werden der Entwurfsraum der Netzwerkschicht aufgezeigt und der Einfluss von Entwurfsentscheidungen auf die Erfüllung von Anforderungen qualitativ untersucht. Die durchgeführten Systematisierungen weisen auf inhärente Zielkonflikte sowie Forschungsmöglichkeiten hin und unterstützen die Entwicklung von Permissionless Blockchains.

Weiterhin wird auf Grundlage von seit 2015 durchgeführten Messungen eine Charakterisierung des Bitcoin-P2P-Netzwerks präsentiert. Die Charakterisierung ermöglicht die Parametrisierung und Validierung von Simulationsmodellen und die Bewertung der Zuverlässigkeit von realen Experimenten. Darüber hinaus gewährt die Netz-

werkcharakterisierung Einblicke in das Verhalten von Netzwerkknoten und deren Betreibern. Beispielsweise kann gezeigt werden, dass Sybil-Ereignisse in der Vergangenheit im Bitcoin-P2P-Netzwerk stattgefunden haben und dass die Leistung und die Anonymitätseigenschaften der Transaktions- und Blockausbreitung durch Implementierungs- und Protokolländerungen verbessert worden sind.

Auf Grundlage dieser Charakterisierung werden zwei *ereignisdiskrete Simulationsmodelle* des Bitcoin-P2P-Netzwerks entworfen. Die Modelle werden durch einen Vergleich der simulierten Informationsausbreitungsverzögerung mit der beobachteten Informationsausbreitungsverzögerung im realen Netzwerk validiert. Da der Vergleich eine hohe Übereinstimmung zeigt, ermöglichen die vorgestellten Simulationsmodelle die Simulation des Bitcoin-Netzwerks mit einer Genauigkeit, die für die Analyse von Angriffen im Bitcoin-Netzwerk ausreicht.

Die vorgestellten Simulationsmodelle sowie die durchgeführte Systematisierung von Angriffen verdeutlichen die Bedeutung der Kenntnis der Netzwerktopologie als Grundlage für Forschung und die Analyse von Deanonymisierungsangriffe. Daher addressiert die zweite Forschungsfrage dieser Dissertation Methoden der Topologieinferenz und der Deanonymisierung: *Unter welchen Voraussetzungen und in welchem Maße sind netzwerkbasierte Topologieinferenz und Deanonymisierung in Bitcoin (un)möglich?* Diese Frage wird durch Anwendung der vorgeschlagenen Methodenkombination aus Messungen, Simulationen und Experimenten beantwortet.

In dieser Dissertation werden vier verschiedene Methoden zur Topologieinferenz vorgestellt und unter Verwendung von Experimenten und Simulationsstudien analysiert. Anhand von Experimenten wird gezeigt, dass ein Angreifer, der in der Lage ist, Verbindungen zu allen Knoten des Netzwerks zu etablieren, die direkten Nachbarn eines Netzwerkknotens mit hoher Sensitivität (*recall*) und Genauigkeit (*precision*) (87 % recall, 71 % precision) durch die Veröffentlichung von *widersprüchlichen* Transaktionen im Netzwerk herausfinden kann. Unter der Annahme eines passiven Angreifers, der in der Lage ist, sich mit allen erreichbaren Netzwerkknoten zu verbinden, war 2016 ein Rückschluss auf die Nachbarn eines Netzwerkknotens mit einer Sensitivität von 40 % bei einer Genauigkeit von 40 % durch Beobachtung von mindestens acht Transaktionen, die von diesem Netzwerkknoten stammen, möglich. Darüber hinaus ist es möglich, die Akkumulation mehrere Transaktionen zum Zwecke der Topologieinferenz zu geringen Kosten auszunutzen. Allerdings bleibt die erwartete Inferenzqualität aufgrund fehlender Validierungsmöglichkeiten unklar. Schließlich kann simulativ gezeigt werden, dass der Peer-Discovery-Mechanismus eines P2P-Netzwerks bei bestimmte Parametrisierungen Topologinferenz ermöglichen kann.

Abschließend wird die Möglichkeit einer netzwerkbasierten Deanonymisierung bewertet, indem analysiert wird, ob eine Korrelation zwischen der IP-Adresse des Netzwerkknotens, der eine Transaktion veröffentlicht, und dem mutmaßlichen Ersteller der Transaktion besteht. Der zugrundeliegende Datensatz basiert auf den durchgeführten Messungen und besteht aus fast 10 Millionen Transaktionen mit zugehörigen IP-Adressen. Es wird gezeigt, dass Transaktionen von 5 % bis 8.3 % der Benutzer auffallend häufig von einzelnen Netzwerkknoten veröffentlicht wurden, was diese Benutzer dem Risiko netzwerkbasierter Deanonymisierungsangriffe aussetzt.

Abstract

Permissionless blockchains are decentralized consensus systems. The most prominent example of a permissionless blockchain is the electronic payment system Bitcoin that achieves consensus on financial transactions issued by participants. While distributed consensus has been a subject of research for decades, Bitcoin was the first system to achieve consensus in the *permissionless* model, i.e., without prior establishment of identities of participants.

The participants of a permissionless blockchain communicate over an unstructured peer-to-peer (P2P) network. As the consensus protocol of permissionless blockchains relies on data transmitted through that P2P network, vulnerabilities in the network layer can also affect the establishment of consensus and, therefore, the intended function of the system. While unstructured P2P networks have been extensively analyzed in the past, their deployment in permissionless blockchains leads to different security requirements and adversary models that existing work has not considered yet. Furthermore, although a number of attacks on the network layer of permissionless blockchains have been analyzed, no general notion of the desired security properties of the network layer of permissionless blockchains exists. This motivates the first research question addressed in this dissertation: *How to research requirements and tradeoffs present in the network layer mechanisms of permissionless blockchains?*

First, we provide a *systematization of attacks* on the network layer of Bitcoin regarding the exploited network layer mechanisms and the effects of the attacks on the application and consensus layers of Bitcoin. Based on this systematization, we derive five requirements for the network layer of permissionless blockchains: performance, low cost of participation, anonymity, denial-of-service resistance, and topology hiding. Furthermore, we *systematize the design space* of the network layer and qualitatively show the effect of design decisions on the fulfillment of requirements. Our systematizations indicate inherent tradeoffs, point out research possibilities, and guide developers of permissionless blockchains.

Secondly, based on measurements performed since 2015 we provide a *characterization of the Bitcoin P2P network*. The characterization enables the parametrization and validation of simulation models and the assessment of the reliability of real-world experiments. Furthermore, the network characterization provides insights into the behavior of peers and their operators. For instance, we provide evidence that Sybil events happened in the past in the Bitcoin P2P network. Additionally, our measurements show that the performance and anonymity of transaction and block propagation has been improved by implementation and protocol changes.

Thirdly, research of the network layer of permissionless blockchains can be performed by employing network simulations. Based on the network characterization, we present two *discrete-event simulation models* of the Bitcoin P2P network that can be used for performing simulations at the full scale of the Bitcoin network. We validate our models by comparison of the simulated information propagation delay to the observed one in the real-world network. As the comparison shows a high correspondence, the presented simulation models enable the simulation of the Bitcoin network with a precision sufficient for the analysis of attacks.

Our simulation models and our systematization of attacks highlight the importance of knowledge of the network topology as a basis for research and as a prerequisite for certain attacks, such as network-based deanonymization attacks. Therefore, we address topology inference and deanonymization in the second research question of this dissertation: *Under which assumptions and to which degree are network-based topology inference and deanonymization (im-)possible in Bitcoin?* We answer this question by building on the results of the first research question, i.e., by applying a combination of measurements and experiments, and by using the presented simulation models.

We evaluate four different topology inference methods using real-world experiments and simulations. Using real-world experiments we show that an adversary that is able to connect to all peers of the network can infer the direct neighbors of a peer with high recall and precision (87 % recall, 71 % precision) at low costs by *actively* publishing *conflicting* transactions on the network. Furthermore, for a *passive* adversary that is able to connect to all reachable peers, inference of the neighbors of a peer was possible in 2016 with a recall of 40 % at a precision of 40 % by observing at least 8 transactions originating from that peer. Additionally, network topology information can be inferred at low costs by exploiting the client behavior *transaction accumulation*, however, the expected real-world inference quality remains unclear, because a ground-truth validation is inherently hard to perform for the proposed method. Subsequently, we show in simulations that a P2P network's peer discovery mechanism can be exploited for topology inference for certain parametrizations of the peer discovery mechanism.

Finally, we assess the possibility of network-based deanonymization by analyzing whether a correlation between the IP address of the peer announcing a transaction first and the suspected creator of the transaction can be detected. Our dataset is based on our network characterization and consists of almost 10 million transactions with associated IP addresses. We show that transactions of 5 % to 8.3 % of users were conspicuously often published by individual peers, potentially making these users susceptible to network-based deanonymization attacks.

Contents

List of Figures

List of Tables

1 Introduction

> What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party.
>
> (Satoshi Nakamoto, 2008)

With an increasing volume and complexity of trade, banks have become core institutions of the economic system centuries ago. In addition to traditional banks, the widespread use of online shopping gave rise to digital payment systems (e.g., PayPal) during the last decade. Banks and digital payment systems serve as intermediaries between merchants and buyers, acting as a trusted third party between those two entities. Motivated by the required trust, a person or group using the pseudonym *Satoshi Nakamoto* published the concept of *Bitcoin* [Nako8], which enables financial interaction through the Internet without a single trusted entity.

Bitcoin realizes a decentralized ledger that is used to store transactions created by users in order to transfer money. As long as all participants agree on the stored transactions, a public ledger can be sufficient to realize a currency system.[1] Since its inception, more than 300 million Bitcoin transactions were issued by users of the system.

Bitcoin is the first known implementation of what later became known as a *permissionless blockchain*.[2] Technically, permissionless blockchains are decentralized systems that achieve consensus on data (e.g., transactions) that is stored in *blocks*. These blocks are chained using cryptographic hash functions in order to guarantee

[1] The *Yap stone money* is often used as an analogy [Cor75]: A society on Micronesian islands used to use large stone disks as a form of currency. Because these stones were hard to transport, ownership of a stone was not indicated by physical access, but by the recorded history of past changes in ownership. This requires that everyone agrees on the history of past transactions.

[2] The Bitcoin paper contains neither the term *blockchain*, nor the term *permissionless*.

integrity of the stored data. While distributed consensus has been a subject of research for decades, Bitcoin was the first system to (eventually) achieve consensus in the *permissionless* model [Wat16].

In contrast to *permissioned* consensus, no prior establishment of identities of participants is required for permissionless consensus. This allows new participants to join the system without any form of authentication. In order to reduce the effectiveness of Sybil attacks [Dou02], which are inherently possible in the permissionless model, Bitcoin utilizes a *proof-of-work* scheme that limits the ability to contribute to the consensus process to the computational resources of the participant.

Participants of permissionless blockchains communicate over an unstructured peer-to-peer (P2P) network in order to achieve a high degree of decentralization. As the consensus process relies on data transmitted by the network layer, vulnerabilities in the network layer can also affect the establishment of consensus and, therefore, the realized function of the system (e.g., the Bitcoin currency system). Hence, the security of the overall system also depends on the security of the network layer.

Unstructured P2P networks have been widely used in the past decades, especially for the purpose of *file sharing*. Therefore, unstructured P2P networks also have been the subject of extensive research. However, the use of unstructured P2P networks as communication infrastructure for permissionless blockchains leads to vastly different (security) requirements and adversary models that were not considered, yet. Therefore, insights delivered by existing research are often not directly applicable to the domain of permissionless blockchains.

Specifically, we observe the following limitations of existing research on the network layer of permissionless blockchains: While a number of attacks on the network layer of permissionless blockchains has been analyzed, no general notion of the desired security properties of the network layer of blockchains exists. Furthermore, the relationship between design options, the degree of fulfillment of security properties, and the effect of inherent tradeoffs among design options is not sufficiently understood. Finally, there is a lack of data and models that are required to analyze security and anonymity aspects of the network layer of permissionless blockchains. Motivated by these limitations, the first research question addressed in this dissertation is:

> How to research requirements and tradeoffs present in the network layer mechanisms of permissionless blockchains?

Two important requirements of the network layer of permissionless blockchains are resistance against denial-of-service (DoS) attacks and resistance against deanonymization attacks. In permissionless blockchains, a lack of resistance against DoS attacks not only reduces the availability of the system, but can also enable attacks on the consensus system, which can result in direct monetary loss of participants of the system [HKZG15]. Ensuring the anonymity of users is especially important considering the fact that Bitcoin publicly stores financial transactions. While users do not include their real-world identity in Bitcoin transactions, they use pseudonyms (Bitcoin *addresses*). The use of pseudonyms makes it crucial to prevent a linking between the pseudonyms of a user and other personally identifiable information of that user,

in order to ensure anonymity. Such linking could be facilitated by the observation of the message propagation process of the underlying P2P network [FV17].

Knowledge of the topology of the P2P network is a key factor for the feasibility, costs, and success of these attacks. While specific approaches for topology inference of the Bitcoin P2P network exist [MLP+15], it is not clear to which degree topology inference is generally possible in P2P networks of permissionless blockchains, and what the resulting inference quality is. Besides for these adversarial purposes, knowledge of the P2P network topology is also highly beneficial for research, as it can enable precise simulations of the system. Therefore, the second research question addressed in this dissertation is:

> Under which assumptions and to which degree are network-based topology inference and deanonymization (im-)possible in Bitcoin?

1.1 Contributions

The contributions of this dissertation address the two main stated research questions, and can, therefore, be categorized into contributions that mainly enable research of the network layer of permissionless blockchains, and contributions that address the possibility and quality of network-based topology inference and deanonymization.

Advancement of Research of the Network Layer of Permissionless Blockchains

In this dissertation we (1) systematize threats and design options of the network layer, (2) collect and provide measurement data of the real-world Bitcoin P2P network, and (3) develop, parameterize, and validate simulation models for Bitcoin.

Systematization We systematize known attacks on the network layer of permissionless blockchains regarding the exploited mechanisms and effects on higher layers. The systematization shows that all known attacks aiming at a monetary benefit of the adversary are based on DoS attacks. Furthermore, the aim of all studied attacks is either monetary benefit or the deanonymization of users by linking IP addresses to application data (i.e., pseudonyms of users). The systematization also allows us to derive five requirements: performance, low cost of participation, anonymity, DoS resistance, and topology hiding. Furthermore, we survey the design space of the network layer by analyzing design decisions of a wide range of proposed and deployed systems. For each design decision we qualitatively show the effect on the fulfillment of the requirements and identify existing tradeoffs. Showing the existence of such tradeoffs indicates directions of future research, and enables designers of permissionless blockchains to develop the network layer according to their requirements.

Measurements We perform long-term (2015 to 2018, ongoing) measurements of the Bitcoin P2P network by joining the network with two *monitor nodes* and collecting raw data on the propagation of messages through the network. The Bitcoin P2P network

consists of more than 10,000 reachable peers and is used to carry out real-world financial transactions valued to up to several billion Euros per day.[3]

The measurements provide insights into the technical and human activity on the network. First, the ongoing characterization of the network allows to monitor the effects of changes to client implementations. For instance, improvements to the performance and anonymity of message propagation in commonly used client implementations manifest in the observed propagation characteristics. Furthermore, as the Bitcoin P2P network is an open network, the composition of peers can change drastically within short periods of time. For instance, during our measurements we could observe multiple Sybil events with several thousand peers joining the network in a coordinated way. Finally, although the Bitcoin P2P is a technical system, the observations also allow conclusions regarding the behavior of the persons operating the peers. For instance, our measurements show that it usually takes several months between the release of a new client version and the widespread deployment of that client version. This indicates that peers are operated by a large number of individuals.

In addition to the described insights regarding the network itself, the measurements enable simulations of the Bitcoin P2P network and experiments performed in the Bitcoin P2P network. Our ongoing characterization of the network enables the assessment of the reliability of experiments performed in the real-world network. For instance, if experiments are performed during periods with observed unusual user or network behavior, the reliability of experiments is substantially reduced. Furthermore, in order to perform simulations that represent the real-world system, the simulated network should be a precise model of the real-world network. Based on our measurements we parametrize and validate our simulation models. Aggregated measurement data has been made available to the research community.[4]

Simulation Models We present two approaches for modeling the behavior of peers of the Bitcoin P2P network as a discrete-event model. Both models differ in their degree of abstraction and in the computational effort required for the execution of simulations. Based on the conducted measurements we parametrize our simulation model, and perform an empirical validation by comparing simulation results to real-world measurements. The validation shows a close correspondence between simulated and measured information propagation. The parametrization based on our real-world measurements enables the simulation of the Bitcoin network with a precision sufficient for the analysis of attacks.

Possibility and Quality of Network-Based Topology Inference and Deanonymization

In the dissertation we propose several methods for network based topology inference and deanonymization and analyze their feasibility using simulations and real-world experiments.

[3] https://blockchain.info/de/charts/estimated-transaction-volume-usd

[4] https://dsn.tm.kit.edu/bitcoin

Network-Based Topology Inference We present and analyze four different methods for inferring the P2P topology of permissionless blockchains. The first method targets a behavior (*transaction accumulation*) specific to the most commonly used Bitcoin client. While the general feasibility of the approach is shown in simulations and real-world experiments, the method lacks the possibility to influence which existing connection of the network is inferred. As this makes a ground-truth validation infeasible, the real-world inference quality of the proposed method remains unclear.

The second method exploits a behavior of clients regarding conflicting transactions: As conflicting transactions can be created by participants of the network at minimal costs (i.e., the cost is constant in the number of created transactions), such *double spends* are dropped by clients to prevent DoS attacks on the network. We show in simulations and real-world experiments that this behavior can be exploited for inferring the neighbors of a peer at low costs reaching a recall of 87 % at a precision of 71 %. This inference quality is potentially sufficient for academic and adversarial purposes, assuming the adversary is capable of connecting to all peers of the network.

The third method is based solely on passively observing the timing of message propagation on the Bitcoin P2P network, which is inherently possible in permissionless blockchains. Experiments in the real-world Bitcoin network show that topology inference was possible with a recall of 40 % at a precision of 40 % at the time of performing the experiments in 2016. However, improvements to the transaction propagation mechanism of commonly used Bitcoin clients render this method hardly feasible in 2018. Finally, the fourth method targets Bitcoin's *peer discovery* mechanism, i.e., the exchange of IP addresses of reachable peers between clients. Our simulation results show that only certain parametrizations of the peer discovery mechanism are exploitable for topology inference.

Network-Based Deanonymization We assess whether the use of network observations facilitates the deanonymization of participants by linking Bitcoin addresses contained in published transactions to the IP addresses of peers that were observed to relay the transaction first. While Bitcoin addresses are pseudonyms, prior research shows that the set of addresses used by an individual participant can be clustered using several proposed heuristics based on information available from the public blockchain. We apply all clustering heuristics that are known to us to blockchain information and associate the resulting clusters with IP address information extracted from our network observations. Our results indicate that for the vast majority of users network information cannot facilitate deanonymization in the considered adversary model. The adversary model is defined by the performed measurement of the Bitcoin P2P network (i.e., the adversary runs a small number of passive monitor nodes connected to all reachable peers of the network, but does not have access to any further information such as the network topology). Still, a small number of participants (5 % to 8.3 %) exhibit conspicuous behavior that might make them susceptible to network based deanonymization attacks in the considered adversary model.

Parts of the contributions presented in this dissertation have been published in the following previous works:

- T. Neudecker, P. Andelfinger, and H. Hartenstein. A simulation model for analysis of attacks on the Bitcoin peer-to-peer network. In *IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015*, pages 1327–1332, May 2015.
- T. Neudecker, P. Andelfinger, and H. Hartenstein. Timing analysis for inferring the topology of the Bitcoin peer-to-peer network. In *Intl. IEEE Conference on Advanced and Trusted Computing (ATC), 2016*, pages 358–367, July 2016.
- T. Neudecker and H. Hartenstein. Could network information facilitate address clustering in Bitcoin? In *4th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 2017*, 2017.
- T. Neudecker. Bitcoin Cash (BCH) Sybil nodes on the Bitcoin peer-to-peer network. In *Karlsruhe Reports in Informatics 4*, 2018.
- M. Grundmann, T. Neudecker, and H. Hartenstein. Exploiting transaction accumulation and double spends for topology inference in Bitcoin. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 2018*, 2018.
- T. Neudecker and H. Hartenstein. Network layer aspects of permissionless blockchains. Accepted in *IEEE Communications Surveys & Tutorials*, 2018.

1.2 Thesis Outline

The remainder of this thesis is structured as follows. In Chapter 2 we revisit existing definitions, models, and results from the past decades in order to give a broader view on the theoretical foundation of permissionless blockchains. Furthermore, we introduce the technical concept of Bitcoin as the first instance of a permissionless blockchain, and discuss which existing models and results are applicable to permissionless blockchains, and which are not.

In Chapter 3 we define the requirements of the network layer of permissionless blockchains. Furthermore, we provide a systematization of known attacks on the network layer of permissionless blockchains, and survey its design space.

In Chapter 4 we give an extensive characterization of the real-world Bitcoin P2P network. For this, we first describe the used methodology, then the network is characterized regarding its general, long-term properties. Finally, an analysis of a selection of unusual events and a discussion is presented.

In Chapter 5 we introduce the simulation models that will be used thoughout this dissertation. We present two approaches for modeling the behavior of peers of the Bitcoin network as a discrete-event model. Furthermore, based on the measurements presented in Chapter 4 we parametrize our simulation model, and perform an empirical validation by comparing simulation results to real-world measurements.

In Chapter 6 we present and analyze four different topology inference methods targeting the P2P network of permissionless blockchains. While the first method

exploits the accumulation of transactions in a specific Bitcoin client implementation, the second analyzed method targets the handling of double spending transactions, which is equal among all clients. The third proposed and analyzed method is based solely on passively observing the timing of message propagation. The fourth method targets the peer discovery mechanism of permissionless blockchains, which is required for the establishment of connections in a P2P network.

In Chapter 7 we empirically address the question whether observations on the Bitcoin P2P network can be used to link the creators of transactions to IP addresses of Bitcoin peers. Finally, a conclusion of the results presented in this dissertation and an outlook is given in Chapter 8.

2
Fundamentals

The public hype associated with blockchains can easily give the impression that blockchains are a fundamentally new technology, regularly associated with the promise to improve the life of individuals and the expectation to change societies as a whole. However, although the concept of permissionless blockchains was first published in 2008 [Nak08], neither the problem that permissionless blockchains aim to solve, nor the building blocks combined into permissionless blockchains are new to computer science [NC17].

Permissionless blockchains solve a specific type of the *consensus problem*, a problem that has been the subject of research in the field of distributed systems for decades [Asp]. Recent works show that existing models require only minor adaptions to be able to be used in the analysis of permissionless blockchains (e.g., [GKL15]). Furthermore, the individual technologies combined into permissionless blockchains have been known for many years before the concept of Bitcoin was published. For instance, the proof-of-work mechanism was published in 1992 [DN92], the elliptic curve used for public key cryptography has been standardized in 2000 [Res10], and peer-to-peer networks like Napster were extensively used around the year 2000.

In this chapter we first revisit existing definitions, models, and results from the past decades in order to give a broader view on the theoretical foundation of permissionless blockchains. Then, we introduce the technical concept of Bitcoin as the first instance of a permissionless blockchain. Finally, we discuss which existing models and results are applicable to permissionless blockchains, and show aspects of permissionless blockchains that actually push existing models out of their limits.

2.1 Definitions and Key Literature Results

In this section we revisit definitions and results that are highly relevant to permissionless blockchains. We acknowledge that there are many more definitions, models, and results that are in some sense relevant to permissionless blockchains, which are not included in this section.

Distributed Systems Distributed systems have been a subject of research for several decades. One of the most commonly referenced definitions for distributed systems is given by Andrew Tanenbaum [TVS07]:

> A distributed system is a collection of independent computers that appears to its users as a single coherent system.

Several remarks can be made regarding this definition: A collection implies that a distributed system consists of at least two independent computers. Typically, a distributed system consists of a large number of independent computers.[1] *Independent* means that each computer[2] has a local state, which is affected by local computations. *Independent* does not imply that computers have to be controlled by separate entities, i.e., all computers of a distributed system can be under the control of one single entity. In order to appear as a coherent system, the independent computers need to coordinate, which is usually achieved by passing messages between computers, or by having memory shared among computers. A common example for distributed systems are multi-core processor systems, where each processor can be seen as an independent computer that interacts with the other processors via shared memory. Another example are cloud storage systems (e.g., Amazon S3), which store data transparently to the user on a large number of computers.

Decentralized Systems Decentralized systems are a subset of distributed systems, which impose additional constraints on the control over the components of the distributed system [TIDH17]:

> Decentralized system: A distributed system in which multiple authorities control different components and no single authority is fully trusted by all others.

The definition implies that systems, in which multiple authorities control different components, but there is a single fully trusted authority, are not regarded as decentralized. An example for such a not decentralized system could be an online game, which is run at the computers of multiple authorities, but which relies on a central server coordinating the game. On the other hand, the *Domain Name System* (DNS)

[1] Aspnes defines distributed systems by their structure: „A typical distributed system will consist of some large number of interacting devices that each run their own programs but that are affected by receiving messages, or observing shared-memory updates or the states of other devices.“ [Asp]

[2] Other commonly used terms for *computer* are *component*, *node*, or *peer*, which will be used interchangeable throughout this work.

can be regarded as a decentralized system, because no single fully trusted entity exists in the system. One could argue that the DNS root servers are components of the system that are *more trusted* than other components, however, this is not excluded by the given definition.

Peer-to-Peer System In order to achieve decentralization, communication between components must not be controlled by a single entity. This suggests *peer-to-peer* system architectures as used in, e.g., BitTorrent [PGES05] and Bitcoin. Steinmetz and Wehrle [SW05] define a peer-to-peer system as a

> self-organizing system of equal, autonomous entities (peers) which aims for the shared usage of distributed resources in a networked environment avoiding central services.

Clearly, according to their definitions, peer-to-peer systems are a subset of decentralized systems. The definition of peer-to-peer systems even goes a little bit further by requiring *equal* and *autonomous* peers. For instance, the Domain Name System is not a peer-to-peer system, because there are DNS root servers, which perform a distinct role in the system, different from the role of other DNS servers.

Consensus As stated above, distributed systems require some form of coordination in order to provide their service to the user. One example of such coordination is reaching *consensus*, i.e., the components of a distributed system have to agree on some common value. Permissionless blockchains agree on a set of data, therefore, they can be seen as systems solving an instance of the consensus problem.

We will now briefly introduce the terms and scenario considered in the consensus problem. The system is composed of two types of nodes: correct nodes, which perform the protocol as specified, and faulty nodes, which may fail according to a specified failure model. In order to solve the consensus problem, a system has to satisfy three properties [Wat16]:

- **Termination:** All correct nodes terminate in finite time.
- **Agreement:** All correct nodes agree on the same value.
- **Validity:** Every node starts with some input value. The decision value must be the input value of a correct node.

There are various common assumptions that can be made for different aspects of the system model. For instance:

- **Communication Model:** There are several possibilities to model the communication between nodes. First, communication can be assumed to be synchronous. There are several synchronous communication models, some are very strict, i.e., assume that all messages between nodes are transmitted within a single timestep, others only give an upper bound on the delay of messages. Synchronous communication models allow protocols to use some kind of timeout mechanism

to detect faulty nodes. Contrary, in the asynchronous communication model messages can be delayed by an arbitrary duration (but are eventually delivered). Therefore, timeouts cannot be used to reliably detect the failure of nodes in asynchronous communication models.

- **Failure Model:** Nodes and communication links can both fail according to some failure model. The most common process failure models are *crash failures*, which let faulty processes stop working, and *Byzantine failures*, which allow faulty processes to arbitrarily divert from the specified protocol. Communication failure models include *omission failures*, which drop messages, and *Byzantine failures*, which can also create and alter messages [Asp].
- **(In-)determinism:** The consensus protocol can be either deterministic or probabilistic.

Although this list is not exhaustive, it sketches important aspects in the analysis of consensus systems. We will discuss whether these assumptions can be made in the analysis of Bitcoin in Section 2.3.

Key Literature Results We will now briefly describe key results from the literature on consensus protocols. These results are often mentioned in discussions on properties of blockchains. A discussion of the applicability of these results on permissionless blockchains will be presented in Section 2.3.

The **Two Generals Problem** states that no protocol can guarantee consensus between two parties if omission failures (i.e., messages may be dropped on the link) are allowed [Gra78, AEH75].[3] However, there are probabilistic protocols, which can reduce the probability of violating the agreement property asymptotically close to zero by increasing the number of rounds [Asp].

The **Byzantine Generals Problem** [LSP82] uses the following system model: A synchronous communication model is assumed, i.e., messages are reliably delivered after a known delay. This assumption circumvents the two generals problem described previously. However, Byzantine process failures are allowed, i.e. faulty processes can deviate from the protocol in any way. It was shown that the number of correct nodes must outnumber the number of faulty nodes by strictly more than a factor of two (i.e., the number of faulty nodes must be strictly less than one third of the total number of nodes). There are several protocols solving the Byzantine Generals Problem, for instance Practical Byzantine Fault Tolerance (PBFT) [CL+99].

Another key result (commonly referred to as the **FLP result** [FLP85]) states that no deterministic protocol can guarantee consensus in the asynchronous communication model in the presence of at least one crash-failure node. Please note that this result only applies to deterministic protocols. Randomized protocols can guarantee consensus in the asynchronous communication model.

[3] The idea for the proof is that if there was an protocol that guarantees consensus after n communication rounds, the protocol would also guarantee consensus after $n - 1$ rounds, because the last message could be dropped. This reduces the number of required rounds to 0 by induction, which is a contradiction. Therefore, no such protocol can exist.

Finally, the **CAP theorem** [FB99] is a result which states an inherent tradeoff between the properties *consistency*, *availability* and *partition-resilience* in distributed systems. In contrast to the previously discussed results, the CAP theorem has its origin in the observation of real-world large-scale applications such as distributed databases. Therefore, it is less formalized and uses a different system model than the system model used for the standard consensus problem. For the CAP theorem, we consider a distributed system, which offers a read and write service on some data to its users [GL12]. Ideally, the service provides *consistency*, i.e., every read access to data gives the value that was written last for every read access. Ideally, the service also provides *availability*, i.e., every read or write access will be successfully completed within a short (to be defined) period of time. Ideally, the service is also *partition-resilience*, i.e., consistency and availability are still achieved, if the nodes of the system are partitioned so that no communication between the partitions is possible.

The CAP theorem states a system can only provide either consistency or availability in case of a network partition, but not both properties. For instance, in case of a network partition, a system can still provide availability, if nodes in both partitions execute the user's read and write operations. However, in that case the system does not guarantee consistency anymore, because a write access to a node in one partition remains unknown to nodes in the other partition, hence violating consistency. Contrary, a system could choose to cease operation in case of a network partition, hence sacrificing availability but guaranteeing consistency.

In this section we sketched a number of definitions, models, and results that serve as a theoretical basis for the understanding of permissionless blockchains. We will now describe the technical concept of Bitcoin as the first deployed instance of a permissionless blockchain and later discuss the applicability of the described models and results on Bitcoin.

2.2 Bitcoin

In 2008 a person or group using the pseudonym *Satoshi Nakamoto* published a paper titled *Bitcoin: A Peer-to-Peer Electronic Cash System* [Nak08] on the cryptography mailing list.[4] The paper proposes an electronic payment system, which is „based on cryptographic proof instead of trust“ [Nak08]. While electronic payment systems already existed (e.g., VISA, PayPal), these systems rely on trusted financial institutions. Furthermore, payments made using these systems are reversible in case of dispute, which requires merchants to „hassle [their customers] for more information than they would otherwise need“ [Nak08].

In order to illustrate the tasks required for operating a payment system, let us consider the example of a traditional, simplified bank. The bank has a set of customers that have their account run by the bank and that wish to transfer funds to other customers. The bank's task threefold: First, its customers needs to be able to send transactions to the bank, e.g., on paper or through online banking. Secondly, when a customer has

[4] `http://satoshi.nakamotoinstitute.org/emails/cryptography/1/`

sent a transaction to the bank, the bank has to validate the authenticity of the transaction. Traditionally, this is done by verifying the signature on a printed document or by checking the PIN/TAN used for online banking Thirdly, the bank has to keep track of all accepted transactions and has to apply them to the customers' accounts.

Customers have to trust the bank with regard to several aspects. First, the bank could retroactively temper with transactions, e.g., remove old transactions from a customer's ledger. Secondly, the bank could not validate incoming transactions properly. Even worse, the bank could create transactions in the name of their customers. Finally, the bank could suppress certain transaction and simply not accept them, although the transaction is valid. Traditionally, customers trust banks, because banks are subject to strict regulations and are incentivized to honestly perform their business.

We will now describe how Bitcoin provides a payment system without relying on a trusted entity such as the bank from our example. The mechanisms used by Bitcoin can be divided into three layers. The application layer handles the actual financial transactions, i.e., it allows users to transfer fund and also allows validation of transactions. The consensus layer achieves consensus on the set of accepted transactions that were created by the application layer. Finally, the network layer enables the communication between participating peers, which is required for the functioning of the upper layers. We will now introduce the mechanisms used in each layer.

2.2.1 Application Layer

Let us for now ignore the consensus layer and the network layer, and assume that there exists a system, which is capable of storing some system state in a consistent way. We will first describe the data structures used by the application layer of Bitcoin and then discuss the properties of the application layer.

Transactions

The main data structure of the application layer of Bitcoin is the *transaction*. Transactions can be created by participants of the system and allow the modification of the system state. Actually, the (application layer) system state consists exactly of the set of transactions that were accepted by the system.

Each transaction is identified by its hash value (i.e., the SHA-256 hash value computed over a serialized representation of the complete transaction). A transaction can contain several *inputs* and several *outputs* (Figure 2.1). Each input contains a reference to exactly one output of a previously accepted transaction. Each reference consists of the hash value of a previous transaction and the index of the referenced output. Additionally, each input contains an executable script denoted *ScriptSig*. Each output contains the value of the output (i.e., the amount of bitcoins to be transferred) and an executable script denoted *ScriptPubKey*.

In order to be valid, a transaction has to satisfy several conditions:

- All inputs have to reference *unspent* outputs of previously accepted transactions. *Unspent* means that no transaction has already been accepted that also references the output in question.

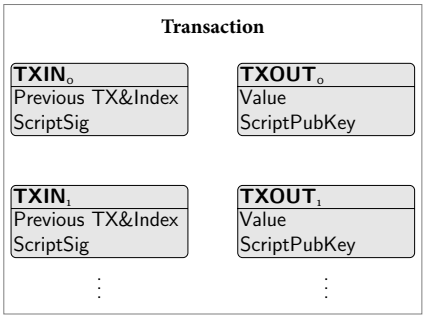

Figure 2.1: Elements of a Bitcoin transaction.

- The sum of the values of the outputs must be equal or smaller than the sum of the values of the outputs referenced as inputs.
- For all inputs, the *ScriptSig* and the *ScriptPubKey* of the referenced output are evaluated; this evaluation has to return true.

Scripts

We will now describe the purpose of the scripts in Bitcoin, and explain how scripts are evaluated. Both scripts are part of the access control mechanism that is used to ensure that valid transactions can only be created by authorized users. The idea is that the ScriptPubKey, which is part of each output, states conditions, which the ScriptSig has to satisfy in order for the transaction to be valid. An example for such a condition is to require a transaction to be signed using a private key that matches a given public key. While such a condition could also be hardcoded into the system, the use of scripts allow much more flexibility in the definition of the used access control policy.

Bitcoin scripts are written in a stack-based programming language, which allows the implementation of simple flow control structures, data manipulation operations, and cryptographic operations such as the calculation of hash values and signatures. Bitcoin scripts are, however, not Turing complete, because it is not possible to implement jumps or loops in Bitcoin scripts. While this property limits the expressiveness of the programming language, it makes it possible to trivially give an upper bound on the required execution time for each script (derived from the total number of operations in a script). The ability to give an upper bound for the execution time of a script is important, because scripts with very long execution times (e.g., infinite loop) could be created by malicious users for denial of service attacks.

In order to check, whether a transaction satisfies the access control policy specified in the ScriptPubKey of another transaction, whose outputs are referenced as inputs in the transaction, the ScriptPubKey and the ScriptSig have to be executed. Figure 2.2 visualizes how both scripts interact with the stack during execution. First, the ScriptSig is executed, which in the example pushes two data item on the stack (a signature `<sig>` of the transaction and a public key `<pubKey>`). Both data items are part of the script and are specified by the creator of the transaction. Then, ScriptPubKey is executed while preserving the content of the stack as it was at the end of the execution

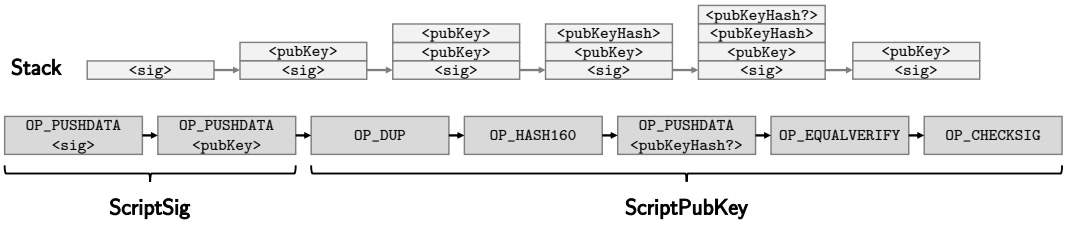

Figure 2.2: Bitcoin script: Pay To PubKey Hash (P2PKH)

of ScriptSig. This way, ScriptSig can push a solution onto the stack, after which ScriptPubKey can verify the correctness of the solution.

In the example depicted in Figure 2.2 ScriptPubKey checks whether the redeeming transaction (i.e., the transaction referencing the unspent) is correctly signed with a signature corresponding to a public key, which has a specified hash value. In order to perform this validation, ScriptPubKey first duplicates the public key that was pushed by the ScriptSig on the stack and hashes the public key. Then, it pushes the required hash value of the public key (`<pubKeyHash?>`) on the stack and compares both hash values. `<pubKeyHash?>` has been previously specified by the creator of the redeemed transaction. If that comparison fails, the execution of the script is aborted and the redeeming transaction is regarded invalid. Otherwise, the correctness of the given public key and the signature of the transaction is validated. If this validation is positive, the script returns true and the transaction can be accepted (assuming all other required conditions were positively evaluated). This kind of script is one of the most commonly used script pattern in Bitcoin, called *Pay To PubKey Hash* (P2PKH).

Properties

We will now discuss the properties of the application layer of Bitcoin. First, the set of accepted transactions can be represented as a *transaction graph*, with each vertex representing a transaction, and each directed edge representing one reference between the input of a redeeming transaction and the output of a previous transaction. Because every transaction input references the output of a previous transaction, there exists a partial ordering between transactions. Therefore, the resulting transaction graph is a directed acyclic graph (DAG).

The set of unspent transaction outputs (*UTXO set*) can also be directly derived from the set of accepted transactions by enumerating all transaction outputs that have not been redeemed by any transaction, yet. While the set of accepted transactions contains complete historic information about all past transactions, the UTXO set contains all information required to validate new transactions.

In order to illustrate the functioning of Bitcoin from a user's perspective, let us consider the example of Alice, who wants to transfer a certain amount of money (say, one bitcoin (1 BTC)) to Bob. Obviously, in order to be able to transfer 1 BTC to Bob, Alice has to *own* at least 1 BTC. Ownership of bitcoins is defined by the permission to spend certain bitcoins. This means, the current UTXO set has to contain outputs with a value of at least 1 BTC that Alice is able to spend. If that is the case, Alice can

create a transaction that has one or more input, so that the sum of input values is at least 1 BTC. For each input in her transaction, Alice has to provide a valid ScriptSig. The transaction will also have an output that transfers the ownership of 1 BTC to Bob, by requiring redeeming transactions to be signed with Bob's private key (i.e., Bob's public key is encoded in the output as shown in Figure 2.2). This means that Bob has to tell Alice his public key. In case the input values sum up to more than the amount that Alice wants to pay to Bob, Alice can add an additional output to the transaction, which sends the *change* (i.e., the difference between the sum of input values and the amount to be paid to Bob) back to a public key under her own control.

The example illustrates that the described system can be used for payments between participants of the system. The system also makes it possible for everyone with access to the set of accepted transactions to validate every transaction. However, until now we have assumed that there is a system that is capable of storing the system state (i.e., the set of accepted transactions) in a consistent way. Furthermore, the described system has to be somehow bootstrapped: If every transaction input refers to the output of a previous transaction, there would be an infinite chain of transactions. Both issues are solved in the consensus layer of Bitcoin, which will be described now.

2.2.2 Consensus Layer

Recall that Bitcoin aims at providing payment functionality without the need for a trusted third party. In the previous subsection we described the application layer of Bitcoin, which provides payment functionality, but requires the consistent storage of the set of accepted transactions. Without a single trusted third party, the task of storing the system state has to be accomplished by multiple parties in a *decentralized* way. With multiple parties storing the state, the problem of (in)consistency arises.

Consider a situation in which the set of accepted transactions is stored at two sites (s_1 and s_2). Then, two new transactions t_1 and t_2, which both redeem the same (unspent) output but differ in their outputs, are created and t_1 is sent to s_1 and t_2 is sent to s_2 at the same time. Both sites will accept the transaction sent to them, because both transactions are valid. However, both sites now store a different system state, which means that there are different opinions on who owns certain Bitcoins. The fact that a user can create multiple transactions that are all valid on their own, but conflicting in the sense that they are redeeming the same outputs, is called *double spending problem*. The consensus layer has to ensure that the system state is consistent among all participants. This is achieved by ensuring that only one of the double spending transactions becomes part of the system state.

Assume for now that there is a set of peers that can communicate via some peer-to-peer (P2P) network, which we will describe later. If a user wants to issue a payment, the user creates a transaction and publishes the transaction on the P2P network, which will transmit the transaction to all peers. Because of the double spending problem and because of network latencies, different peers can receive a different set of transactions. We will now describe how the consensus layer of Bitcoin achieves consensus on the set of accepted transactions. For this, we first introduce the used

mechanisms, then explain how the mechanisms are combined in Bitcoin, and finally discuss properties and assumptions of Bitcoin's consensus layer.

Mechanisms

Merkle Tree A Merkle tree [Mer80] is a data structure proposed by Ralph Merkle in 1980, which enables the efficient verification of the integrity of large data structures. Consider a data structure consisting of n single elements. A straightforward way to ensure integrity of that data structure is to calculate a cryptographic hash value of the complete data structure. One drawback of this approach is that it is not possible to verify the integrity of one single element without calculating the hash value of the complete data structure, which requires access to the complete data structure. Merkle trees address this issue by individually hashing each element of the data structure and then combining the resulting hash values into a tree structure (cf. Figure 2.3). This structure allows verification of any element of the data structure with access to only $\log_2 n$ hash values from the tree, instead of the complete data structure.

Blockchain A blockchain is a data structure that guarantees integrity of a growing amount of data by successively hashing blocks of data. Each block contains the data itself and the hash value of the previous block. This creates a chain of blocks with the property that the newest hash value is sufficient for the verification of the integrity of the complete data. It is not possible to alter any data (i.e., modify, remove or add data) without affecting all subsequent hash values.

Proof-of-Work Proof-of-work is a mechanism originally proposed to mitigate email spam by requiring the sender of an email to solve a computationally expensive task in order to send the email [DN92]. The task should be hard to solve, however, its solution should be easy to verify. A common task for proof-of-work is finding a partial hash inversion for a given data (with some data allowed to be modified). For instance, an email server could accept incoming emails only, if the hash value of the email is smaller than a defined threshold. In order to achieve this, the sender of the email has to modify the content of the mail (e.g., a header entry designated for that purpose) so that its hash value satisfies the condition. Because the result of the hash function cannot be predicted by the sender, there is no method to find an accepted hash value except for repeatedly changing the email's content until the hash value is accepted. While the sender has to put a substantial effort into finding a solution, the receiver can easily verify the solution's correctness by hashing the email.

Combination of Mechanisms

We will now describe how the discussed mechanisms are combined in the consensus layer of Bitcoin. The main data structure of the consensus layer is the *block* (Figure 2.3). A block consists of a header part and the Merkle tree of the set of transactions, which are included in the block. The transactions contained in a block cannot be changed without also changing the Merkle root, which is part of the block header. The block header also contains the hash value of the previous block. This creates a

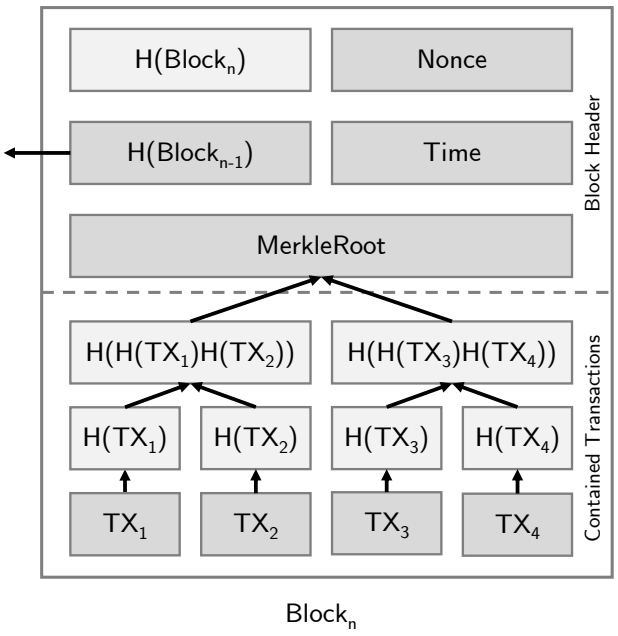

Figure 2.3: Elements of a Bitcoin block. Elements in light gray (i.e., $\mathsf{H}(\mathsf{Block}_n)$ and the Merkle tree) are not part of serialized blocks but can be computed from the other elements.

blockchain, defined by the references from each block to its predecessor. In order for a block to be valid, all transactions included in that block have to be individually valid (cf. Section 2.2.1) and all transactions have to be non-conflicting to all transactions included in the blockchain defined by the current block and all of its predecessors. Furthermore, the hash value of the block header ($\mathsf{H}(\mathsf{Block}_n)$) has to be smaller than a certain value, i.e., a valid block contains a proof-of-work.

Blocks are created in the process of *mining*. As new transactions are continuously broadcast on the P2P network, certain peers (*miners*) verify these transactions, include them in a new block and continuously try to solve the proof-of-work required for the block to be valid. If a miner finds a solution to the proof-of-work puzzle, it publishes the block on the P2P network. Other miners then verify the correctness of the block and start working on a new block, i.e., disregard all transactions that are already part of the received block and update the reference to the previous block.[5] The mining process creates a blockchain that grows in length and contains a set of non-conflicting, valid transactions.

The miner of a block is rewarded for the computational effort put into finding a valid proof-of-work by allowing the miner to create a special *coinbase transaction* and include that transaction into the mined block. A coinbase transaction has no inputs

[5] The used hash-based proof-of-work mechanism is *memoryless*, i.e., the probability of finding a solution in the next trial is independent of the effort that has already been put into finding a solution in the past. Therefore, changing the content of the block does not change the expected time when a solution to the proof-of-work puzzle is found.

(i.e., it does not redeem any previous transaction), yet it may contain outputs up to a certain value. The reward of the miner consists of two parts: transaction fees and block rewards. Transaction fees can be (implicitly) specified by users by creating a transaction with a sum of output values being strictly smaller than the sum of input values. A miner is rewarded with the sum of all transaction fees of all transactions included in the mined block. Furthermore, a miner is rewarded with the block reward, which is a certain amount of bitcoins specified by protocol rules. The block reward was initially 50 BTC, and halves every 210,000 blocks (roughly every four years). As of 2018, the block reward is 12.5 BTC. In the transaction graph, coinbase transactions form the beginning of every transaction chain.

In order to account for changes in the overall computing power of the miners, the *difficulty* of the proof-of-work puzzle (i.e., the maximum value the block header hash is allowed to have) is adapted every 2,016 blocks so that on average every 10 minutes a new block is found by a miner. Because blocks carry a timestamp, the calculation of the difficulty can be carried out by each peer individually and will result in the same difficulty value for all peers with access to the same blockchain.

The transactions contained in the created blockchain could be seen as the set of accepted transactions. However, it is possible that two miners each generate a new block at the same point in time (*blockchain fork*). Both blocks can each contain different sets of transactions, and transactions contained in each block can be conflicting. Therefore, transactions that are contained in a block cannot be necessarily regarded as accepted, and there has to be a method for resolving the inconsistency.

Resolving the inconsistency is based on the specification made by Bitcoin that miners should always work on the blockchain, in which the most computational effort has been put into. When a blockchain fork occurs, there are two blockchains, in which the same computational effort has been put into. Therefore, miners will choose one blockchain at their discretion (e.g., based on which block they received first) and work on extending that chain. Once a miner finds a new block on either chain and publishes that block, it increases the computational effort that has been put into that blockchain. Therefore, other miners will now switch their mining effort to that chain, adding even more blocks to the chain. This means, that while there can be short periods in which two or more blockchains are of equal length, eventually one blockchain will become the single longest chain, which serves as a reference for the accepted transactions.

Assumptions and Properties of the Consensus Layer

In the beginning of this section we laid out the example of a centralized bank, which could attack the system by improperly validating transactions, removing old transactions, and by suppressing transactions. We will now discuss how Bitcoin addresses these threats.

First, the application layer of Bitcoin and the public availability of the blockchain makes it possible for every participant to validate transactions. Therefore, if miners decide to include invalid transactions into their blocks, all other peers can detect this and will reject the block.

Secondly, consider the following scenario: An adversary creates and publishes a

transaction t that moves funds to a merchant, who delivers some goods after seeing the transaction being included in the blockchain in block b_n. After reception of the goods, the adversary wants to remove the transaction from the blockchain, thus regaining control of the funds that were previously sent to the merchant. In order to succeed, the adversary has to create a blockchain, which does not include the transaction t and which is longer (i.e., having more computational effort being put into) than the existing blockchain. If the adversary has some computational power, the adversary could replace the block b_n, which contains the transaction t, by a block $b_{\hat{n}}$ that does not contain the transaction t. The adversary could publish that block, however, because other miners already created a larger number of new blocks succeeding b_n, other peers will ignore the block created by the adversary. However, if the adversary controls more than 50 % of the computational power of all miners, the adversary is able to create blocks faster than the honest miners and will eventually be able to *overtake* the honest miners, i.e., creating a blockchain which is longer than the original chain. Once the adversary managed to create a blockchain, which is longer than the original blockchain, all peers will accept the data contained in the adversary's chain as the data being agreed on, effectively undoing the transaction t.

The example illustrates that the consensus layer of Bitcoin is not secure against adversaries with more than 50 % of the computational power of all miners.[6] The example also illustrates that consensus is an emergent property in permissionless blockchains, i.e., there is no single point in time when the system has *settled* and a transaction is ultimately accepted. However, the effort to remove a transaction from the blockchain increases with the number of mined blocks referencing the block that includes the transaction. Therefore, the required number of subsequent blocks before regarding a transaction as settled is a tradeoff between performance (i.e., confirmation time) and security against a double spend attack. It is common behavior of Bitcoin client software to treat a transaction included in a block with 5 subsequent blocks as settled. This corresponds to an average confirmation time of one hour, assuming an average block interval of 10 minutes.

Furthermore, the example shows that immutability is not an absolute property of blockchains. The content of permissionless blockchains can be changed by investing enough computational power. The term *Mutable-By-Hashing-Power* has been suggested to account for this characteristic [CdLSJ+17].

2.2.3 Network Layer: Peer-to-Peer Network

So far, we have assumed that all Bitcoin clients have the ability to communicate via some broadcast communication medium in order to exchange transactions and blocks. The network layer of Bitcoin creates a peer-to-peer network, which serves as such a broadcast communication medium. We will now briefly describe how the network is established and how peers communicate. A more detailed description and

[6] If an adversary does not immediately publish new blocks, but instead withholds blocks for a certain duration, even adversaries with less than 50 % of the mining power can successfully attack Bitcoin's consensus layer [ES14].

discussion of the network layer mechanisms can be found in Section 3.3.

Attachment Strategy

In order to establish connections to other peers, clients need IP addresses of other reachable peers. Providing such IP addresses to peers is the task of Bitcoin's peer discovery mechanisms. The first mechanism used by Bitcoin is an out-of-band peer discovery mechanism, which allows clients to query DNS servers that provide IP addresses of reachable peers. Once a client is connected to at least one remote peer, it can query remote peers for additional IP addresses of other peers using an in-band peer discovery mechanism.

Based on the received IP addresses, Bitcoin clients establish connections to other peers. Clients establish a certain number of outbound connections to remote peers. These peers are selected mostly at random, only avoiding establishing too many connections to peers within a small IP address range. However, there are many more possible strategies for neighbor selection, i.e., to which IP addresses connections should be established (cf. Section 3.3.1).

Communication Strategy

After a client has established connections to other peers, it needs to synchronize its copy of the blockchain. Other peers, which already participated in the network, maintain the current blockchain and deliver it to the newly joined client upon request. Recall that the client can validate the correctness of all blocks (e.g., valid proof-of-work, valid transactions) on its own. Furthermore, by asking multiple peers for *their* blockchain, the client can detect if a malicious remote peer presents an alternative (valid) blockchain with less accumulated difficulty.

Once a client is in possession of the current blockchain, it can also validate transactions and new blocks that are flooded through the network. Flooding is implemented in three steps. A client that creates a transaction announces its hash value to its neighbors via an `INV` message.[7] If the announced hash value is unknown to the remote peer, it requests the actual transaction via a `GETDATA` message. Finally, the creating peer sends the transaction using a `TX` message. After validation of the transaction, the receiving peer proceeds as if the transaction was created locally and announces the transaction hash to its neighbors (except for neighbors that have already announced the transaction's hash itself). Blocks are flooded accordingly, but using a `BLOCK` message in the last step.[8]

Properties

In order to serve as a broadcast medium for the consensus layer of Bitcoin, the network layer should transmit messages to all participating peers within a certain period of time. We will now discuss whether Bitcoin's network layer satisfies this requirements.

In order to transmit messages to all participating peers, the resulting network graph has to be connected. As peers may leave the network or adversaries may actively sabotage connections, it cannot be guaranteed that the network graph is always

[7] One `INV` message may contain the hash values of multiple transactions or blocks.

[8] There are two additional methods for block relay, which accelerate dissemination [Daf15, Cor16].

connected. However, the probability of a network split can be reduced close to zero by increasing the number of connections each client establishes.

The maximum allowed delay of a message differs for blocks and transactions: If transactions are delayed for a longer period, transactions may not be included in a block, hence the time until the confirmation of a transaction increases. However, the delaying of transactions has no effect on the consensus layer. Contrary, large delays in the transmission of blocks cause miners to work on outdated blocks, which reduces the overall security of the consensus layer [GKW+16]. In Bitcoin, messages can be theoretically delayed arbitrarily by network and processing delays. Although most messages are transmitted within at most a few seconds (cf. Chapter 4), no strict upper bound on the maximum message delay can be given.[9]

While Bitcoin's network layer cannot guarantee the required properties, it provides the properties with very high probability. Even if the properties do not hold for a short time period, e.g., due to a DoS attack, the system state will only be temporarily inconsistent, as the consensus layer can recover as soon as the required connectivity is reestablished. However, if participants rely on a temporarily inconsistent state without knowing of the inconsistency, or if participants know of the inconsistency but require the availability of the system (cf. CAP theorem), substantial damage can occur. This lack of guarantees by the network layer combined with a substantial risk in case of failure requires a precise understanding of the network layer's properties.

Finally, note that communication channels between peers are not authenticated or encrypted by default[10], i.e., messages are transmitted in plaintext. Authentication of remote peers or messages is not necessary, as the authenticity of all messages transmitted by remote peers can be validated without additional information: Transactions refer only to information (e.g., public keys) that is contained in the blockchain, and blocks contain a proof-of-work, which can be validated on its own. Encryption is not required as all data is published anyway.

2.3 Discussion

So far we have described classical definitions and results, and gave an introduction into the mechanics of Bitcoin. In this section we first discuss how Bitcoin relates to these classical definitions and results. Then we describe the impact of Bitcoin on research and deployed systems.

2.3.1 Relation to Consensus Models and Key Results

In Section 2.1 we briefly introduced several system models that are commonly used in the analysis of the consensus problem. We will now discuss which of these models are suited for the analysis of Bitcoin. We emphasize that the relation of Bitcoin to

[9] Nakamoto states that „messages are broadcast on a *best effort* basis" [Nako8], a term, which is commonly associated with internet communication in general.

[10] There are proposals for encryption of connections between peers [Sch16b] and authentication of peers [Sch16a].

common definitions in the field of distributed systems is subject to ongoing research and debate. Therefore, there is no common understanding of *the correct* system model for Bitcoin. Based on the discussion of suitable models we will address the applicability of the fundamental results on Bitcoin.

Models

Communication Model As stated above, communication between peers is unreliable in the sense that messages can get lost, and message delays can be theoretically unbounded. This suggests that an asynchronous communication model with omission failures matches the behavior of Bitcoin. In practice, however, communication delays are bounded with overwhelming probability by a few seconds. The average interval between the creation of two new blocks in Bitcoin is 10 minutes, i.e., much longer than the maximum message delay (cf. Chapter 4). Furthermore, while omission failures (packet drops) are possible, they rarely affect the consensus protocol, because the used Transmission Control Protocol (TCP) transparently enhances communication reliability using mechanisms such as sequence numbers and retransmissions. Therefore, one could also argue that (almost) all nodes have received the previous block before a new block was created, resulting in a synchronous communication model.[11] Consequently, synchronous as well as asynchronous communication models were used for Bitcoin in the past (synchronous [MLJ14], partially synchronous[12] [GKL15], asynchronous with a-priori bounded delays [PSS17]).

(In-)Determinism Mining is an indeterminstic process, i.e., finding a solution to the proof-of-work puzzle is a random process. Therefore, Bitcoin should be modeled as an indeterministic protocol (e.g., [MLJ14, GKL15, PSS17, Wat16]).

Node Failure Model Because the P2P network is open for everyone to join, adversaries cannot be prevented from joining the network. The behavior of the adversarial peers can be freely defined by the adversary. Therefore, no assumption on the behavior of faulty nodes can be made, i.e., Byzantine node failures have to be assumed. Furthermore, adversaries can join the network with a theoretically unlimited number of peers. Therefore, no upper bound on the number of faulty nodes can be given.

Consensus Problem Definition The standard consensus problem definition assumes that all nodes start with an initial value and agree on one value guaranteeing termination, agreement, and validity. In Bitcoin, the initial value of nodes is a set of published transactions that should be included in a block. However, Bitcoin provides an ongoing service, i.e, instead of solving a single instance of the consensus problem, the consensus problem is continuously solved with new transactions. Therefore, termination has to be interpreted with regard to one single problem instance from the

[11] While transactions are created more frequent, only the creation of blocks is relevant for the consensus protocol. Transactions can be seen as proposals to be included into the block.

[12] In the partially synchronous model a maximum message delay is assumed, however no full synchrony is assumed.

past, i.e., nodes have to decide on one set of accepted transactions in finite time after publication of these transactions. Intuitively, the termination property translates to *regarding a transaction as accepted* (i.e., *deciding*) once the transaction is included in a block and a number of subsequent blocks were appended to that block.

As previously discussed, agreement cannot be guaranteed, because it cannot be completely ruled out that an adversary creates a longer blockchain even after honest nodes terminated. Therefore, the agreement property is usually weakened by allowing disagreement with negligible probability (e.g., [MLJ14]). Allowing a failure with negligible probability is a common approach in cryptography and is required to proof the security of any cryptographic scheme that does not provide information-theoretic security [KL14]. Furthermore, the validity property is often weakened because adversaries with non-negligible computing power have a non-negligible chance (according to their relative computing power) of mining a block [GKL15].[13]

Because the iterative process of blockchains is only weakly captured by the standard consensus problem definition, other problem definitions were proposed. For instance, Pass et al. [PSS17] proposed a property denoted *T-consistency*: T-consistency requires that honest nodes agree on the current chain, except for a number of T unconfirmed blocks at the end of the chain.

Participation Model Classical work on consensus systems typically assumes that the set of participating nodes is known a-priori. Requiring nodes to be somehow *permissioned* to participate in the consensus process is required in order to prevent a theoretically unlimited number of faulty nodes to join the network. Such a Sybil attack [Dou02] would defeat any protocol that relies on any upper bound on the number of faulty nodes. Bitcoin allows nodes to freely join the network (permissionless). However, Bitcoin limits the ability to contribute to the consensus process to the computational resources of the node. This renders Sybil attacks ineffective, as a larger number of nodes does not benefit the adversary. In contrast to a maximum number of faulty nodes that can be tolerated by a consensus protocol, Bitcoin requires that the computation power share of the adversary is bounded by 50 %.

Applicability of Results

Based on Bitcoin's relation to consensus models and assumptions, we will now discuss the applicability of the key results discussed earlier in Section 2.1.

The **FLP impossibility result** is only applicable for determinstic protocols. As discussed, Bitcoin is indeterminstic, hence, the FLP result is not applicable.

While the impossibility result of the **Two Generals Problem** is applicable because of unreliable communication, it can be easily circumvented: The Two Generals Problem requires a protocol to *guarantee* consensus with a probability of 1. However, protocols can still reach consensus with a negligible failure probability, which is the case for Bitcoin.

[13] While this is technically a violation of the validity property as the adversary determines the output value, it does not pose a problem because the correctness of the block mined by the adversary can be verified by every honest node.

Bitcoin can be seen as a probabilistic **Byzantine Agreement** (BA) protocol, assuming the adversary's computational power is bounded by 1/3 [GKL15]. Satoshi Nakamoto gave an informal description of the mapping between Bitcoin and the Byzantine Agreement problem.[14] However, the additional assumption on the adversary's computational power prevents results in the standard BA model from being applied to Bitcoin (e.g., the upper bound of faulty nodes may exceed 1/3 in Bitcoin).

The **CAP theorem** applies to Bitcoin in the following way: A read operation translates to a client looking up state from its local copy of the blockchain. A write operation translates to the creation and publication of a new block by a miner. If a network partition occurs, every peer can still access its local blockchain copy, hence availability is guaranteed. However, miners may concurrently create new blocks unbeknown to nodes in the other network partition, creating inconsistent state. However, after the network links between the partitioned nodes are reestablished, peers exchange the newly mined blocks and the longest blockchain will become the agreed system state, resolving the temporarily inconsistent state. Therefore, Bitcoin is a system that ensures availability but not consistency in case of a network partition.

The discussions on the applicability of classical models and results show that while some existing models can be easily adapted to model Bitcoin, most fundamental results are not directly applicable.

2.3.2 Bitcoin's Impact

We will now discuss the effect of the publication of Bitcoin's concept on the research of decentralized systems and the development of technological innovations inspired by Bitcoin.

With the publication of its concept and the real-world deployment, Bitcoin itself became an object of investigation. This research can be roughly categorized into three categories: empirical research, conceptual research, and the analysis of attacks. The public nature of the blockchain data and the P2P network enables *empirical* research that analyzes data generated by Bitcoin's users and peers. For instance, transaction data stored in the blockchain has been used to assess anonymity of users [RH13], and P2P information propagation has been analyzed [DW13]. In Chapter 4 we present empirical measurements of the Bitcoin P2P network.

Many *conceptual* research contributions focus on analyzing the properties exhibited by Bitcoin. As discussed before, most classical results are not applicable to Bitcoin, hence the development of adapted models was required [GKL15, Mil16]. Furthermore, as Bitcoin couples the consensus layer with the application layer by rewarding miners for their work, economic and game theoretic aspects have to be considered [ES14].

Finally, many more or less severe *attacks* on Bitcoin have been discussed, and countermeasures have been pointed out (e.g., [HKZG15]). In Chapter 3 we present a survey of attacks on the network layer of Bitcoin. Furthermore, we analyze methods for inferring the network topology of the Bitcoin P2P networks in Chapter 6 and analyze a method for attacking the anonymity of Bitcoin users in Chapter 7.

[14] `https://satoshi.nakamotoinstitute.org/emails/cryptography/11/`

Bitcoin has not only been a subject of investigation, but also subject to a large number of proposed improvements. Some of the main challenges of Bitcoin are the limited scalability and weak anonymity.[15] These challenges were addressed by a large number of proposed and/or implemented improvements, ranging from technologies building on top of Bitcoin's application layer (e.g., payment channels [DW15, PD16]), to improvements of the transaction structure (e.g., Segregated Witnesses [LW16]), and to proposals for improved anonymity on the network layer (e.g., Dandelion [FVB+18]).

Furthermore, Bitcoin sparked the development of new concepts for permissionless blockchains with modified or extended functionality. On the one hand, there is a large number of permissionless blockchains that were derived directly from Bitcoin and only modify some parameters, such as the maximum block size, or the average block generation interval (e.g., *Bitcoin Cash*). On the other hand, there are systems that build on the concept of Bitcoin, but significantly modify its functionality, e.g., by introducing certain privacy preserving transaction types [MGGR13]. Finally, completely new concepts for permissionless blockchains, fundamentally differing from Bitcoin's concept were proposed and deployed. The most prominent example for such a system is Ethereum [Woo14], which enables decentralized computation of almost arbitrary code, thus allowing the decentralized enforcement of Smart Contracts [Sza97].

2.3.3 Conclusion

In this chapter we laid out the scientific foundations for permissionless blockchains, introduced the concept of Bitcoin, and discussed the relationship between classical research results and permissionless blockchains. We showed that the concept of Bitcoin combines well-established mechanisms in a way that makes the resulting system similar to existing (consensus) systems, yet different enough so that most established theoretical results are not applicable to permissionless blockchains. Especially a blockchain's network layer does not provide any formal guarantees, which requires a thorough understanding of its properties in order to assess the properties of the complete system. In the next chapter we contribute to the understanding of the network layer of permissionless blockchains by defining its requirements, by providing a systematization of known attacks, and by surveying its design space.

[15] The first responses to the Nakamoto's publication of the Bitcoin concept on the mailing list questioned the scalability and anonymity of Bitcoin (`https://satoshi.nakamotoinstitute.org/emails/`).

3
Network Layer Requirements and Design Space

The content presented in this chapter has been previously published in [NH18].

So far, the network layer of permissionless blockchains has been analyzed primarily with regard to specific attacks such as network based deanonymization attacks (e.g., [FV17, BKP14]), double spend attacks (e.g., [KAC12]), or eclipsing attacks (e.g., [AZV17, HKZG15]), or in an empirical way (e.g., [DPSHJ14, DW13]). While these works help in the design and research of the network layer by pointing out certain weaknesses and propose countermeasures against them, they neither make requirements of the network layer of permissionless blockchains explicit, nor do they show the complete design space or point out the inherent tradeoffs for some design decisions. This coincides with the observation that many design choices made in the network layer of client implementations seem ad-hoc and without sufficient reasoning.[1]

This chapter aims at supporting the design and research of the network layer of permissionless blockchains by (1) analyzing the requirements of the network layer of permissionless blockchains, and (2) giving a comprehensive survey of the design space of the network layer of permissionless blockchains. The set of requirements contains functional, non-functional, and security requirements, which are derived based on a survey of relevant attacks. The design space survey includes design aspects and options that are implemented in permissionless blockchains or proposed in the literature, and shows which tradeoffs are implied by design decisions.

The network layer of permissionless blockchains is related to two classes of systems that have been analyzed in the past: unstructured peer-to-peer networks and

[1] This does not make them bad choices, however, knowledge of which choices are possible and knowledge of the inherent tradeoffs between design choices could result in better choices or justified design decisions.

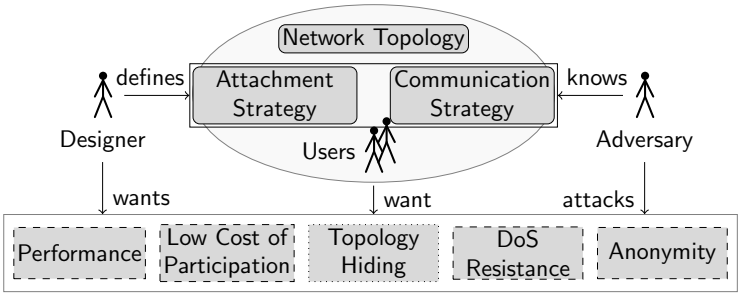

Figure 3.1: The network layer of permissionless blockchains is characterized by the P2P network topology, the client's attachment and communication strategies, and the users' behaviors. The requirements include performance, low cost of participation, anonymity and DoS resistance, and the intermediate goal network topology hiding [NH18].

anonymity providing networks. The use of flooding or gossip protocols makes the network layer of permissionless blockchains unstructured P2P networks, which have been used for decades (e.g., Gnutella [Rip01]) and were extensively analyzed (e.g., [LCP+05, JC10]), however, mostly from a performance perspective or with adversary models not matching the threat to blockchain systems. Although anonymity providing networks (e.g., Tor [DMS04]) have different requirements regarding information propagation than blockchain based systems, they are similar in the considered adversary models and security requirements. Commonly considered requirements in anonymity providing networks are high performance, low bandwidth cost, resistance to traffic analysis, and resistance to catastrophic denial of service (DoS) [BMS01], which we will use as a basis for our analysis.

Figure 3.1 gives a high level overview of the aspects, requirements and actors affecting the network layer of permissionless blockchains. The blockchain's P2P network is characterized by its network topology and the behavior of its peers, which is determined by the client software behavior (i.e., its communication and attachment strategy) and the user behavior. In addition to the (non-security) requirements *performance* and *low cost of participation* and the security requirements *anonymity* and *DoS resistance*, we also consider *network topology hiding* as an intermediate security requirement, because many attacks rely on knowledge of the network topology.

3.1 Related Work

We will briefly discuss related work that covers network security in blockchain systems or P2P networks on an abstract level. Discussion of related works with more focus (e.g., specific attacks) is given in Sections 3.2.3 and 3.

Gervais et al. [GKW+16] give a thorough security analysis of proof-of-work blockchain systems, however, their focus is on the consensus layer (i.e., block generation) whereas the network layer is abstracted. Troncoso et al. [TIDH17] show a broader perspective covering numerous systems apart from Bitcoin and Tor, but also abstract

from the network layer. A recent paper by Delgado-Segura et al. [DSPSHJ+18] explores the characteristics of the peer-to-peer network established by Bitcoin, but abstracts from the design space of the network layer. Lua et al. [LCP+05] focus on file sharing peer to peer networks and leave out anonymity objectives and strong adversary models. There are several surveys covering Bitcoin in general [TS16, BMC+15], the security of Bitcoin [CKLR18], and the privacy and anonymity of permissionless blockchains [HJPS16, KL18].

3.2 System Requirements

Based on the sketch given in Figure 3.1 we analyze the requirements of the network layer of permissionless blockchains. Bitcoin serves as a prototype for the considered scenario. However, we emphasize that our definition also matches most other permissionless blockchains and parts of this chapter are also applicable to a wider range of unstructured P2P networks.

3.2.1 Functional Requirements

In contrast to a private blockchain or a permissioned blockchain, in a permissionless blockchain there is neither a restriction on the ability to read from the blockchain, which ensures public verifiability, nor a requirement for pre-established identities for write access to the blockchain [WG17]. In order for anyone to join the system, there need to be enough peers on the network that accept incoming connections. There may be peers that are not reachable (e.g., because they are behind a NAT), however, the **openness** of the system can only be guaranteed with a sufficiently large share of reachable peers. Although unreachable peers can only connect to reachable peers, they can still serve the system by increasing the network's robustness (cf. Section 3.3).

In order to provide public verifiability and allow peers to create blocks, all relevant data must be accessible by peers. Therefore, the main requirement of the network layer of blockchains is the **dissemination of information** among all participants. Peers need to be able to retrieve historic information from the network (e.g., when a new peer initially joins the network), but also need to stay informed continuously about new information. To ensure a fast dissemination of new information, a flooding or gossip mechanism is used, that broadcasts new information to all peers. There may be clients that are unable to process that amount of data (e.g., because they are running on a mobile device), which only get a subset of data relayed (e.g., SPV clients in Bitcoin [HC12]). These clients, however, put some trust in the peers they connect to and assume that these peers do not withhold messages.

In contrast to (communication) anonymity providing systems, which ensure confidentiality of the exchanged data between a certain sender-receiver pair, permissionless blockchains publish information to all participants. Therefore, the data is not encrypted, which makes each piece of information (e.g., a transaction) uniquely identifiable and distinguishable. Encryption of the connections between peers was proposed [Sch16b], however, without pre-established identities of peers, this approach

is vulnerable to man in the middle attacks as authenticity cannot be protected. Authentication of peers, which requires the out of band exchange of a public key, has also been proposed for Bitcoin [Sch16a, AZV17], and is implemented in Ethereum and Tor.

3.2.2 Non-Functional Requirements

Back et al. [BMS01] identified two non-functional requirements for anonymity providing networks: performance and low bandwidth cost. Although the network layer of blockchains and anonymity providing systems differ in many ways, both requirements still apply. We generalize the goal of a low bandwidth cost and require a low cost of participation, which implies low bandwidth costs.

Performance: Information dissemination should be fast. For a blockchain system, a slow dissemination of information implies a longer time until consistency on the information is reached. This can facilitate attacks on the application layer such as double-spends (cf. Section 3.2.3). Furthermore, the efficiency of proof-of-work blockchains depends on fast information dissemination [GKW+16]. A metric that quantifies the performance is the delay between the initial sending of a piece of information until the time that n percent of peers have received that information.

Low Cost of Participation: As permissionless blockchains aim to be open for participants to join the network, participants wishing to run a peer on the network should not be faced with unbearable costs.[2] The costs of running a peer include the required bandwidth, computation, and storage costs. Reducing these costs enables more users to run a peer, which improves the overall reliance of the system. On the other hand, a large number of peers make the consideration of scalability issues necessary. Metrics that quantify the cost of participation are the number of bytes a peer has to send and receive depending on the application layer load (e.g., the number of transactions) during a time period, or the required storage.

3.2.3 Security Requirements - Attack Survey

In order to derive the security requirements of permissionless blockchains, we systematize network based attacks on Bitcoin.[3] The resulting systematization of attacks is visualized in Figure 3.2 using the concept of attack trees [Sch99] (with OR nodes only). We will now briefly describe the structure of the systematization and the surveyed attacks.

As described in Chapter 2, the network layer of permissionless blockchains supports the consensus layer, which is used to execute some application logic (e.g., the Bitcoin payment system). While attacks may target the network layer, the *goals* of all surveyed attacks are located at the application layer, where some *aspect* of the application layer

[2] While users can participate in the blockchain without running a peer by delegating the communication with the network to trusted parties, we only consider the direct communication with the network by running a peer as participation from a network-level perspective.

[3] Our systematization does not include attacks without interaction with the network layer, e.g., selfish mining [ES14] or address clustering [RH13]. For a comprehensive survey of such attacks please refer to [CKLR18].

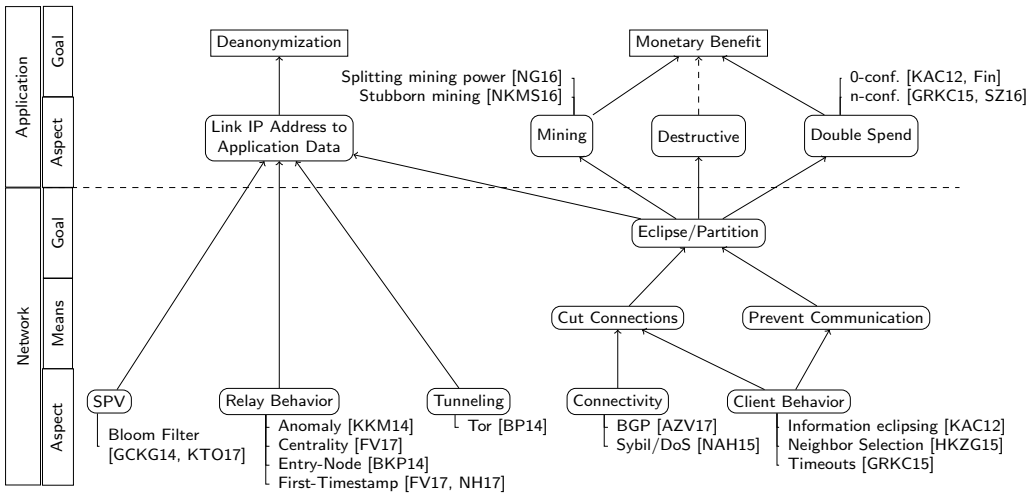

Figure 3.2: Known network-based attacks on permissionless blockchains at the example of Bitcoin, visualized as attack trees [NH18].

is attacked.[4] Attacks usually exploit some *aspect* of the network layer by certain *means* in order to achieve an intermediate *(network layer) goal*. This hierarchy represents the structure of our systematization of attacks shown in Figure 3.2.

We will now discuss the properties of the resulting attack tree and briefly cover selected attacks. One possible goal of an adversary is the deanonymization of users by associating network layer information (e.g., IP addresses) to application layer information (e.g., Bitcoin addresses). Several aspects of the network layer can be exploited for this purpose: Clients with limited resources that are unable to store the complete blockchain and are unable to process all transactions (*SPV Clients*) can request the forwarding of relevant transactions from remote peers by sending a Bloom filter [Blo70] to remote peers. The Bloom filter encodes the Bitcoin addresses of the SPV client in a probabilistic way. Gervais et al. [GCKG14] showed that adversaries can learn the Bitcoin addresses of SPV clients from their Bloom filters.

The transaction relay behavior of Bitcoin has been shown to be susceptible to deanonymization attacks in several previous works. For instance, it has been shown that anomalous relaying behavior of clients can facilitate the linking of IP addresses to Bitcoin addresses [KKM14]. Furthermore, an adversary with knowledge of the network topology can exploit properties of the propagation graph to identify the IP address of the sender of a transaction [FV17]. We provide an analysis a deanonymization method based solely on observing the first announcement of a transaction in Chapter 7. Finally, tunneling the communication to the Bitcoin network through Tor has been shown to enable deanonymization attacks [BP14].

Apart from deanonymization, another goal of an adversary could be to gain a monetary advantage by either earning disproportionally high mining rewards, by

[4] In this systematization we do not distinguish between the consensus and application layer as done in Chapter 2, because attacks targeting the consensus layer (e.g., the aspect *mining*) also ultimately target the application layer.

double spending funds, or by performing a destructive Goldfinger attack[5]. Destructive attacks can also be ideologically or politically motivated without any monetary incentive. All published attacks have in common that some form of interruption of information flow between peers on the network is required. This can be either the eclipsing of single peers from certain information or the complete partitioning of large parts of the network. Two general methods to achieve this have been proposed: Either to directly attack the connections between peers, i.e., after a successful attack the victim peer has no connections to other honest peers, or to prevent communication from and to victim peers, i.e., after a successful attack the victim peer has functioning connections to other peers but does not receive the required information over them. Preventing communication can be achieved by exploiting client behavior (e.g., [KAC12, HKZG15, GRKC15]). Cutting connections can also be achieved by exploiting client behavior, but also by directly attacking the underlying network stack (e.g., [AZV17, NAH15]).

Two observations can be made regarding the overall structure of the presented attack tree: First, no common adversarial network layer goal can be identified for deanonymization attacks, because the observation of any information enabling the linking between IP addresses to Bitcoin addresses is sufficient for deanonymization. Secondly, all attacks aiming at monetary benefit have the intermediate goal of preventing the information flow by partitioning the network or eclipsing certain peers. Therefore, we can identify two security requirements for the network layer: Prevent linkage of IP addresses and application layer information (**anonymity**) and prevent interruption of the information flow by eclipsing or partitioning (**DoS resistance**).

A number of attacks (e.g., [NAH15, FV17, BKP14]) require the adversary to know the network topology of the P2P overlay network. As this information is not publicly known and known to be hard to infer (cf. Chapter 6), an intermediate goal for an adversary can be to approximate the topology of the underlying P2P network. Although many attacks are still possible without knowledge of the network topology, lack of such knowledge causes a less precise attack, which generally requires more resources to be spent by the adversary. From the perspective of a designer of a permissionless blockchain, hiding the network topology can prevent subsequent attacks. Therefore, we also consider **topology hiding** as an intermediate security requirement.

Metrics that quantify topology hiding are the precision and recall with which a certain adversary can estimate connections of the network topology. These metrics will be used in Section 6. Metrics that quantify anonymity are the precision and recall with which a certain adversary can link network layer to application layer information. A metric that quantifies DoS resistance is the required amount of resources required to execute a DoS attack for a certain adversary.

[5] In a Goldfinger attack the adversary earns money by destruction of the system, e.g., by going short on the cryptocurrency or by blackmailing [KDF13].

3.2.4 Adversary Models

The attacks discussed in the previous section made various assumptions on the ability of the adversary. Therefore, also the discussed metrics of the security requirements depend on the adversary model. We will now discuss several aspects of adversary models.

For network-based attacks, the most critical aspect is the **network power** of the adversary. An adversary can easily run a peer that connects to all reachable peers in the network. Such *monitoring peers* have been previously shown to be able to connect to several thousand peers on the Bitcoin P2P network (cf. Chapter 4) using standard hardware and requiring a bandwidth of less than 100Mbit/s. Adversaries with more resources could perform a *Sybil attack* on the network by inserting a large number of peers to the network. As the cost of running a network peer should be low (cf. Section 3.2.2), the number of Sybil peers on the network could outnumber the number of honest network peers. The adversary could run the Sybil peers on own hardware, use cloud service providers, or use botnet services. Finally, an adversary with *access to core internet infrastructure* (e.g., ISPs, internet exchange points, intelligence services) is additionally able to monitor and manipulate traffic, e.g., by hijacking BGP routes [AZV17] or simply by dropping packets.

The **computation power** available to an adversary can be either modeled to allow the adversary to only run the peers required for the attack, or the adversary can also have a share of *mining power* under control. With mining power available to the adversary, network based attacks that result in disproportional mining rewards are possible (e.g., [NKMS16]).

An additional variable in the adversary model is the **attack duration**. Thanks to cloud services, it may be cheap for an adversary to spawn several thousand Sybil peers for a short period of time.[6] However, carrying out such an attack for a longer period increases the adversary's costs.

Network power, computation power, and attack duration are highly specific to each single adversary. In contrast, there are similarities among all adversaries: Adversaries in permissionless blockchain systems are always able to create new pieces of information (e.g., transactions) and insert them to the network. Furthermore, all adversaries are aware of the source code of most client's software running on the network. Client software is usually published open source in order to establish trust in the implementation and the system. Although it is possible for single parties to use a modified client with a deviating behavior, for the majority of users this is not viable. Hence, adversary models should assume that it is not possible to hide any behavior from the adversary (cf. Kerckhoffs' principle [KL14]).[7]

[6] We show that short-term Sybil events actually occurred in the past on the Bitcoin P2P network in Section 4.3.1.

[7] Of course the client can make use of pseudorandom number generators, the output of which remains unknown to the adversary.

3.2.5 Related Requirements and Adversary Models

We will now discuss additional requirements and adversary models that have been considered for permissionless blockchains or related systems.

Requirements

Scalability has been identified as a separate non-functional requirement of anonymous communication systems [GRPS03]. We will treat scalability as the dependance of the overall *cost of participation* and the *performance* on the number of peers.

A high level of decentralization, i.e., the absence of one or a few authorities that control a large share of the system components (e.g., network peers, mining power), is also a requirement of permissionless blockchains [GKCC14]. From a network layer perspective, a low cost of participation is a requirement for decentralization. Centralization in a system can be modeled by considering adversary models that have, e.g., a large share of mining power or a large number of (Sybil) peers. Therefore, while decentralization is an important requirement, especially on the consensus layer, we do not treat decentralization as a distinct requirement.

Finally, there can be a lack of incentives to actively participate in the P2P network and actually forward transactions and blocks to other peers [BDOZ12, EREL17]. For instance, miners can increase their revenue by withholding transactions with high transaction fees from other miners. Hence, incentive compatibility can also be regarded as a requirement. While we acknowledge the importance of incentive compatibility, the analysis of incentive compatibility requires the common consideration of network, consensus, and application layer aspects using game theoretic approaches and is therefore out of scope in this work.

Adversary Models

Slightly different adversary models than the ones discussed before have been used in the field of rumor source detection (e.g., [FKO+17]). In this setting the only goal of the adversary is the identification of the source of a rumor that is propagated across the network. This equals the attack of anonymity by linking network data (*the rumor source*) to application data (*the rumor*). The adversary is either modeled to obtain a *snapshot* of the propagation of a rumor at a certain point in time or to have a number of *spy nodes* in the network that gather the time of reception of a rumor as well as all possibly attached metadata.

One main difference in the considered adversary models is that the adversary in rumor source detection is usually assumed to know the topology of the network. Although this assumption can be valid in certain scenarios, it does not hold in the case of a peer-to-peer network with dynamically established links.

3.3 Design Space Survey

We will now analyze the design space of the network layer of permissionless blockchains from the perspective of a developer, who maintains the source code of a client software. As the network is assumed to be open, users are still free to use modified

Table 3.1: Design space of the network layer of permissionless blockchains [NH18].

	Main Affected Requirements & Tradeoffs	Further Readings
Peer Discovery	Functional requirement: establish and maintain list of IP addresses of other peers.	
Out-of-Band	DoS Resistance, Anonymity, Topology Hiding	required for bootstrapping, trusted third party
In-Band	[Performance, DoS Resistance] ↔ [Topology Hiding]	Section 6.4, [MLP+15], [HKZG15]
Neighbor Selection	Functional requirement: establish and maintain connections to network peers	
Information Sources	—	IP address based, own observation, reputation
Number of Connections	[Performance, DoS Resistance] ↔ [Cost of Participation]	[AJB00, AB02]
Incoming Connections	[Performance, DoS Resistance] ↔ [Cost of Participation]	[BKP14, Alm17, HKZG15, BDE+13]
Topology Generation	[Performance, Cost of Participation] ↔ [DoS Resistance]	[FOA], remarks: requires trusted information
Stability	[DoS Resistance] ↔ [Topology Hiding]	[CKS+06]
Connection Anomaly Detection	DoS Resistance	[AZV17]
Communication	Functional requirement: information propagation	
Information Sources	—	Message content, metadata [BVFV17], side-channel
Push vs. Announce-and-Request	[Performance] ↔ [Cost of Participation]	[GRKC15]
Flooding vs. Gossip	[Performance, DoS Resistance] ↔ [Cost of Participation]	[PSCVMV15]
Relay Delay	[Performance] ↔ [Anonymity, Topology Hiding]	Section 6.3.3
Message Accumulation	[Performance, Cost of Participation] ↔ [Topology Hiding]	[GNH18]

client software with deviant behavior. Table 3.1 summarizes the affected requirements and further readings for each aspect that will be discussed. The second column shows the requirements that are mainly affected by a certain design aspect. Tradeoffs between two or more requirements are marked with the symbol ↔.

There are two strategies that can be modified in the client that affect the network layer of the system: First, the **attachment strategy** defines which connections to other peers are established. Second, the **communication strategy** defines how clients communicate with their neighbors.

As all public P2P networks require interconnection between its peers, the attachment strategies in different P2P networks are very similar among a wide range of P2P networks including permissionless blockchains. This allows the analysis of the attachment strategy of permissionless blockchains along with the attachment strategy of systems like Tor in order to identify similarities and varying approaches. In contrast, the communication strategy is heavily driven by application layer requirements and makes a comparison between blockchain based systems and systems used for applications such as file sharing not beneficial.

We will now analyze the design space by covering all aspects of the attachment and communication strategies, which either appear in deployed systems, or were proposed or discussed. Although these aspects characterize the network layer of known systems, new systems may include aspects that are not covered in our analysis.

3.3.1 Attachment Strategy

The attachment strategy defines how clients establish connections to remote peers. In order to establish outgoing connections, a client needs to discover the IP addresses of other peers (*peer discovery*). Then, the client has to decide to which peers it establishes connections and how it handles incoming connections (*neighbor selection*).

Peer Discovery

The main goal of peer discovery is to establish and maintain a set of reachable IP addresses of other peers in order to establish connections to them. Peer discovery can be done using out-of-band communication with one or more *seed nodes* that provide IP addresses of reachable peers. If a client is connected to at least one peer on the network, peer discovery can also be performed in-band by requesting IP addresses of reachable peers from neighbors.

Out-of-band peer discovery with one or more seed nodes (e.g., DNS server, IP addresses hard coded to the client) is in general required for bootstrapping.[8] Although a large number of seed nodes can be used, it is still some form of centralization that can affect the requirements DoS resistance, anonymity, and topology hiding.

Discussion: Malicious seed nodes might return only IP addresses of peers under its own control, hence enabling eclipsing attacks on peers that rely solely on information from that seed.[9] Furthermore, malicious seed nodes could try to attack anonymity by linking IP addresses of requesters to application layer data that is later transmitted via the peer returned in the seed's reply. Finally, malicious seed node operators could try to infer topology information by linking IP addresses of requesters to IP addresses returned by the seed node. Adversaries might also try to perform DoS attacks on the seed nodes itself, making it impossible for peers to connect to the network.

In order to prevent these attacks, a large number of seed nodes operated by different parties is required. Clients should request IP addresses from multiple seed nodes operated by different parties to minimize chances that all seed nodes are compromised. Connections should then be established to IP addresses received from different parties. To improve topology hiding, clients should receive a large number of IP addresses but only connect to a small subset. Obviously, all measures cause a bandwidth overhead for clients and seed node operators, thus increasing the cost of participation.

Examples: Tor uses out-of-band peer discovery with a hard coded list of directory authorities along with their public keys[10]. The Bitcoin client[11] and the Monero client have hard coded lists of DNS seeds for bootstrapping.[12] The communication to the DNS seeds is not authenticated. Bitcoin has an operator policy that requests from seed nodes to provide unbiased samples of functioning network peers and to not use

[8] Another possibility is random address probing [DW09], where a clients tries to connect to peers randomly selected from the IP address space. Random address probing comes with significant drawbacks, especially a high bandwidth cost, bad performance, and it is practically infeasible in IPv6 address space.

[9] The IP addresses received using in-band communication from that peer are then also controlled by the adversary.

[10] `https://www.torproject.org/docs/faq#KeyManagement`

[11] When discussing client implementation aspects, we use the term *Bitcoin* to refer to the reference client bitcoind (`https://github.com/bitcoin/bitcoin`).

[12] Bitcoin: `https://github.com/bitcoin/bitcoin/blob/13f53b750dc09cb59192b2aa4ac8e499ee36e1ca/src/chainparams.cpp#L127`, Monero: `https://github.com/monero-project/monero/blob/75563db6e36f044ca0fd08722e2b29a3c950430a/src/p2p/net_node.h#L133`

data gathered from operating the seed node to attack the anonymity of users.[13]

Ethereum has a list of IP addresses along with the public keys of the peers hard coded into the client.[14] Other methods for bootstrapping include IRC channels (e.g., used by Gnutella and formerly used by Bitcoin).

In-band peer discovery can be used once at least one connection to another peer is established. Clients can either request IP addresses from their neighbors or clients can send IP addresses unsolicited to their neighbors. The requirements performance and DoS resistance are affected by the peer discovery, because it determines the speed of establishing connections. The requirement topology hiding is also affected, because in-band peer discovery can be used to infer connections between peers.

Discussion: The announced IP addresses should be reachable with substantial probability, i.e., a successful connection to the announced address should have been made in the past. However, the announced IP addresses should not indicate the connections of a peer. If a client naively announced its neighbor's IP addresses as reachable peers, adversaries could easily infer the connections of that client. However, as peers join and leave the network, it is important to distinguish between peers that are still online and peers that already went offline. Otherwise, old IP addresses keep being announced on the network because they were reachable in the past. A detailed analysis of these tradeoffs is presented in Section 6.4.

Examples: Bitcoin clients can request IP addresses from their neighbors by sending `GETADDR` messages. Furthermore, clients can send IP addresses unsolicited to their neighbors. Peer discovery has been successfully used to infer the network topology of the Bitcoin P2P network [MLP+15]. Ethereum uses a Kademlia-like [MM02] system to discover IP addresses of peers based on their public key.[15] This mechanism was vulnerable to eclipse attacks [MHG18].

Neighbor Selection

The main goal of neighbor selection is the establishment of connections to other peers so that the requirements of the system (e.g., performance, DoS resistance) are satisfied. Generally speaking, a peer can choose to which peers it establishes outgoing connections and from which peers it accepts incoming connections. For this decision, the following questions need to be answered:

- Which information sources are available for the assessment of remote peers?
- How many outbound connections are established? How many inbound connections are accepted?
- Does the selection of neighbors aim at creating a certain network topology?

[13] Bitcoin DNS seed operator policy: `https://github.com/bitcoin/bitcoin/blob/57b34599b2deb179ff1bd97ffeab91ec9f904d85/doc/dnsseed-policy.md`

[14] `https://github.com/ethereum/go-ethereum/blob/79b11121a7e4beef0d0297894289200b9842c36c/params/bootnodes.go`

[15] `https://github.com/ethereum/devp2p/blob/master/rlpx.md#node-discovery`

- Are connections maintained for as long as possible?
- Is the correct functioning of connections monitored?

Information Sources: Once a client has a set of possibly reachable IP addresses obtained through peer discovery, the client can either randomly select peers to connect to from that set, or discriminate remote peers based on information about that peer. Discriminating remote peers can be advantageous in order to (1) prevent adversaries from monopolizing all connections of a client (i.e., improving DoS resistance), or (2) enhance the performance or reduce bandwidth cost by creating certain network topologies.

Discussion: A peer can utilize three types of information about foreign peers: (1) only the IP address and static information associated with it, (2) information based on own observations in the past, (3) information provided by others. The IP address of a peer can be used to identify, e.g., from which IP address ranges, autonomous systems, or geographic regions other peers come from. It is known to the client even before a connection is established. On the other hand, using information based on own observations requires the previous establishment of connections to that peer.[16] Using information provided by other peers on the network (i.e., reputation) is vulnerable to Sybil attacks [Dou02]. Hence, one or more trusted entities providing that information are required.

Examples: IP address information is used in Bitcoin to limit the number of outgoing connections per IP address range in order to improve DoS resistance by preventing the client from establishing too many connections to adversaries with limited IP address resources. Furthermore, taking AS level information into account has been proposed for Bitcoin [AZV17] and Tor [IBW17]. Information based on own observations is used in Bitcoin to blacklist IP addresses that were misbehaving in the past for a certain amount of time. Tor uses trusted directory authorities to provide information such as available bandwidth and availability (i.e., uptime) to clients in order to mitigate Sybil attacks with only a short duration and also to enhance performance by providing information required for load balancing.

Number of Connections: The most basic parameter of the attachment strategy is the number of connections a peer establishes, both, inbound and outbound. Typically, a client establishes a certain number of outgoing connections and may allow up to a certain number of incoming connections. The number of connections is mainly a tradeoff between the cost of participation on the one hand and performance and DoS-Resistance on the other hand.

Discussion: In a flooding network, the bandwidth cost of a peer increases in the number of connections. When naively flooding messages, the increase is linear in the number of connections as every message will be sent over every link once. However, using a two-legged process for flooding (cf. Section 3.3.2), where new messages are

[16] Without any identification mechanism of peers, a peer's identity can only be coupled to the IP address it uses. With dynamic IP addresses, this can lead to false association of information and peers.

first announced and only sent on request, can reduce the overall bandwidth cost to be sublinear in the number of connections. To which peers a client is connected has no significant effect on the total bandwidth cost.

A large number of connections per peer can reduce the propagation delay if the communication strategy makes use of the connections. Many connections can also lead to a reduction in the network diameter, improving propagation speed and enhancing robustness of the network. The required effort for an adversary to isolate a single peer from the network or to partition the whole network increases with the number of connections the clients establish. Many models for the analysis of the effect of the number of connections on performance and security properties of the network have been published (e.g., [AJB00, AB02]).

Examples: The default number of outgoing connections for Bitcoin is 8. It has been proposed to increase this number in order to enhance DoS resistance [HKZG15].

Incoming Connections are less trustworthy than outgoing connections, because even a very limited adversary (i.e., a small number of Sybil peers) can establish a large number of incoming connections to other peers. The maximum number of incoming connections is, as with the total number of connections, a tradeoff between the cost of participation on the one hand and performance and DoS-Resistance on the other hand. However, DoS resistance increases less with more incoming connections than with the same number of outgoing connections.

Discussion: Allowing a large number of incoming connections enables other peers to establish (more trustworthy) outgoing connections. Furthermore, it is the only possibility for clients that are not able or willing to accept incoming connections to establish connections to the network at all. These peers, although not publicly offering their service, help in connecting the network further, and these peers are hard to identify and attack for adversaries without access to core infrastructure. Hence, allowing incoming connections to be made also improves DoS resistance for the peer itself.

The fact that adversaries can easily establish a large number of incoming connections led to the discussion of several options. In order to prevent information eclipsing, it was proposed to not allow incoming connections when accepting zero confirmation payments[17] in Bitcoin [BDE+13]. While certain peers may opt to establish only outgoing connections, the overall network requires peers to accept incoming connections.

A primitive but still possibly effective attack is to use up all incoming connections slot from all peers on the network. This would prevent honest peers from establishing connections. To increase the cost of such attacks it has been proposed to require peers to solve a proof-of-work in order to establish outgoing connections [BKP14, Alm17], which would also increase the cost of participation. Furthermore, it was proposed to limit incoming connections based on IP address information [HKZG15].

Examples: The default number of allowed incoming connections in Bitcoin is 117.

[17] Zero confirmation (0-conf) refers to the behavior of regarding a transactions as accepted as soon as the transaction is received through the Bitcoin network, instead of waiting for the transaction to be included in a block. While this strategy drastically improves performance by reducing the payment delay, it is highly susceptible to double spending attacks.

The number of unreachable peers in the Bitcoin network (i.e., peers behind NATs or peers not allowing incoming connections) has been estimated to up to 155,000 during a 6 hour period in May 2017 [WP17]. With about 5,500 peers reachable via IPv4 during that time, there are almost 30 unreachable peers per reachable peer, according to the estimate. We will discuss this estimate in more detail and compare it to our own measurements in Section 5.3.

Topology Generation: Based on the information sources available to the client, the client can decide to which peers connections are established. This decision affects the resulting network topology, which has a strong effect on the requirements cost of participation, performance, and DoS resistance.

Discussion: Proposals were made to increase performance by favoring the establishing of connections to peers in geographic proximity [FOA]. Geographic proximity can be easily deducted using IP address information and freely available databases, hence, the required information can be easily obtained. However, although this idea might improve the network's performance, it comes at the cost of highly reduced DoS resistance and robustness against random failure. As the number of long distance links such as inter-continental links is reduced, failure of these connections due to random error or attack can cause the network to partition.[18]

It might also be desirable to create network topologies with certain node degree distributions for performance reasons. For instance, scale free networks result in faster information propagation compared to random Erdős–Rényi (ER) graphs for larger networks [CLA16]. However, there are several issues with these approaches: First, in order to create a certain network topology apart from a random graph, clients need information about the node degree of others (e.g., in order to perform some form of preferential attachment). This information can only be provided by the peers itself (peer may lie about their connection count), or by a trusted entity that is able to monitor connections between peers. Despite their better performance, scale free networks were shown to have a lower resistance to targeted attacks, i.e., to an attack where the adversary knows the network topology and attacks and removes specific peers [AJB00]. As ER graphs exhibit a high attack tolerance it is questionable whether it is a good idea at all to change the network topology to not be a random graph.

Examples: One implemented example of a topology generating attachment strategy is the load balancing in Tor. A network wide load balancing is implemented by choosing the probability to establish a circuit through a certain peer according to its relative bandwidth share of the whole network.

Stability: Once connections are established, clients can either try to keep connections for the longest possible duration (i.e., keep the network topology as static as

[18] The distinction between error and attack tolerance of a network implies that an adversary knows the network topology at least partially and is able to selectively attack connections or peers. In order to enhance the DoS resistance of a network it is advantageous to hide the network's topology from an adversary. An adversary that does not know the network topology has to fall back to randomly attack peers of the network, which equals random failure of peers.

possible) or can deliberately disconnect from connected peers after some period and connect to other peers. The stability of the network topology affects DoS resistance and topology hiding.

Discussion: On the one hand, a static network topology makes it harder for adversaries to infiltrate the network with Sybil peers, because connections between honest peers remain as long as possible and only cease to exist due to churn or DoS attacks. On the other hand, a static network topology makes inferring the network topology easier, which enables DoS attacks on central peers of the network and also facilitates deanonymization attacks that rely on knowledge of the network topology (e.g. [FV17]).

Examples: For peers without any incoming connections it was proposed to establish new outgoing connections between publishing two transactions in order to avoid deanonymization [BKP14]. A beneficial effect of continuous changes to the network topology has also been suggested against DoS attacks that are based on hijacking routes to certain autonomous systems [AZV17]. We will revisit the aspect *stability* in the discussion on countermeasures against topology inference in Chapter 6.

Connection Anomaly Detection: In order to avoid being eclipsed, a peer relies on its neighbors to relay relevant information. Clients can monitor their connections and terminate connections to peers that do not behave as expected. This can improve DoS resistance, however, it can also enable DoS attacks.

Discussion: One scenario in eclipsing attacks is that the adversary tries to monopolize all connections of a peer by inserting Sybil peers that stop relaying messages to the peer at some point in time. In order to counter such attacks a client could monitor the rate at which neighbors relay messages. If that rate is significantly lower than the rate observed in the past (or by other neighbors), the client should establish additional connections to avoid being eclipsed. Obviously, the message rate varies over time so that false positives and false negatives can occur. However, as the cost of temporarily establishing more connections is low, clients at risk of being eclipsed should employ this measure to enhance DoS resistance.

In case a client observes a significantly lower message rate from a certain peer, the client could choose to terminate the connection to that peer. On the one hand this would make monitor and Sybil attacks more expensive as the peers of the adversary are then required to relay messages to their neighbors. On the other hand, such a measure would detain users with minimal resources to passively participate in the system.

Examples: Bitcoin monitors connections in the sense that neighbors which send messages not compliant with the protocol are disconnected and blacklisted. This mechanism has been exploited to disconnect Tor exit nodes from the Bitcoin network [BP14], i.e., exploiting an anti-DoS mechanism for a DoS attack. Anomaly detection was also proposed to include metrics such as the round-trip time to neighbors in order to detect other kinds of attacks (e.g., attacks on routing) [AZV17].

Although anomaly detection has been proposed in the past, there is a lack of models that actually can be used for monitoring and detection. Such models should reliably predict message rates and provide configuration options for balancing the affected tradeoffs.

3.3.2 Communication Strategy

Whenever a client creates a new message (e.g., a transaction or block in Bitcoin), that message needs to be disseminated to all other peers on the network. As the creating client is only connected to a small number of peers, it relies on other peers that relay the message to all other peers. The communication strategy of a client decides at runtime for all messages received in the past, which of these messages are relayed to which neighbors at which point in time and how this relaying is implemented.

For this decision, the following questions need to be answered:

- Which information sources are available?
- Are messages pushed or announced and pulled?
- To which neighbor are messages relayed?
- When are messages relayed?
- Is each message treated separately or are messages aggregated?

Information Sources: The client can either treat each message equally, or adapt the communication strategy according to additional information on the message.

Discussion: Strategies might use the content of the message in order to decide relaying times and peers. In contrast to the message content, which serves the actual application, additional metadata can be sent along with the message to provide information to the communication strategy of other peers. Finally, side-channel information can be any information transmitted via the network or collected locally that can be used by the client.

Examples: One example for using the content of the message is the different treatment of blocks and transactions in Bitcoin, where blocks are immediately relayed whereas random delays are applied before relaying transactions. Another use of the message's content can be to treat own messages (e.g., transactions created by the user of the client) differently from relayed messages. That way, a dissemination strategy that enhances anonymity (at the cost of other goals) can be used for initial sending of a message only. Further relaying of the message can then utilize strategies with better performance.

Dandelion [BVFV17] proposes to use a two phase communication strategy and to indicate the current phase via metadata attached to the message. Clients can then apply the currently selected phase of the communication strategy in order to enhance anonymity.

Meta information could also be used as a general means for users to express their personal tradeoff between anonymity and performance for a certain message by setting a suggested mean message delay value. As the metadata is relayed to a client's neighbors, even for the weakest adversary models the adversary learns of this information. Hence, the use of metadata only makes sense if the advantages in a better communication strategy outweigh the disadvantages of an additional information source for the adversary.

Side-channel information is used when Bitcoin SPV clients tell their neighbors which transactions should be relayed by sending a bloom filter. Because bloom filters can be used to attack the anonymity of users, the design of privacy-preserving bloom filters [KTO17] should be considered. Other proposals to use side-channel information include to send Canary status messages upon detection of double spends [OAB+16].

Push vs. Announce and Request: A protocol can be designed to directly relay (push) new messages to neighbors, or it can be designed to use a two-legged process, in which new messages are first announced to neighbors using some form of ID (e.g., the message's hash value). The neighboring client can then check whether the message has already been received and can request the message if necessary (announce-and-request). This aspect is mainly a tradeoff between performance and cost of participation.

Discussion: Push results in a faster propagation than announce-and-request. When pushing a message, only one latency between peers elapses until the message is delivered, with announce-and-request three latencies elapse.

The average bandwidth cost of both approaches can be calculated given the average size of messages (s_m), the size of the ID (s_h), and the probability of request (p_r). The average bandwidth cost per message per connection is s_m for a push strategy, and $s_h + p_r(s_h + s_m)$ for an announce-and-request strategy. The relative bandwidth saving can be calculated as $s_h/s_m(1 + p_r) + p_r$. For instance, assuming $s_h = 32$ Bytes, $s_m = 500$ Bytes and $p_r = 1/16$, which is a very rough estimate of the parameters for Bitcoin transactions, announce-and-request only consumes 13 % of the bandwidth of push.

Only in cases where the additional latency is significant, or where the messages are very small compared to their ID and the probability of request is very high (i.e., in sparsely connected networks), a push protocol is favorable. Even in these systems, peers still need to provide a mechanism for other peers to request messages. For instance, new participants in blockchain systems need to access historic data in order to verify new messages.

Examples: When using announce-and-request, clients need to keep track of requested messages and monitor whether the message is in fact delivered by the remote peer. The timeout mechanism used in Bitcoin for that purpose was vulnerable to a DoS attack in which an adversary announces new blocks to the victim, but does not deliver the blocks [GRKC15]. The block synchronization mechanism in Ethereum was also vulnerable to an attack that delayed the reception of valid blocks at remote peers [WG16]. Proposed countermeasures include using dynamic timeouts, penalizing non-responding peers or requesting the message from multiple peers [GRKC15].

For the transmission of blocks in Bitcoin it has been proposed to announce a new block by sending the block header, which is only 80 Bytes. Furthermore, an extension to the Bitcoin protocol reduces the required bandwidth of block propagation by replacing complete transactions in blocks by short transaction IDs [Cor16]. This is possible because most transactions have been previously received by peers through the transaction propagation process. Transactions that have not been previously received can be requested subsequently from the remote peer.

Flooding vs. Gossip: When a message has been received or created, the client has to decide to which of its neighbors the message is relayed. Messages can be relayed either to all neighbors (flooding), or to a (randomly selected) subset of neighbors. This decision mainly affects the requirements performance, cost of participation, and DoS resistance, but can also affect anonymity.

Discussion: Flooding has a higher bandwidth cost compared to gossiping. However, when using an announce-and-request protocol, the bandwidth cost does not increase linearly in the number of selected neighbors. With an increasing number of selected neighbors the probability that these neighbors already have received the message increases, thus eliminating the need to relay the message itself.

Not relaying messages to certain neighbors has the same effect as not having a connection to these neighbors. Therefore, it has the same negative effect on DoS resistance as a small number of connections in a very dynamic network (i.e., with quickly changing connections). As the subset of neighbors is randomly selected (e.g. for each message), there is a risk that certain messages do not propagate through the whole network. How large that probability is, given a certain network topology and relay probability has been analyzed for decades in the field of epidemic spreading (e.g., [PSCVMV15]).

Examples: The Bitcoin client uses flooding. The first phase of the proposed Dandelion strategy relays messages to one neighbor peer only, which can be seen as a gossip strategy [BVFV17]. It has also been proposed to route messages along certain paths to miners [EREL17].

Relay Delay: Clients can not only decide to which neighbors they relay messages, but also when. This decision mainly affects the requirements performance, anonymity, and topology hiding.

Discussion: The decision when to relay a message can be deterministic or probabilistic. An example for a deterministic approach would be a time-slotted system which rebroadcasts all messages received within a certain time slot at the end of that slot. An example for a probabilistic approach would be a system where upon reception of a message a random delay according to some probability distribution is used to calculate the sending time.

Deterministic decisions might be better from a performance perspective in cases where strict bounds on the information propagation delay are required. For these cases (e.g., real-time requirements), however, the considered systems are in general unsuitable. Indeterminism can be beneficial for topology hiding and anonymity as the attacker has incomplete knowledge of the probabilistic process and can only model the used probability distribution and not the outcome of each single decision. Obviously, deliberately delaying messages reduces the propagation speed, i.e., there is an inherent tradeoff between performance on the one side and topology hiding and anonymity on the other side. A more detailed analysis is presented in Section 6.3.3.

Examples: Bitcoin delays the relaying of messages according to an exponential distribution.

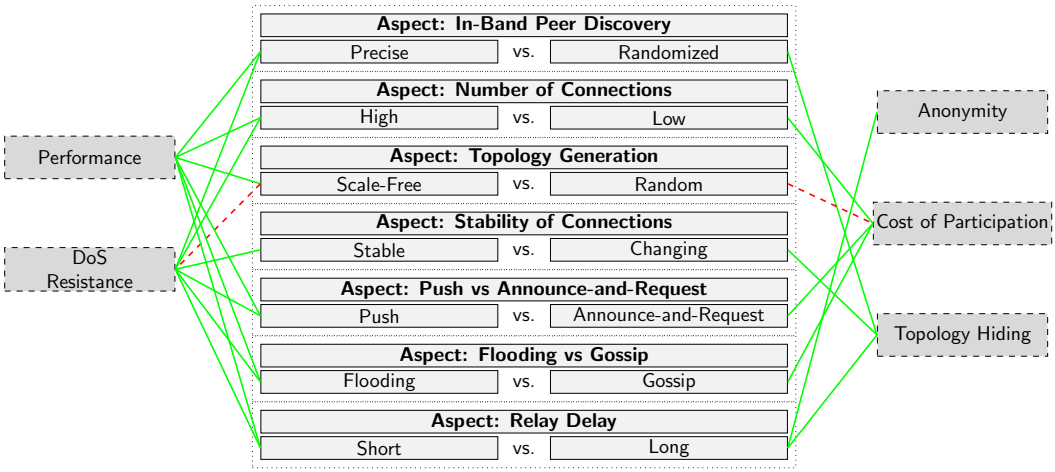

Figure 3.3: Exemplary design choices and their effects (solid green = positive, dashed red = negative) on the fulfillment of requirements. For each shown aspect two possible choices are given (e.g., a short and a long relay delay). Redundant effects (e.g., if a short relay delay has a positive effect on performance, a long relay delay obviously has a negative effect on performance) are omitted for the sake of readability [NH18].

Message Accumulation: So far, we have only considered one message at a time and considered several messages and their relaying times and neighbors to be independent. However, a client could aggregate multiple messages so that they are sent at the same time, which can reduce bandwidth costs by reducing the required control data (e.g., packet header). It can also affect the requirements performance, anonymity, and topology hiding.

Discussion: Message accumulation only makes sense when clients do not rebroadcast messages immediately, because only then the collection of messages is possible. Besides reducing bandwidth costs, it can be more efficient from a software engineering perspective to maintain only one queue per neighbor with one potential sending time instead of maintaining one sending time per message.

Message accumulation can reduce the average relay delay depending on the number of aggregated messages: messages that enter a queue that already contains many messages may be sent early because their time of sending has already been scheduled before they have been received by the client. Message accumulation might improve topology hiding and anonymity because it adds more entropy that is unknown to the attacker as sending times depend on message receptions the attacker does not know about. However, active adversaries can, for example, regularly send new messages to a remote peer and can infer the time the remote peer received a certain message from another peer based on the other messages that are within the same aggregated set. We will present and analyze a topology inference method exploiting message accumulation in Section 6.1.

Examples: Bitcoin uses message accumulation by maintaining one queue of outgoing messages per neighbor. Messages to be relayed to that neighbor are appended to the queue and all transactions within a queue are announced in one `INV` message.

3.3.3 Remarks

Based on the implementation of real-world permissionless blockchains we structured the design space of the network layer into several aspects. Figure 3.3 sketches the effects of exemplary design choices on the fulfillment of requirements. Figure 3.3 can be read in several ways: First, assuming a fixed set of design decisions (e.g., taken from an existing system), one can qualitatively assess the fulfillment of each requirement. Secondly, assuming a fixed importance of each requirement (e.g., based on the envisioned application during the design of the network layer), one can derive design decisions that support the important requirements. Furthermore, the figure shows which design options can be enabled by relaxing certain requirements. For instance, if anonymity is not required in a certain scenario, the relay delay might be shorter, which enhances performance and DoS resistance.

A more general observation can be made regarding the prevailing tradeoffs between requirements: With the exception of the aspect *topology generation*, all sketched design decisions either benefit the requirements performance and DoS resistance *or* the requirements topology hiding, cost of participation, and anonymity. This also implies that there are only few tradeoffs between performance and DoS resistance and between topology hiding, cost of participation, and anonymity. Roughly speaking, achieving performance and DoS resistance requires peers to send more data, whereas achieving anonymity, a low cost of participation, and topology hiding requires peers to send less data.

While only two design choices per aspect are sketched in Figure 3.3, design choices are typically not binary. For instance, the number of connections can be anywhere between one and the total number of reachable peers. Figure 3.4 depicts the effect of the number of connections on the fulfillment of the requirements performance, DoS resistance, and cost of participation. Because a minimum number of connections is required for the network to be connected (i.e., to have a path between any two peers), any design with less than that number becomes unusable (i.e, performance and DoS resistance are not sufficient). While increasing the number of connections substantially enhances performance for a small number of connections, the effect diminishes with a higher number of connections. On the other hand, the cost of participation increases linearly with each new connection. Therefore, there is a certain range in which the inherent tradeoff should be adjusted to satisfy the requirements.

Although we discussed each aspect individually, there are interdependencies between certain aspects. Obvious dependencies are the available information sources and how the information is used (e.g., increasing the number of connections based on anomaly detection, increasing the relay delay based on network statistics). In order to comprehensively assess the fulfillment of a requirement in a certain design, all aspects that affect a requirement (cf. Table 3.1 and Figure 3.3) need to be considered.

Finally, the design of the network layer is, with few exceptions, only limited by the creativity of the designer. Hence, while covering a wide range of aspects, no claim of completeness can be made. Having described the design space and implications of design decisions facilitates the detailed, quantitative analysis of certain aspects as demonstrated in the following chapters.

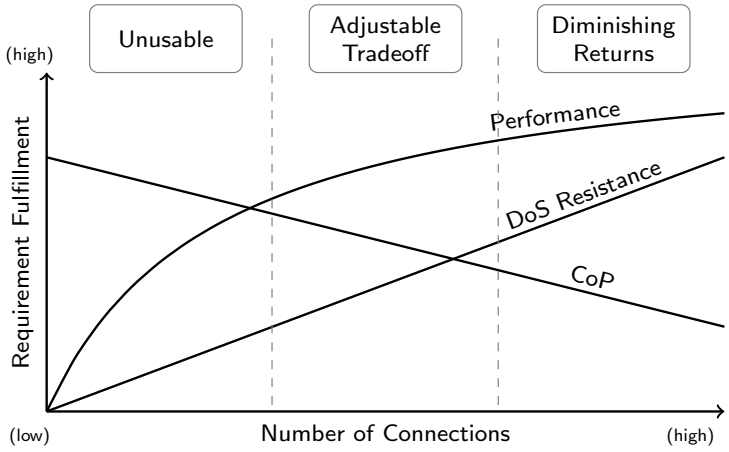

Figure 3.4: Qualitative effect of the the *number of connections* on the requirements performance, DoS resistance and cost of participation (CoP) [NH18].

The remainder of this dissertation builds on the results presented in this chapter in two main ways: First, the identification of the security objectives topology hiding and anonymity in the first part of this chapter motivates the second main research question, which is addressed in Chapter 6 (analyzing topology inference) and in Chapter 7 (analyzing network-based deanonymization). Secondly, the design survey indicates possible countermeasures against analyzed attack methods, and allows their assessment regarding all considered objectives.

4
Network Characterization

The Bitcoin P2P network is real-world phenomenon, which is used as a basis for processing financial transactions in the range of several billion Euros per day.[1] Its structure and behavior is subject to continuous variation and is influenced by numerous internal and external factors. First, the usage of Bitcoin on the application layer (i.e., the creation of transactions by users) varies, thus the information propagated through the network varies. Furthermore, the behavior of those users operating their own Bitcoin peer influences the network, e.g., by creating churn. Many client implementations that are used to connect to the network are actively developed and, therefore, their behavior changes over time. Finally, the underlying network infrastructure, which influences IP routes and network latencies, varies over time.

There are several reasons, why a characterization of the network is required in order to perform research on the network layer of permissionless blockchains. First, in order to perform simulations that produce results, which are applicable to the real-world system, the simulated network should resemble the real-world network. Therefore, a model of the real-world network has to be created. Secondly, even when performing experiments directly in the real-world network, an ongoing characterization of the network is required to assess the reliability of the performed experiments. For instance, in order to avoid experiments being performed during periods with very unusual user or network behavior, an ongoing characterization is required. Furthermore, an ongoing network characterization helps in monitoring the effects of changes to client implementations. For instance, if changes are made, which aim at reducing propagation delays, the comparison of the observed propagation delay at different points in time can help monitoring the success of the implemented modification.

[1] `https://blockchain.info/de/charts/estimated-transaction-volume-usd`

Finally, by characterizing the network, insights on the human activity, which in the end creates the network, can be gained.

In this chapter we will first describe the methodology used to characterize the network. Then, the network is characterized regarding its general, long-term properties. This chapter ends with the analysis of a selection of unusual events and a discussion.

4.1 Methodology

In order to characterize the Bitcoin P2P network, observations from the operation of the network have to be made. Without access to link level data between remote peers of the network (e.g., obtained by packet sniffing at ISPs or internet exchange points), participating in the P2P network with a modified client is a common way to make observations from the network's operation. We will now describe the system architecture and software design used to participate in the network and perform measurements. Furthermore, a description of the collected data and the accessibility and usage of the collected data is given.

4.1.1 Architecture & Software

The main idea of our measurement system is to run a modified client (*monitor peer*), which connects to all reachable remote peers and observes and logs the announcements of transactions and blocks made by other peers. One important principle in the design of our measurement infrastructure is to minimize the effect we have on other peers of the network. Specifically, we aim to reduce resources like bandwidth and processing power required by other peers to serve our measurement infrastructure. We acknowledge that while we participate in the network, we do not provide any service directly to the network and only consume resources.[2] Scaling our approach to a large number of monitor peers to obtain more measurements could be considered a DoS attack on the network. We also chose to make our monitor peer not reachable by other peers, to avoid other (non-reachable) peers to establish connections to our monitor peers, which do not provide any service to them.

The monitor peer has several functional and non-functional requirements. First, it needs to be able to establish and maintain connections to several thousand Bitcoin peers. In order to do that, it also needs to discover IP addresses of remote peers to connect to. Furthermore, it has to monitor and persistently log inbound messages from its neighbors. Finally, it should measure the latency to remote peers, which is a parameter usually required to perform network simulations.

In addition to these functional requirements, there are also non-functional (i.e., performance) requirements. First, the discovery of reachable IP addresses and the successful establishment of connections should be fast. This is especially important, because the number of reachable peers is much smaller than the total number of IP addresses obtained through the peer discovery mechanism. Furthermore, the

[2] This behavior is quite common in public P2P networks, such as filesharing systems, and often referred to as free-riding (e.g., [FPCS06]).

monitor peer has to be capable of processing and storing a large amount of inbound data (in the range of several gigabytes per hour). Finally, inbound messages have to be timestamped precisely, i.e., the logged time of reception of a message should be close to the actual reception of the message *on the wire*.

We rejected the option to write a monitor client implementation from scratch, because of the required effort. Instead, we chose to base our monitor peer implementation on the Bitcoin reference client implementation bitcoind (version 0.10) to ensure compatibility of our monitor peer with other peers. We will now describe the changes made to the client implementation in order to satisfy all requirements listed above.[3] Furthermore, we will describe the used parameters, which are also summarized in Table 4.1.

Connection Limit The maximum number of connections in bitcoind is limited by two independent factors. First, the number of connections is limited to 125 by default by a configured constant value, which we simply removed. Secondly, bitcoind 0.10 uses the POSIX `select` API[4] in order to access its network sockets. However, `select` only supports up to 1024 sockets, i.e., the total number of connections is limited to 1024. While later versions of bitcoind abandoned the `select` API in favor of a completely asynchronous software architecture, we opted for a less invasive change and replaced `select` by the `epoll` system call. `epoll` provides a similar interface as `select`, and allows more than 1024 concurrent connections.

Peer Discovery & Connection Attempts In order to receive IP addresses of other peers to connect to, a client can send `GETADDR` messages to its neighbors, which in turn respond with a list of up to 1000 IP addresses. The interval with which our monitor peer sends out these `GETADDR` messages is a tradeoff between the number of available IP addresses (and, therefore, the number of connections that can be established) and the effort (e.g., bandwidth usage) created at remote peers. In accordance to our principle of minimizing the effect on other peers, we configured our monitor peer to send on average one `GETADDR` message every 2 minutes to one of the connected peers (i.e., at 10,000 connected peers, one peer receives one `GETADDR` message every 2 weeks on average). Because IP addresses are also announced unsolicited by other peers, the configured request frequency turned out to be sufficient to supply enough IP address for the establishment of connections.

Received IP addresses are stored in a local database and used for connection requests. The strategy used for the establishment of connections has to account for several aspects: First, only a very small share of the announced IP addresses are actually reachable (e.g., because peers are located behind NAT routers or peers terminating their client), i.e., a large number of connection attempts has to be made for a small number of successful connection establishments. Secondly, even reachable peers can be temporarily not reachable, because the remote peer already has hit its own

[3] Many aspects of bitcoind have been changed in newer releases of bitcoind. Therefore, the following description only applies to bitcoind version 0.10.

[4] `http://man7.org/linux/man-pages/man2/select.2.html`

Table 4.1: Measurement system parameters.

Parameter	Value	Comment
Connection Limit	∞	Original: 125
`GETADDR` interval	2 minutes	For complete network
Initial connection backoff	10 seconds	After first failed connection attempt
Backoff increment	10 seconds	Per failed connection attempt
Connection retry count	5	
#Connection threads	50	
Failed connection blacklist	6 hours	Prevent connection attempts
`PING` Interval	2 minutes	Per peer

configured limit on the number of connections. Finally, connection attempts should be rate limited in order to avoid being classified as abusive traffic and in order to conform to the principle of least effect on other peers.

We implemented the connection establishment strategy as a parallelized system using a configurable number of threads, which continuously try to establish connections. Rate limitation is implemented as a linear backoff, i.e., the minimum interval between two connection attempts increases with every failed connection attempt by 10 seconds. After a certain number of failed connection attempts, an IP address is removed from the database and blacklisted for a certain duration. The IP address will be added once the blacklist period expired and it is again announced by another peer. The strategy has many configuration parameters listed in Table 4.1, which all balance the speed of connection establishment and the bandwidth usage of our monitor peer and of other peers. The operation of the monitor peer shows that a high degree of parallelization (e.g., 50 threads establishing connections) is beneficial for a fast establishment of connections. Contrary, the minimum interval between connection attempts as well as the maximum number of connection attempts was set quite low (e.g., 10 seconds initial backoff, at most 5 connection attempts).

Message Logging The main message type that is monitored by our monitor peer are `INV` messages, which announce blocks and transactions by their hash value. In order to estimate propagation delays, a precise timestamping of these messages is required. Therefore, it is important to avoid any processing delay between the reception of a message and its timestamping. bitcoind 0.10 separates the processing of messages into two threads: *ThreadSocketHandler* reads incoming data from the sockets into queues for later processing by *ThreadMessageHandler*. We chose to keep this software architecture, as it enables a fast timestamping directly after reading a message from the socket in the *ThreadSocketHandler*. In order to avoid any delays in *ThreadSocketHandler*, we also removed any functionality that requires the acquisition of locks, which might be held by slower threads.

ThreadMessageHandler processes the messages written into queues for each neighbor peer. `INV` messages are directly written to permanent storage in binary format

containing the announced hash value, the announcing IP address, and the timestamp as set by the *ThreadSocketHandler*. `ADDR` messages are used as input to the modified peer discovery mechanism (see above). `PING` messages trigger `PONG` replies according to the protocol specification, in order to prevent connections from being closed. The processing of all other message types has been removed to avoid unnecessary delays and complexity. All further processing of the measured data is done independently from the monitor peer.

Latency Measurement Our monitor peer uses three distinct methods for the measurement of latencies to remote peers. First, the Bitcoin protocol `PING`/`PONG` messages are used to measure the time between sending a `PING` message and receiving a `PONG` message. One advantage of these messages is that all peers respond to these messages. Because the messages are handled by the Bitcoin client itself, this method measures not only the network link latency, but also the delay introduced by the client software such as processing times and waiting times for the acquisition of locks. The second method we use is sending ICMP echo messages. In contrast to the Bitcoin protocol `PING` messages, responses to ICMP messages are sent by the underlying operating system, hence no application delays are introduced.[5] One drawback of ICMP echo messages is that a substantial number of peers do not respond to ICMP echo messages because of firewalls or network stack configuration. Therefore, the third method we employ are TCP SYN pings: A TCP SYN packet is sent to a peer's port running Bitcoin and the time until either a RST or a SYN/ACK packet is received is measured as a round-trip time. This method is commonly used by network scanners (e.g., nmap), and has the advantage that almost all peers respond to TCP SYN messages.

All three methods are implemented within the monitor application as three additional threads, which regularly send all types of pings to remote peers and monitor the reception of the responses. In order to avoid storing the sending time of all sent ping messages, the sending time of the messages are stored in the ICMP payload and in the TCP sequence number, respectively. The measured round-trip times of all methods are stored as-is, i.e., no aggregation or combination is performed during the monitoring process, as such processing can be performed later in the analysis process (cf. Section 4.2.3).

4.1.2 Dataset

We will now describe the acquired dataset. Measurements initially started in July 2015, however, several features of the monitor peer were added subsequently. As of April 2018, the measurements are ongoing. During the observation period there are several short measurement gaps, e.g., because of maintenance events, network outages, and system restarts. Furthermore, there is one larger measurement gap (December 21st 2015 until January 29th 2016) due to a unrecoverable disk system failure. As of April 2018 the total amount data accumulates to around 12 terabytes

[5] Delays can be introduced by the operating system, if the complete system is operating at full capacity.

per monitor peer, i.e., 24 terabytes in total.

Aggregated data including continuously generated statistics have been made available to the research community[6] under a Creative Commons license.[7] Furthermore, anonymized snapshots of the network are provided. The data has been used in several scientific publications, e.g. [MBK+17, GSY18].

Our monitor peers collect data from four different categories. We will now briefly describe the collected raw data.

Churn The collected churn data consists of tuples containing the current timestamp, the IP address of the remote peer, and the *event* to be logged. Logged events are the establishment of a connection, the closing of a connection, and the reception of a version message from a remote peer. Since April 2016, the logged version event also contains the announced client version string. Furthermore, since April 2016 the monitor peers also regularly log the set of all connected peers. Since May 2017 the logged version event additionally contains the announced services and version bits.

Latency As described above, the monitor peers regularly send various types of ping messages to their peers. The collected data consists of tuples containing the current timestamp, the IP address of the remote peer, the type of the ping message, and the measured latency. For every reception of a pong message, such a tuple is created. Failed ping attempts, i.e., sending a ping message without receiving a corresponding pong message, are not logged.

INV The reception of INV messages is logged as tuples consisting of the current timestamp, the IP address of the remote peer, and the announced hash value. While one INV message can announce multiple single hash values, each hash value is logged individually, but with the same timestamp.

ADDR The reception of each ADDR message is logged using the the current timestamp, the IP address of the remote peer, and the list of announced IP addresses including the nTime parameter for each announced address.

4.2 General Network Properties

We will now describe and discuss the long-term results of our measurements since July 2015. A detailed analysis of certain short-term events will be presented in Section 4.3.

4.2.1 Connections

We will first analyze properties of the connections established to other peers, specifically, the number of established connections, the connection duration, and the churn.

[6] https://dsn.tm.kit.edu/bitcoin
[7] https://creativecommons.org/

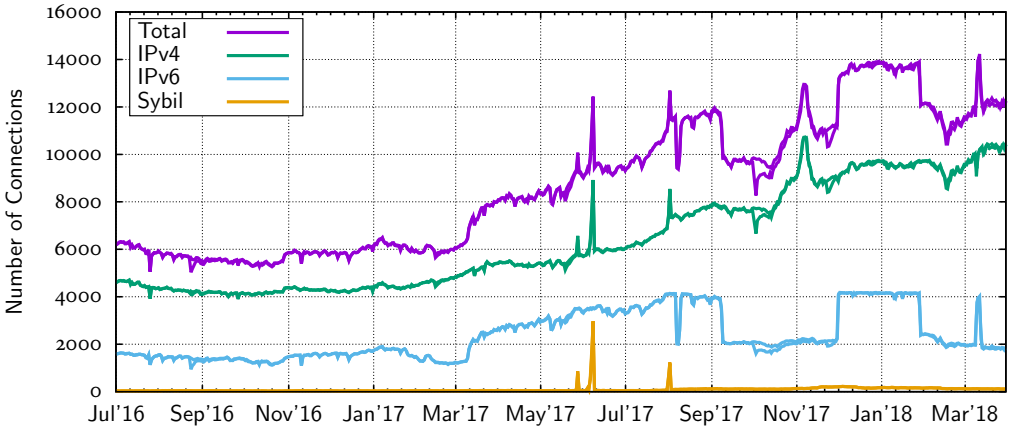

Figure 4.1: Measured number of connections between July 2016 and April 2018 for both monitor peers. The same line colors are used for both monitor peers, because of predominantly overlapping graphs.

Connection Count

The number of peers participating in a P2P network is of interest, because it indicates user adoption of the system and is also important when assessing the possibility of certain types of attacks on the network. Because we can only connect to reachable peers, we do not know the total number of peers on the Bitcoin P2P network. However, the number of reachable peers can be approximated by the number of established connections by our monitor peers.

Figure 4.1 shows the number of connections maintained by our monitor peers between July 2016 and April 2018. For earlier dates, the number of connections can only be unreliably approximated, because of missing data.[8] The plot shows the number of IPv4 and IPv6 connections, and the total number of connections, which is the sum of IPv4 and IPv6 connections. Furthermore, the number of *Sybil* connections is displayed. Sybil connections refer to multiple established connections by the same IP address, i.e., the number of Sybil connections is the difference between the total number of connections and the number of unique IP addresses we are connected with.[9]

Data from both monitors overlap generally overlap closely, with only a few exceptions (e.g., in October 2017). The total number of connections varied between less than 6,000 connections in late 2016 and around 14,000 connections in 2018. The number of IPv4 connections increased at a relatively constant rate during 2017 (with the exception of a few peaks). The number of IPv6 connections increased from less than 2,000 to 4,000 until September 2017, but started to oscillate between 4,000 and 2,000 connections. This oscillation is caused by IPv6 tunneling protocols, as

[8] Until July 2016 only the establishment and closing of connections was logged, which would be only sufficient to derive the number of established connections, if the monitor peer was continuously running.

[9] Sybil here refers to a very simple and easy to detect form of a Sybil attack. Of course, a single person running a large number of peers with different IP addresses would still be considered a Sybil attack, but would not be detected that easily.

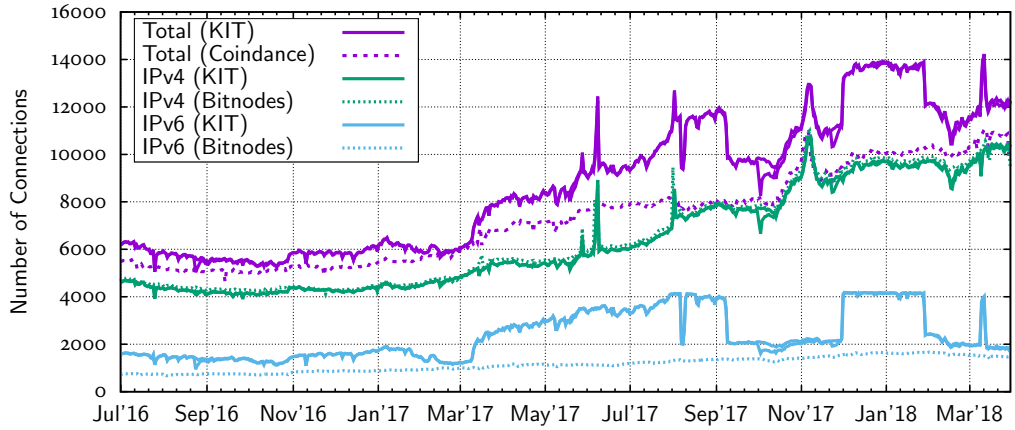

Figure 4.2: Comparison of the number of connections according to our measurements (*KIT*), and the number of connections reported by *Coindance* and *Bitnodes*.

we will discuss later (cf. Figure 4.6).

The number of Sybil peers is generally very low (less than 50 prior to July 2017, less than 200 after August 2017), with the exception of short events in June 2017 and August 2017. We will discuss these events in detail in Section 4.3.1.

In order to assess the validity of our measurement, we compare our results to available results obtained by independent measurements. We are aware of two projects, which perform similar measurements: Bitnodes[10] uses a Python based monitor implementation[11] to connect to peers of the Bitcoin network. It also obtains IP addresses of reachable peers using the in-band peer discovery mechanism and establishes connections them. Coindance[12] also publishes measurements on the total number of peers, however, no differentiation for IPv4 and IPv6 connections are made. Furthermore, no information on the used methodology is provided.

Figure 4.2 shows a comparison of the number of connection as measured by us (*KIT*), Coindance, and Bitnodes. The Bitnodes IPv4 connection count follows very closely the IPv4 connection count measured by us. Even short peaks are congruent in both datasets. However, the Bitnodes IPv6 connection count shows a larger deviation to our measurements, with a consistently smaller number of IPv6 connections reported by Bitnodes (ranging from a minimum difference of 200 in Feb 2017 to a maximum difference of more than 2,000 connections during summer 2017 and January 2018).

The total number of connections reported by Coindance lies between our measured total number of connections and the number of IPv4 connections. Since August 2017, the number reported by Coindance is very close to our number of IPv4 connections, leading to the guess that Coindance only establishes IPv4 connections (or a very small number of IPv6 connections). Until August 2017, Coindance could have either established a similar number of IPv6 connections as we and Bitnodes, or Coindance

[10] `https://bitnodes.earn.com/`
[11] `https://github.com/ayeowch/bitnodes`
[12] `https://coin.dance/nodes`

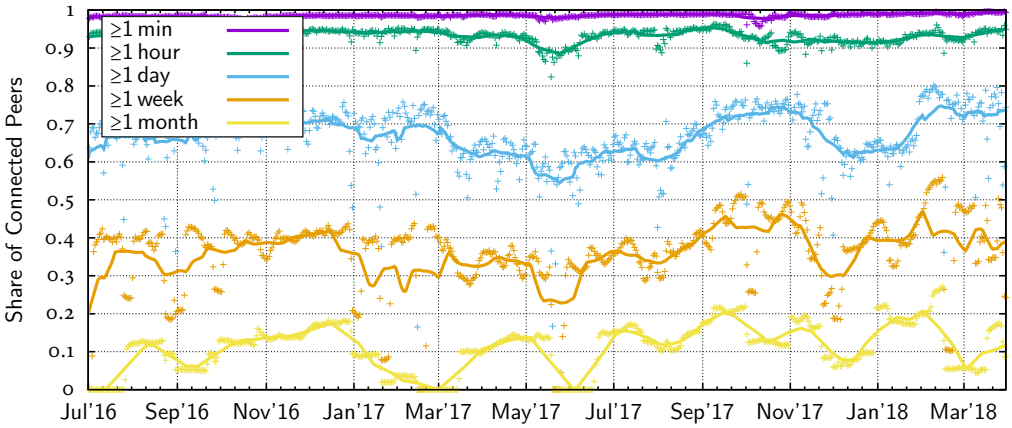

Figure 4.3: Share of connected peers with a connection duration longer than one minute, hour, day, week, or month, respectively. Every data point shows the average per day (48 measurements). The lines show moving averages over the range of one month.

could have established more IPv4 connections than we did. While there are differences between all three measurements, we emphasize that all measurements deviate only to a reasonable extend, all measurements reproduce the same general trends, and all measurements show the same short-term effects.

Churn & Connection Duration

Besides the size of the P2P network, the churn is an important property of P2P networks. Peers entering and leaving the network can be a representation of user behavior [Jün15]. However, we cannot directly measure churn, as connections established between two remote peers are not observable to us. The only events we can observe, are the establishment and closing of connections to or from our monitor peers. These events, however, can be used to gain insights on the churn of the network.

One important observation is the duration for which peers stay connected to our monitor peer. There are two ways to sample this value: First, we can select a random point in time and create a statistic over the connection durations of all peers that are connected to our monitor peer at that point in time. Secondly, we can consider a time interval and create a statistic over the connection durations of all peers that were connected during that period. Both approaches differ in the statistical population, with long connection durations dominating the first population, whereas short connection durations dominate the second population. Furthermore, the composition of the second population depends on the considered time interval. Therefore, we chose to sample the connection duration of peers connected at distinct points in time.

Figure 4.3 shows the share of peers connected for at least a certain duration to our monitor peer since July 2016. Each data point is the average of 48 measurements (i.e., 24 measurements per day for two monitor peers), the lines are moving averages over a period of one month. Consistently, around 99 % of peers are connected for at least one minute. Between 90 % and 95 % of connected peers are connected for at least one hour.

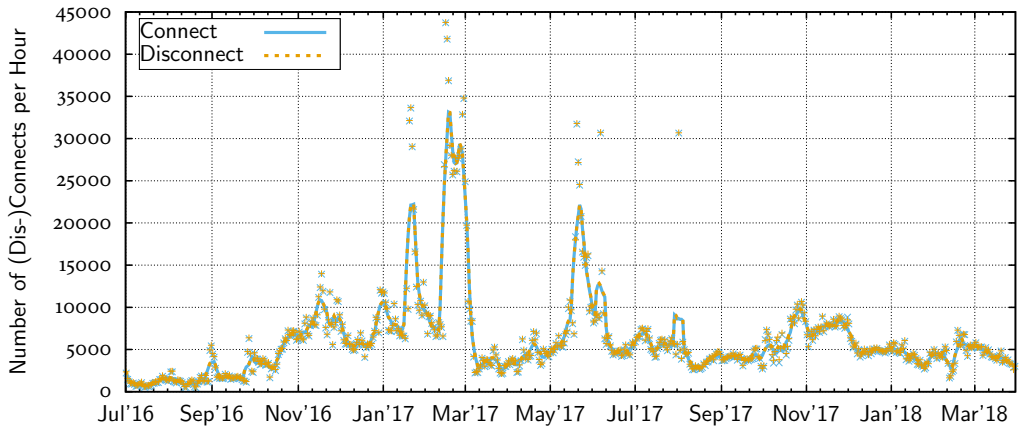

Figure 4.4: Number of connections established and closed, respectively, per hour. Points show data averaged per day. Lines show moving averages over the range of one week.

The average share of peers connected for at least one day varies between 55 % and 75 %, the share of peers connected for at least one week varies between 20 % and 50 % Finally, between 0 % and 20 % of connected peers are connected for more than one month.

The large variance in the share of peers connected for more than one month is caused by the regular loss of connections because of system restarts or network outages. After such an event, it takes one month until peers are connected for one month again. Therefore, the share of peers being connected to the Bitcoin network for more than one month, can be expected to be close to the maximum observed share (i.e., around 20 %).

Compared to measurements of churn in P2P networks primarily used for file sharing (e.g., Kademlia [Jün15]), many peers are connected to the Bitcoin network for a very long duration. This indicates that running a peer is not directly associated with user activity in Bitcoin. Users of file sharing systems usually run their clients on their desktop computers, start the client when downloading a file, and quit the client after finishing the download. In contrast, operators of Bitcoin peers often seem to continuously run their clients on hosted machines (cf. Figure 4.8). Furthermore, while the estimated total number of Kademlia peers oscillated with a frequency of 24 hours, indicating human behavior, the total number of Bitcoin peers does not show such an oscillation.

As previously discussed, the used sample method favors connections with long connection durations. Therefore, Figure 4.4 shows the number of connections established per hour, and the number of connections closed per hour. This gives a complementary view on churn, because it completely ignores the duration of connections, but only focuses on the establishment and loss of connections. Each data point shows the daily average, the lines indicate moving averages (one week). The average number of established and closed connections per hour, respectively, varies between a few hundred and several thousand. Furthermore, the moving averages of connect and disconnect events seem to overlap perfectly during the displayed interval.

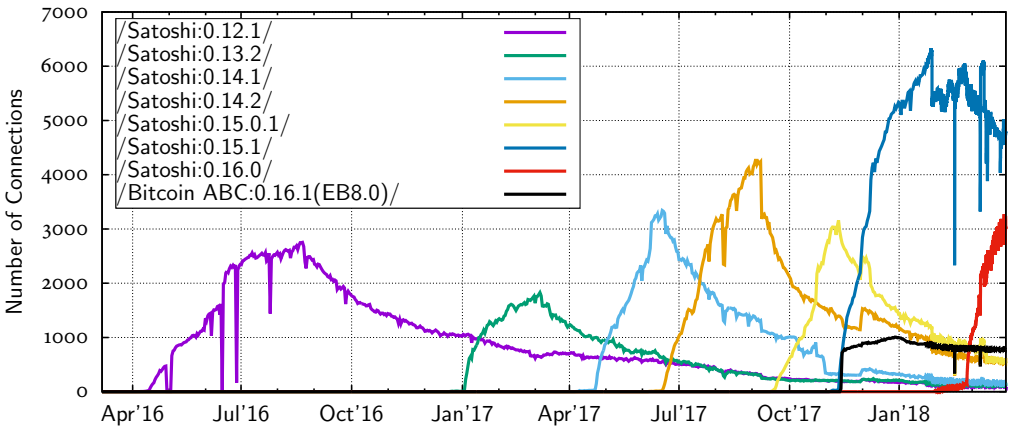

Figure 4.5: Number of peers announcing displayed version string for the top eight version strings between April 2016 and April 2018.

Both observations seem to contradict the previously presented measurements regarding the total number of connections (which increased from 6,000 and 14,000), and the long durations of connections. However, the results are not contradicting but show different aspects of churn: First, while there is an increase in the number of connections, it is too small to be visible as a difference in the number of connects and disconnects at the scale of the shown plot. Secondly, the majority of connections established by our monitor peers are closed within a few seconds. Often, connections to one single remote peer are established and closed on a continuous basis.[13] Although only few such connections are established at any single point in time, they account for the majority of (dis-)connect events.

Client Versions

After establishment of a new connection, both clients send a *version* message to the remote peer to indicate their *version string*, their *protocol version number*, and their *services*. The main goal of the exchange of this information is to ensure compatibility between both clients of a connection. For instance, clients that support the Bitcoin Cash fork and are, therefore, incompatible with clients supporting the Bitcoin main chain, announce a distinguished set of services. Furthermore, certain protocol extensions such as *Bloom filters* [CT15] and *Segregated Witnesses* [LW16] are encoded in the announced services.

Figure 4.5 shows the number of peers announcing a certain client version between April 2016 and April 2018 for the eight most common version strings during that period. Except for one version of the Bitcoin ABC client, which supports the Bitcoin Cash fork, all most common version strings belong to different versions of the Bitcoin reference client implementation *bitcoind*, announced as */Satoshi:xx/* (xx denoting the client version number). Whenever new versions of *bitcoind* are released, the

[13] Remote peers may chose to disconnect from our monitor peer after detecting that we do not provide any service to them.

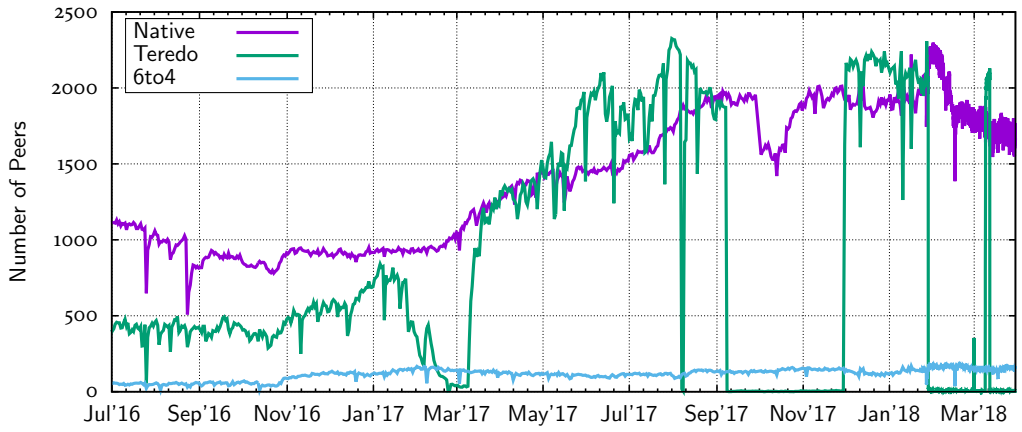

Figure 4.6: Number of IPv6 peers using native IPv6, Teredo, and 6to4 between July 2016 and April 2018.

deployment of these versions at the monitored peers can be observed. Usually, there is an increase in the number of peers running a new version in the first two months after the release of the new client version. This gradual adoption of new client versions is likely caused by peers being operated by a large number of distinct users. Each user chooses their own time to upgrade on a new client version based on factors like importance of the update to the user, available time to actually perform the upgrade, and reluctance to use a new version because of possible bugs. As these factors are highly subjective and vary from user to user, the deployment of a new client version takes the observed time.

Contrary, the number of peers using the Bitcoin Cash client *Bitcoin ABC* increased from 0 to more than 700 within one day. A thorough discussion of the Bitcoin Cash fork will be presented in Section 4.3.1.

4.2.2 IP Properties

So far we have only considered information directly obtained from the establishment and closing of connections to remote peers. Because all connections are established using TCP and peers are addressed using their IP addresses, the characterization of the Bitcoin P2P network can also rely on information that can be associated to remote IP addresses. In the remainder of this subsection we will look at the usage of IPv6 tunneling protocols, and the countries and autonomous networks associated with the IP addresses of remote peers.

IPv6 Tunnel

In order to allow hosts that are connected only via IPv4 to communicate with IPv6 hosts, several tunneling protocols exist, with *Teredo* [Hui06] and *6to4* [CM01] being the most prominent ones. The use of both protocols by a remote host becomes evident, because both protocols use a specified range of dedicated IPv6 addresses. Therefore, based on the IP address of a remote peer, we can determine whether the remote host

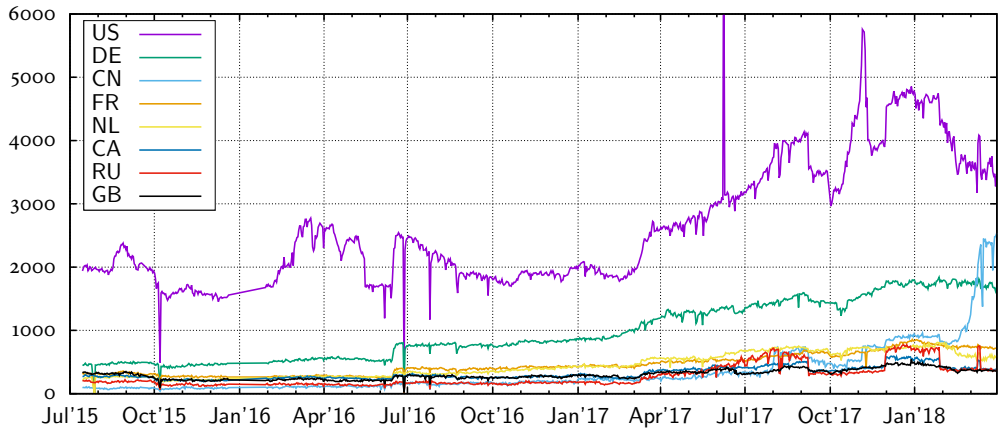

Figure 4.7: Number of peers per country for the eight countries with the most peers.

is using native IPv6 or one of the tunneling protocols for its IPv6 communication.

Figure 4.6 shows the number of IPv6 peers using native IPv6, Teredo, and 6to4, respectively. During the displayed period, the number of native IPv6 peers increases from less than 1,000 in late 2016 to more than 1,500 in 2017 and 2018. The number of 6to4 peers remains at a low level between 50 and 150. The number of Teredo peers shows several abrupt rises and declines during the displayed period. For instance, it increases from less than 100 to more than 1,000 within a few days in March 2017, falls from more than 2,000 peers to almost zero (around 10) several times in 2017 and 2018, and increases back to more than 2,000 peers.

The increases and drops in the number of Teredo peers correspond to the observed changes in IPv6 connections in Figure 4.1. We will discuss possible reasons for this observation in Section 4.3.1.

Countries

During the assignment process of IP addresses, information about the assignee is stored in databases of organizations such as ICANN and RIPE. This information can be retrieved using the *WHOIS protocol* [Dai04], in order to determine the countries, to which the IP addresses were registered. Although this information is not 100 % accurate, it gives a reasonable impression of the geographic distribution of peers among various countries.

Figure 4.7 shows the eight countries, with the most peers during the displayed period from July 2015 until April 2018. Consistently, most peers are located in the US (ranging from around 1,500 to more than 5,000). Between 500 and 1,800 peers were located in Germany during the observed period. The number of peers located in China increased from less than 400 in April 2017 to more than 2,500 in April 2018.

As peers from the US account for roughly one third of all reachable peers, and many peers are located in Western Europe, the geographical and politically distribution of peers can be regarded as somehow centralized, although there are peers from other continents as well.

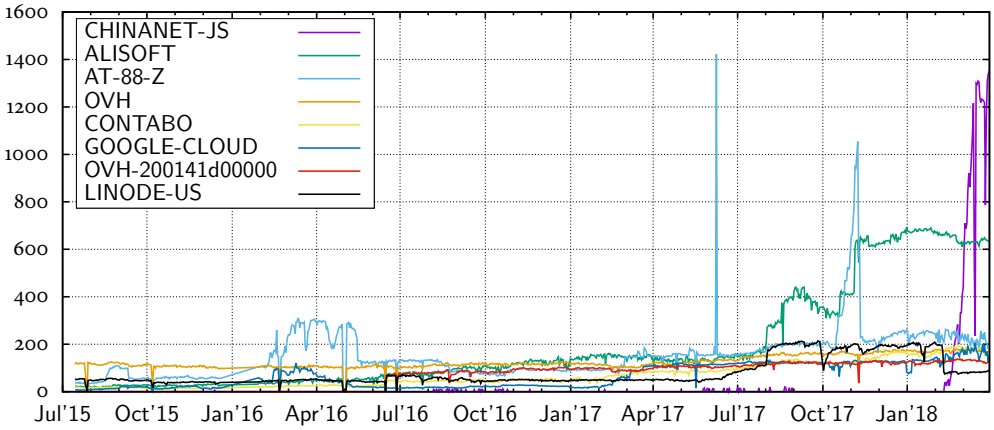

Figure 4.8: Number of peers per AS for the eight AS's with the most peers.

Autonomous Systems

For internet routing purposes, a set of IP addresses under a single technical administration (e.g., under the control of one ISP) is grouped into one *autonomous system* (AS) [HB96]. The AS of an IP address can be resolved using BGP data or using the WHOIS protocol.

Figure 4.8 shows the number of peers per AS for the eight most common AS's. As of April 2018, the two AS's with the most peers both are assigned to Chinese ISPs (Chinanet and Alisoft). The number of peers in the Chinanet AS increased from zero in January 2018 to more than 1,200 within less than two month. All of the Chinanet peers (with the exception of two peers) are announcing the same client version (Satoshi 0.15.1), which leads to the guess that those peers are administrated by one single party. The AS with the third most peers (AT-88-Z) is assigned to the US based company Amazon. The number of peers in the Amazon AS shows some short peaks, e.g., in June and November 2017.

The data shows that a large number of peers is located at cloud and hosting providers, i.e., many peers are run on hosted servers and not at home. For instance, the largest AS from the ISP Verizon only has 72 peers.[14] In general, the data shows a similar centralization as seen on the country level, but on a more fine grained level and focusing on internet structure instead of political structure. It has been shown that this AS-level centralization makes the Bitcoin P2P network vulnerable to routing attacks [AZV17].

4.2.3 Latency

Besides properties of the remote client and their IP addresses, our monitor peers also perform regular estimations of the latencies to their neighbors. As previously described, latencies are measured using Bitcoin protocol ping messages, ICMP echo messages, and TCP SYN messages. We will use the latency measurements in our

[14] Verizon manages a large number of different AS's, which might reduce the number of peers per AS.

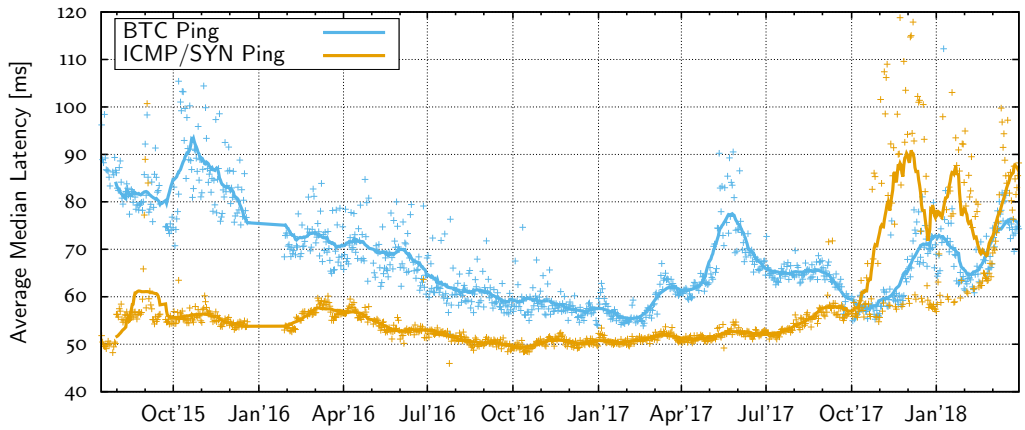

Figure 4.9: Average median measured latency from monitor peers to remote peers.

simulation model in Chapter 5.

Figure 4.9 shows the development of the *average median* latency from July 2015 until April 2018: First, the median of all measured latencies within one hour (typically 30 to 60 single measurements) of each peer is calculated. Then, the average of these medians over all peers is calculated and displayed as one data point in the plot. Finally, the lines show moving averages over one month of data points. Measured latencies using the ICMP echo method and using the TCP SYN method are regarded as one measurement, because of their similar handling by the operating system network stack, and their resulting similar measured latencies.

The measured latency using ICMP/SYN remains relatively constant between 50 ms and 60 ms until October 2017. From October 2017 until April 2018, the observed latencies as well as their variance increase drastically. We will now discuss possible reasons for this increase in the measured latency.

First, the measured latencies seem to originate from a bimodal distribution: While many data points still show a latency of about 60 ms, other data points indicate latencies of more than 100 ms. A closer look into the collected data reveals that starting in September 2017, the measured latencies strongly vary in the course of a day, with high measured latencies during working hours (around 6am to 6pm, weekdays only), and low measured latencies at night. Because both monitor peers show the same behavior and run on different hardware, we can exclude a saturation of the monitor peers itself as the cause of the increased measured latency. Furthermore, we also see the effect of an increased measured latency to peers that are topologically close to our monitor peer, e.g., a peer located at the University of Erlangen, which is connected with KIT directly through the *German National Research and Education Network* (DFN). Hence, we suspect that saturation of the KIT network may cause the increased observed latency.

Figure 4.9 also shows that the latencies measured using Bitcoin protocol pings increased much less since October 2017. In contrast to ICMP and TCP SYN packets, these messages are transmitted through an existing TCP connection. Upon reception

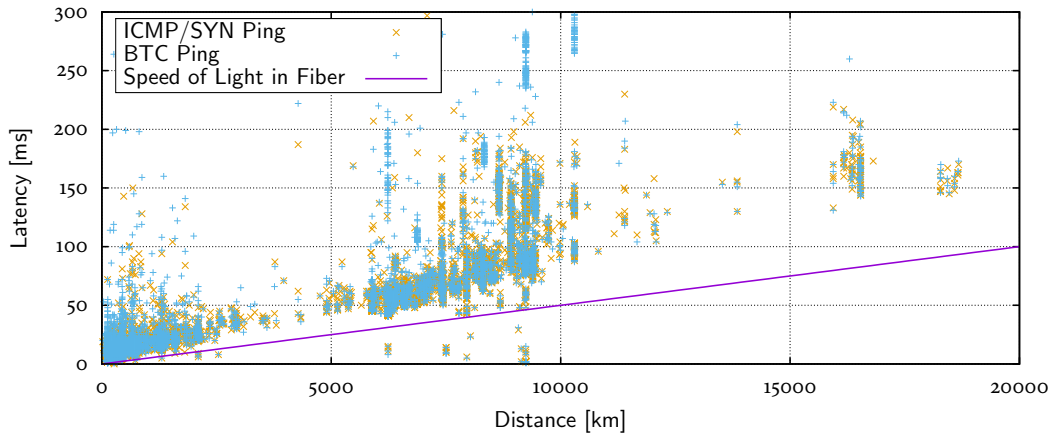

Figure 4.10: Average measured latency per remote peer w.r.t. distance to remote peer. Data from 1st July 2017. Furthermore, the speed of light in fiber (200.000 km/s) is displayed.

of a packet, stateful firewalls usually try to match packets to an existing connection, which can be done very fast. If no such connection exists, the firewall has to evaluate the packet against its ruleset, which can be much slower. Therefore, a saturation of the ruleset evaluation of a stateful firewall at KIT could cause the observed effect. We contacted the technical staff in charge of the network infrastructure at KIT, however, no data was available to confirm our observation.

The latency between two hosts on the internet is the sum of all processing delays of all routers on the path between and the hosts, and the transmission delays of all links between routers. A lower bound on the possible latency between two hosts can be calculated as the quotient of the distance between both hosts and the speed of light in fiber. Figure 4.10 shows the average measured latency for every remote peer depending on its distance to our monitor peers in Karlsruhe, Germany. The location of remote peers is obtained using the Maxmind GeoIP Database.[15] Furthermore, the speed of light in fiber (200.000 km/s) is displayed.

As can be seen, a small number of peers seem to have a latency that is lower that what is physically possible. There are two possible reasons for these errors: First, the distance estimation can be wrong, caused by a wrong mapping between IP address and location. As IP address ranges are frequently transferred between ISPs, such inaccuracies are quite common.[16] Secondly, we cannot exclude the possibility of wrong measurements.

Figure 4.10 also shows that the latency to many peers is only slightly larger than the physically possible minimum latency. This means that the overall latency to these peers is dominated by the limited propagation velocity, and not by processing delays or queuing times. This observation is coherent to our previous observation that many peers are hosted at hosting providers and not connected via consumer ISPs, where a larger effect of the *last mile*, i.e., the communication link to the consumer's

[15] `http://dev.maxmind.com/geoip/`
[16] `https://www.maxmind.com/en/geoip2-city-database-accuracy`

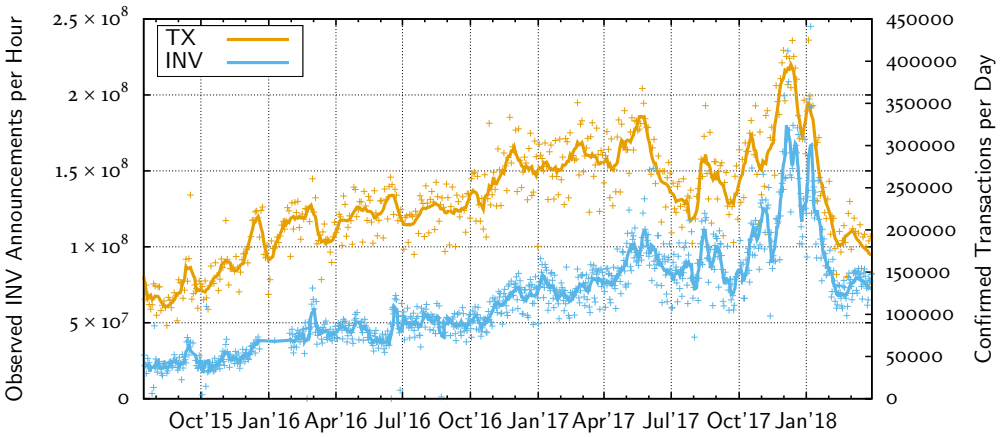

Figure 4.11: Total number of observed INV announcements per hour per monitor peer from July 2015 until April 2018. For comparison, the number of confirmed transactions (i.e., the number of transactions included in the blockchain) is also displayed.

home, could be expected.

4.2.4 Propagation of Transactions and Blocks

So far we have only focused on (derived) properties of remote peers, but not on the behavior of the peers and the network in general. We will now look at the announcement and propagation of information (i.e,, transaction and blocks) over the Bitcoin P2P network.

INV Announcements per Hour

Figure 4.11 shows the total number of received INV announcements per hour per monitor peer since July 2015. One INV announcement means the announcement of a single hash value, multiple of which can be announced within one single INV message. The lines indicate moving averages over the time interval of one week. In addition to the number of received INV messages, Figure 4.11 also shows the number of transactions included in the blockchain per day.

Every valid transaction published on the network should be announced by every peer to all of their neighbors. Therefore, the number of received INV announcements should be equal to the number of published transactions multiplied by the number of connections (cf. Figure 4.1). As can be seen from the plots, this relationship roughly holds.[17] In detail, the relationship does not hold because of several reasons: First, peers may stay passive and not announce transactions at all to our monitor peer. This behavior can be observed for several hundred peers. Secondly, peers may also

[17] For instance, there were roughly 260,000 transactions included in the blockchain in January 2017, i.e., 11,000 transactions per hour. Our monitor peer had around 6,000 connections at that time, hence around 65 million INV announcements per hour can be expected, which is coherent to the number of received INV announcements.

announce a single transaction hash more than one time. This behavior can be observed for around 50 peers. Peers may also come to different decisions whether transactions should be forwarded or not, depending on the fee specified in the transaction, or depending on the used scripts. Furthermore, a transaction may be published on the P2P network, but not included in the blockchain, e.g., because its fee is too low.

During the considered time interval, the number of received INV announcements varied between less than 30 million per hour in 2015, and more than 150 million per hour in late 2017. Because of the limited capacity of Bitcoin blocks, the fee required for transactions to be included in a block increases, if more transactions are published than can be included in blocks in a timely manner. This effect can be seen during the period of in late 2017.[18] It was speculated, whether the large number of transactions was intentionally created, in order to increase the required fee for transactions to be included into the blockchain. Furthermore, a relation to the fork of the Bitcoin Cash blockchain is subject of discussion.[19].

Propagation Delay

As transactions and blocks are flooded through the network, we can indirectly observe this flooding process by observing the announcements made by remote peers to our monitor peers. Upon reception of an INV announcement from a remote peer, we can conclude that this peer has previously received the corresponding transactions or block. However, we cannot precisely estimate the exact time a remote peer has received a specific message, because of delays between the reception of messages and their announcement to other peers. This effect is stronger for transactions, because the announcement of transactions is deliberately delayed by longer periods, whereas blocks are announced immediately after validation.

A common measure, which reflects the propagation delay in a network, is the time between the start of information dissemination and the time until a certain percentage (e.g., 50 %) of peers have received the information. In order to calculate that value, we use the timestamp of reception of INV announcements by remote peers as the assumed time of reception of information by the remote peers. Then, the time between the first reception of an announcement with a specific hash and the reception of announcements containing that hash value by 50 % of peers is measured. Figure 4.12 shows the resulting propagation delays for blocks and transactions (for 50 % and 90 % percentiles) since July 2015.

Since 2015, block propagation delay has decreased from more than six seconds in 2015 until 50 % of peers have announced a block, to less than one second in 2018. Consistently, the 90 % percentile decreased from more than 15 seconds in 2015 to around two seconds in 2018. There are two main reasons for this increase in block propagation speed: First, relay networks, such as FIBRE[20], transmit blocks using forward error correction and UDP communication at transmission rates close to physical limits (i.e., speed of light in fiber). Secondly, extensions to the Bitcoin protocol

[18] https://jochen-hoenicke.de/queue/#1,all

[19] https://medium.com/@deadwing66/cryptoconspiracy-bitcoin-network-might-be-under-expensive-spam-attack-f2fa7baab113

[20] http://bitcoinfibre.org/stats.html

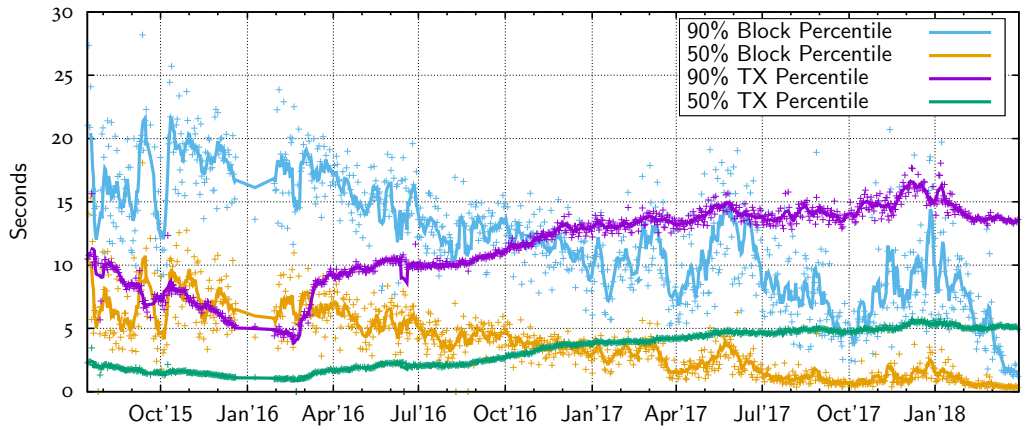

Figure 4.12: Bitcoin propagation delay for block and transaction propagation (50 % and 90 % percentiles).

itself enable a faster transmission of blocks by only sending transactions IDs instead of the complete transaction, which is possible because most transactions have been previously received by peers through the transaction propagation process [Cor16]. Furthermore, performance improvements in client implementations (e.g., using hardware optimization for SHA256 hashing) also decrease the required time to verify new blocks, which reduces overall propagation delay.

On the other hand, transaction propagation delay decreased until February 2016 (to around one second for the 50 % percentile) and increased since then to around 5 second for the 50 % percentile. Transaction propagation delay is mostly caused by deliberately delaying the forwarding of transactions to enhance anonymity and topology hiding. In bitcoind versions prior to 0.12, a change to the software architecture rendered the implemented transaction delay mechanism useless, effectively forwarding transactions immediately. bitcoind 0.12 was released in February 2016 with a modified transaction delay mechanism, which delayed transaction forwarding by a longer duration.[21] These changes are well reflected in the observed transaction propagation delay, especially in the 90 % percentile.

In order to assess the validity of our measurement, we compare our results to available results obtained by independent measurements. We are aware of two projects, which perform similar measurements, namely Bitnodes, which was described in Section 4.2.1, and bitcoinstats[22], which is operated by the authors of [DW13]. In contrast to our measurements, Bitcoinstats connects only to 250 to 1,000 randomly selected peers and monitors INV announcements from these peers.

Figure 4.13 shows a comparison of the 50 % block propagation percentile for all three data sources. In order to enhance readability, single data points were omitted for our measurements, and only the moving average is displayed. No data prior to April 2016 was available for bitnodes. In general a high correspondence between

[21] https://github.com/bitcoin/bitcoin/pull/7125
[22] http://bitcoinstats.com/network/propagation/

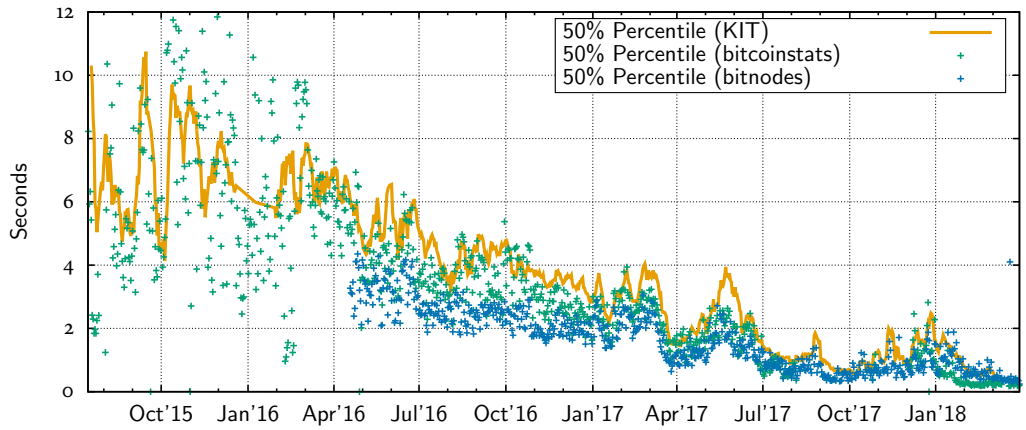

Figure 4.13: Comparison of the 50 % block propagation percentile of our measurements (*KIT*), and the measurements performed by *bitcoinstats* and *bitnodes*.

all three datasets can be seen: The long-term trend as well as short-term variations (e.g., a decrease in March/April 2017 followed by an increase in May 2017) are mostly congruent among all measurements.

However, there are also systematic differences in the collected datasets: Until mid-2017, bitnodes reported the fastest transaction propagation, often around 2 seconds faster than our measurements, which makes a significant difference if the measurements are 3 or 5 seconds, respectively. The transaction propagation delay reported by bitcoinstats is only slightly lower than the one observed by us, and higher than the one reported by bitnodes.

We will now discuss possible reasons for these deviations. All results were obtained using monitor peers located in Europe, hence, no significant difference in latencies to other peers should exist among the three measurements. Furthermore, even if measurements were conducted from other continents, the expected latency difference would be in the range of a few hundred milliseconds (cf. Section 4.2.3), which is not enough to explain the observed differences. As discussed in Section 4.2.1, the number of connections differs between our measurements and the measurement performed by bitnodes. Being connected to a different subset of peers can make a difference in the observed propagation delay, depending on how fast each subset of peers forward transactions. However, the monitor peer operated by bitcoinstats establishes only 250 to 1,000 connections to random peers and the results are still very similar to the other results. Hence, we suspect that the effect of which peers are selected on the propagation delay to be negligible.

A possible explanation for the deviating results could be the method of calculating the percentile values from the collected raw data, i.e., all three projects observe the similar raw data, but might process them slightly differently. Although the calculation of a percentile seems straightforward, there are several parameters, which can affect the output. The main question is which remote peers constitute the statistical population. An obvious answer would be to use the set of connected peers as the statistical

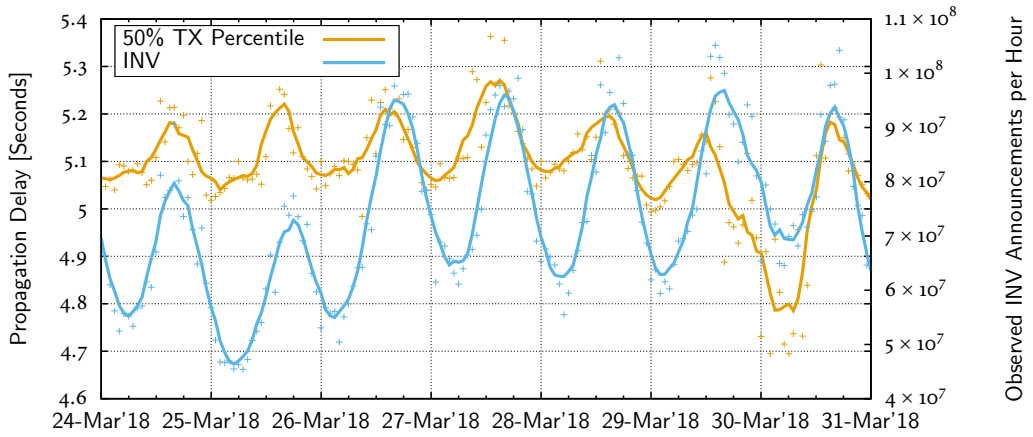

Figure 4.14: Total Number of observed INV announcements per hour and the 50 % transaction propagation percentile between March 24th, 2018, and March 31st, 2018.

population. This approach comes with several problems: The set of connected peers is not constant over time. Furthermore, the set of connected peers includes peers that do not announce a single transaction or block, i.e., that stay completely passive. If more than 10 % of all connections are to such passive peers, we will not receive INV announcements from 10 % of peers, hence, we cannot even calculate a 90 % percentile. This means that the statistical population has to be reduced to the set of peers, which actually announce a new message within a certain interval after the first observation of a new message. The choice of that interval determines the number of outlier peers (i.e., peers which announce a hash many seconds, minutes, or even hours after its first reception) in the statistical population. For instance, choosing a small interval reduces the number of peers with high propagation delays, hence reducing the measured percentiles. We suspect that differences in the choice of parameters cause the deviation between the measurements. We also emphasize that a *propagation percentile* is not a directly measured value, but is derived from measurements, which can be influenced by the measured system, the measurement, and the derivation method.

While Figure 4.12 shows the long-term changes in the propagation delay, there are also short-term changes caused by varying user behavior. Figure 4.14 shows the 50 % transaction propagation percentile for the duration of one week in March 2018. The plot shows that the percentile oscillates between 4.8 seconds and 5.3 seconds with a frequency of 24 hours. Furthermore, Figure 4.14 shows the number of observed INV announcements per hour, which also oscillates between 50 million and 100 million, and correlates to the propagation delay. The minimum number of INV announcements per hour was observed on a Sunday (March 25th).

Because no such oscillation can be seen in the number of peers, we suspect that the variation in propagation delay is actually caused by the variation in network traffic. Please note that while there is an oscillation observable, the amplitude is very small (i.e., below 200 ms for most days). Such a variation can be caused by the transaction delay mechanism in bitcoind: The maximum number of hashes announced in one

single INV message is limited to 35. Therefore, if more than 35 transactions are to be announced to a remote peer, the transactions in excess of 35 are further delayed, increasing the overall propagation delay.

4.3 Case Studies

In the previous section we characterized the Bitcoin P2P network by looking at the long-term changes of network properties. In addition to the general characterization, we also identified several short-term events, which we will now further analyze.

4.3.1 Bitcoin Cash Sybil Peers

The following case study has been previously published as a technical report [Neu18]. As described in Section 4.2.1, there were several short periods, during which a large number of connections from a small number of IP addresses could be observed. One such event took place on August 1st, 2017. This event is of particular interest, because it happened during the fork of the Bitcoin Cash (BCH) Blockchain.[23] Bitcoin Cash is a modification to Bitcoin, which allows block sizes to be larger than 1 MB. Because such blocks are rejected by miners, who follow the traditional consensus rules, the blockchain permanently forks into two independent branches.

Figure 4.15 shows the total number of connections from our monitor peers between July 29th and August 5th, 2017. As in Section 4.2.1, the number of Sybil peers is calculated as the difference between the total number of connections and the number of unique IP addresses we are connected to. While the number of connections is quite constant until August 1st, on August 1st, 2017, the number of Sybil peers increased to up to 5,000. After a period of about 12 hours, the number of Sybil peers decrease to almost zero. The total number of IPv4 connections remains slightly above its previous level (from 6,800 IPv6 connections to 7,400).

Because a relation to the Bitcoin Cash fork seems likely, we analyzed the version strings announced by the Sybil peers. Figure 4.16 shows the number of peers announcing version strings of Bitcoin Cash clients on August 1st. Bitcoin ABC as well as BUCash are clients for the Bitcoin Cash system. Most Sybil peers announced the version string *Bitcoin ABC:0.14.6(EB8.0)*, however, some peers also announced *BUCash:1.1.0(EB12; AD12)* and *Bitcoin ABC:0.14.5(EB8.0)*. The number of peers announcing *Bitcoin ABC:0.14.6(EB8.0)* was below 100 before, and at around 400 after the Sybil period.

The fact that the Sybil peers did not all use the same client version string can be interpreted in multiple ways: First, Sybil peers could be spawned by independent parties, using different client versions. Secondly, different client versions could have been used in order to make the Sybil peers look more *natural*, i.e., caused by normal user behavior. Finally, different client versions could have been used to prevent a single point of failure caused by a potentially existing bug in one client implementation.

[23] `https://www.coindesk.com/bitcoin-cash-what-expect-fork-10000-foot-view/`

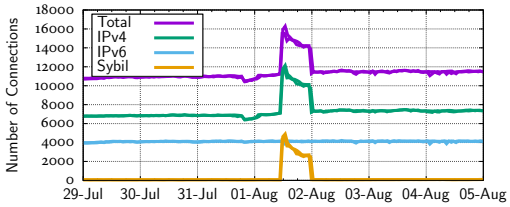

Figure 4.15: Measured number of connections around August 1st, 2017 [Neu18].

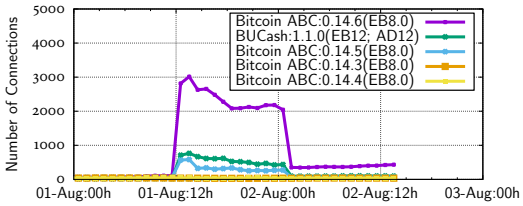

Figure 4.16: Announced client version strings of Sybil peers [Neu18].

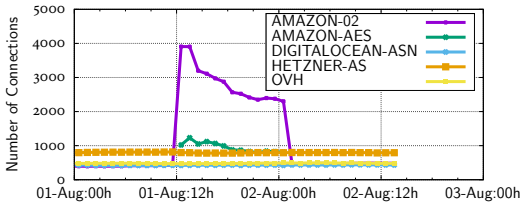

Figure 4.17: Connections per AS, only AS's with most connections shown [Neu18].

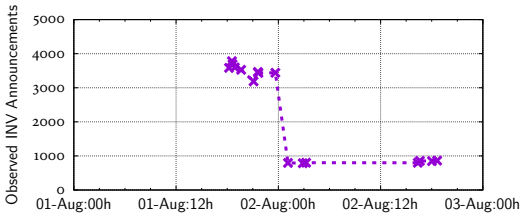

Figure 4.18: Number of INV announcements received for BCH blocks [Neu18].

Figure 4.17 shows the change in the number of peers with IP addresses from the top five autonomous systems during that period. A steep incline in the number of peers from the AS from Amazon (`AMAZON-02` and `AMAZON-AES`) during the considered period can be seen. The total number of connections to peers in Amazon's AS matches the total number of Sybil peers, i.e., all Sybil peers originated from Amazon's AS. This observation suggests that all Sybil peers were spawned by one single party, although it cannot be excluded that several parties independently started a large number of Bitcoin Cash clients on Amazon's hosting services.

Finally, the question arises what the purpose of the large number of Sybil peers was. Sybil peers can be used to attack the anonymity of users, or to perform a DoS attack (e.g., eclipsing) on the network. However, correctly operating Sybil peers can also support the network and defend the network against attacks by increasing the number of peers.

Figure 4.18 shows how many INV messages announcing each Bitcoin Cash block our monitor peers received. The first BCH block was mined on August 1st and announced by 3,583 peers. The following blocks on August 1st were all announced by roughly 3,500 peers, the blocks on August 2nd were announced by roughly 800 peers. No BCH block was mined during a 13 hour period on August 2nd. The number of observed INV messages for each block corresponds well to the total number of Sybil peers. The fact that the Sybil peers actually announced BCH blocks, suggests that the peers should support the BCH network during the critical period of the fork. As the total number of reachable BCH peers is relatively low, a DoS attack on those peers could be easily executed and could have resulted in a partitioned network. The Sybil peers temporarily increased the number of reachable BCH peers by a factor of

about four. Finally, it is also possible that the Sybil peers were spawned by mistake, e.g., by misconfiguration of Amazon cloud instances.

4.3.2 IPv6 Teredo

As discussed in Section 4.2.2, the number of connections to peers using the Teredo IPv6 tunnel mechanism varied abruptly several times, with periods with more than 2,000 Teredo connections, immediately followed by periods with almost zero connections to Teredo IPv6 addresses. This raises two questions: First, what causes the abrupt changes in the number of connections to Teredo hosts? Secondly, are Teredo peers different from the peers that are connected via IPv4 or via native IPv6?

Before addressing both questions, we will now briefly introduce the Teredo tunneling protocol [Hui06]. A host (*Teredo client*) that is connected via IPv4 to the internet and wishes to communicate with an IPv6 host via Teredo, contacts a *Teredo server*, which provides the configuration required for the establishment of the tunnel. After the establishment of the tunnel, traffic relaying is done by *Teredo relays*. Every Teredo client has a unique, routable IPv6 address, which encodes the IPv4 addresses of the Teredo client as well as the Teredo server. Furthermore, because IPv6 packets are encapsulated in IPv4 UDP packets, a traversal of NAT routers is possible.

In order to answer both questions, we analyzed a single snapshot of connections from one monitor peer from January 1st, 2018. At that point in time, a total of 13,885 connections were established, out of which 2,102 connections were made to Teredo IPv6 addresses. The 2,102 connections can be mapped to 2,059 unique IPv4 addresses. This means, that only a small number of peers establish multiple connections via different Teredo tunnels, i.e., we can rule out attempted Sybil attacks using Teredo tunneling as an amplifier for the number of IP addresses available as a cause for the abrupt changes in the number of Teredo connections. Furthermore, out of the 2,059 unique IPv4 addresses, only 239 peers were also connected via native IPv4.

In contrast to the previous case study, the effect does not seem to be caused by a single instance establishing a large number of connections. Furthermore, we contacted KIT's network infrastructure administrator to rule out the possibility that changes to the local network infrastructure caused the effect. Therefore, we suspect that the effect is caused by changes to the Teredo tunneling infrastructure. Interestingly, all 2,102 Teredo connections use only eight different Teredo servers, all of which are in IP ranges assigned to Microsoft.[24] Microsoft announced in 2013 to *sunset* its Teredo services and already performed experiments including temporarily shutting down Microsoft Teredo servers and relays.[25] This suggests that the effect is caused by Microsoft performing changes to its Teredo services.

We will now analyze, whether the Teredo peers differ from the peers connected via IPv4 or via native IPv6. One general limitation of our monitoring method is that only connections to reachable peers can be established. However, there is also a presumably

[24] List of Teredo servers and the number of connections per Teredo server: 157.56.106.184 (504), 157.56.144.215 (444), 157.56.106.189 (392), 157.56.149.60 (288), 94.245.121.251 (270), 157.56.120.207 (146), 94.245.121.253 (57), 65.55.158.118 (1).

[25] `https://ietf.org/proceedings/87/slides/slides-87-v6ops-5.pdf`

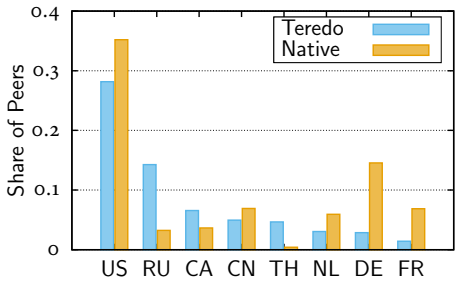

Figure 4.19: Share of peers from displayed country for the set of Teredo peers and the set of natively connected peers.

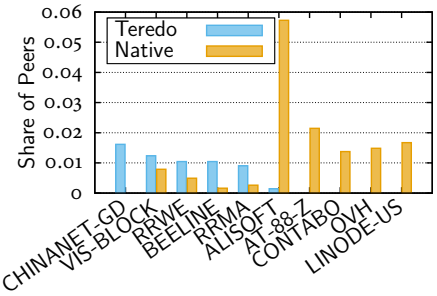

Figure 4.20: Share of peers from displayed ASs for the set of Teredo peers and the set of natively connected peers.

large number of peers that is unreachable, about which we cannot collect information. Because of Teredo's NAT traversal feature, the connections established to Teredo peers are mostly connections to peers that are unreachable via IPv4. As discussed, only 12 % of Teredo peers are also reachable via their IPv4 address. Therefore, analyzing the set of Teredo peers allows a peek into the set of unreachable peers.

Figure 4.19 compares the share of peers from certain counties for the set of Teredo peers and the set of natively connected peers. For both sets of peers, the top five countries are displayed. While some countries have a similar share among both sets of peers (e.g., USA and China), some countries show a vastly different share among Teredo peers and non-Teredo peers: For instance, 14 % of all Teredo peers are from Russia, but only 3 % of all non-Teredo peers are from Russia. Even more extreme, around 5 % of Teredo peers are from Thailand, but only 0.4 % of non-Teredo peers are from Thailand. On the other hand, countries like the Netherlands, Germany, and France are underrepresented in the set of Teredo peers.

One reason for these differences might be different IPv6 adoption rates in various countries and, hence, different strategies used by ISPs to cope with the limited number of available IPv4 addresses. For instance, according to Google[26], the IPv6 adoption rate is high in the US, Germany, and France. Therefore, there is little demand for tunneling mechanisms such as Teredo. Contrary, the IPv6 adoption in Russia is very low at only about 2 %, which might explain the large number of Russian peers using Teredo. The large number of Teredo peers from Thailand, however, cannot be explained with Thailand's IPv6 adoption rate, which is much higher than Russia's IPv6 adoption rate (15 %). Possible reasons for the large number of Teredo peers from Thailand include specifics to the network configuration of Thai ISPs or the operating system configuration of Thai users.

Figure 4.20 compares the share of peers from certain autonomous systems for the set of Teredo peers with the set of natively connected peers. Interestingly, four of the top five AS's for non-Teredo peers (AT-88-Z, Contabo, OVH, Linode) do not have a single Teredo peers. Contrary, the most prominent AS's for Teredo peers are all less common among non-Teredo peers. Furthermore, the share of the most

[26] https://www.google.com/intl/en/ipv6/statistics.html

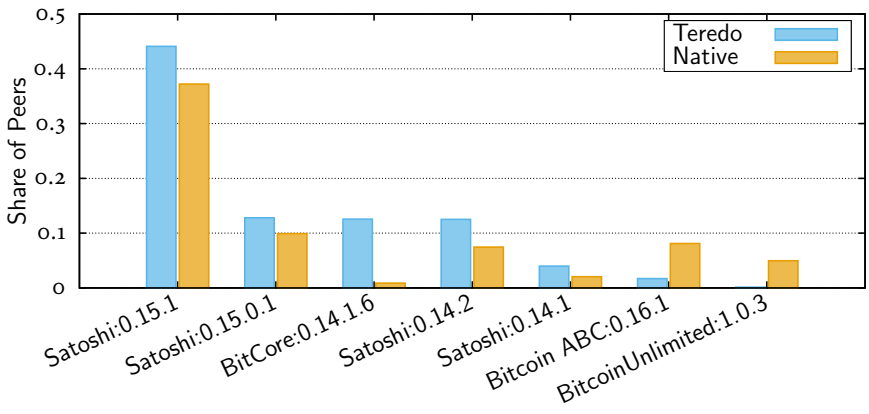

Figure 4.21: Share of peers announcing displayed version string for the set of Teredo peers and the set of natively connected peers.

common AS among Teredo peers (Chinanet-GD), is at 1.6 % much lower than the share of Alisoft peers among native peers (5.7 %), i.e., Teredo peers show a higher degree of AS level decentralization.

The top five autonomous systems among Teredo peers are all operated by ISPs providing consumer internet access (Chinanet, Verizon, Time Warner Internet, Beeline Boradband). Contrary, the top five autonomous systems among native peers are all operated by cloud hosting providers (Alibaba Cloud, Amazon, Contabo, OVH, Linode). Again, this supports the thesis that Teredo peers are run on consumer PCs behind NAT. A similar observation has been made in a previous study focusing on unreachable Bitcoin peers [WP17]: Here, the top 5 AS were all common mobile operators (T-Mobile, Comcast, Verizon, and Rogers).

Figure 4.21 compares the announced version strings among Teredo and native peers. Only minor differences the usage of bitcoind (version string *Satoshi*) can be seen. BitCore is a client for a Bitcoin fork (BTX) with modified block size, block generation interval, and mining algorithm.[27] While less than one percent of all native peers run the BitCore client, more than 12 % of the Teredo peers run the BitCore client. Out of the 370 total peers running BitCore, more than 27 % are from Thailand, which might explain the large share of Teredo peers among BitCore clients.

Contrary, the share of clients for the Bitcoin Cash fork (version strings *Bitcoin ABC* and *BitcoinUnlimited*) is much lower for Teredo peers than for native peers. This is caused by the very large number of Bitcoin Cash peers operated from cloud services: More than 41 % of all reachable Bitcoin Cash peers are operated from the *Alibaba Cloud*. Furthermore, many of these clients seem to be operated by only a small number of parties, because peers simultaneously join and leave the network. For instance, on April 13th, 2018, the number of Bitcoin Cash peers decreased from more than 800 to 375 within three hours.

[27] `https://bitcore.cc`

4.4 Discussion

The presented results indicate that the observation of large P2P networks is not only required for the creation of simulation models (cf. Section 5), but also delivers insights into the network itself. Specifically, our measurement lead to the following statements about the Bitcoin P2P network:

- Performance and anonymity improvements to block and transaction propagation manifest in their observed propagation speed.
- Reachable Bitcoin peers are often run in data centers, unreachable (Teredo) Bitcoin peers tend to be connected via consumer ISPs.
- Bitcoin peers are usually (gradually) upgraded within a few months after the release of a new client version. The upgrade of Bitcoin Cash clients was observed to happen within much shorted time intervals.
- Sybil events actually happened in the past.
- Although Bitcoin is a global network, regional differences can be observed, e.g., in the IPv6 connectivity and in the used client version.

Furthermore, the comparison of our measurement results with other results indicated a reasonable agreement, however, some deviations cannot be explained completely. While the causes for some observed effects can be identified with high confidence, the causes of other effects remain unclear due to a lack of ground truth data, i.e., data collected at remote peers.

We also like to emphasize the potential of measurement errors: The Bitcoin network is a decentralized, changing network, which should be the only system affecting our measurements. However, the measurement systems itself (i.e., monitor hardware, monitor software, local network connectivity) is also subject to change and can affect the measurements. The latency measurements presented in Section 4.2.3 clearly show such an effect of the measurement system, however, it is generally hard to decide whether an effect is caused by the observed network, or by the measurement system. Therefore, a larger number of monitor peers with independent hardware, software, and network connectivity could improve the reliability of the measurements.

5
Simulation Methodology

The security of a system is usually relative to the capabilities of the adversary, which is specified in the adversary model. In Chapter 3 we qualitatively explored the relationship between security objectives, adversary models, design choices, and specific types of attacks. In order to quantitatively analyze this relationship for certain aspects, a method is required that allows an assessment of the effect of certain attacks, without interfering with the deployed system by actually executing the attack.

A variety of methods can be used for the analysis of distributed systems in general and permissionless blockchains in particular (Figure 5.1). Each method has a certain degree of abstraction, requires a certain model of the analyzed system, and can deliver certain kinds of insights. For instance, experiments in the real system come without any abstraction and do not require a model of the system. However, the experiments have to be performed in the real system, the results of the experiments have to be observable, and any potential damage to the deployed system and legal issues have to be avoided. Experiments can lead to results that are highly specific to the analyzed system, however, they tend to lack generality. Furthermore, because of the limited control over the system during the experiments, it is hard to identify causalities, i.e., to explain why a certain result was observed.

Simulation based methods require models for the analyzed system. These models are usually based on measurements of the real network (cf. Chapter 4) and on behavior models derived from protocol specifications (cf. Chapter 2) and client implementations (cf. Chapter 3). However, in most cases at least some assumptions regarding unknown behavior or parameters have to be made because of incomplete knowledge of the system. Because a wide range of model variations and parameter settings can be simulated, the results of simulation based approaches can be generalized to some extent. While realistic simulations can deliver precise results for existing systems, the identification of causalities remains challenging because of the large number of

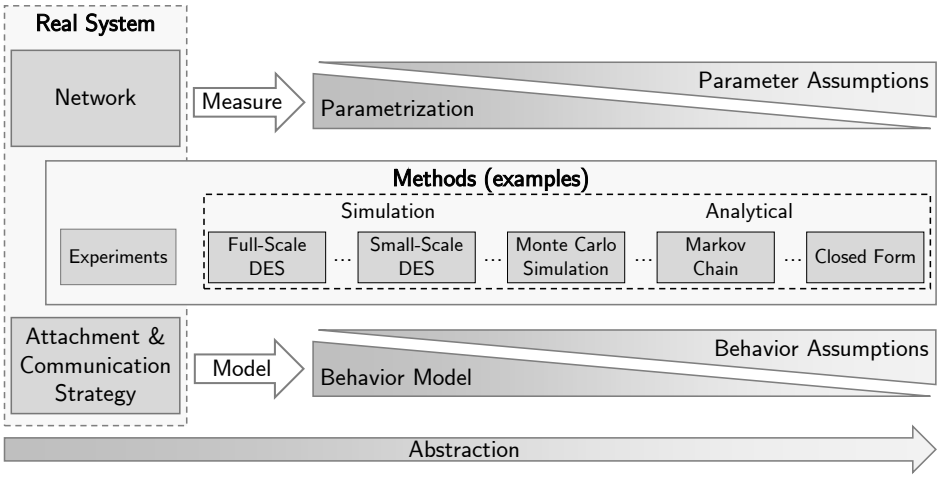

Figure 5.1: Research methodology for the analysis of the network layer of permissionless blockchains [NH18].

overlapping effects. Therefore, it can be beneficial to reduce complexity by abstraction, and use a simplified simulation or analytical approach.

In this dissertation, a wide range of methods is used: The collection of measurement data as described in Chapter 4 itself is an experiment. Furthermore, experiments performed in the Bitcoin network[1] are used for the validation of simulation results. Discrete-event simulations (DES) are used extensively in the analysis of several topology inference methods. Finally, analytical approaches are used to model adversarial knowledge.

This chapter focuses on the used simulation models. Specifically, we present two approaches for modeling the attachment and communication strategy of peers as a discrete-event model. Furthermore, based on the measurements presented in Chapter 4 we parametrize our simulation model, and perform an empirical validation by comparing simulation results to real-world measurements.

5.1 Related Work

We will now briefly cover (simulation) models for permissionless blockchains and related work with focus on the transformation of application code into simulation models. The models are discussed roughly ordered by an increasing level of abstraction.

Testbeds[2] run actual client implementations on a large number of potentially virtualized machines, hence their degree of client behavior abstraction is minimal. However, this lack of abstraction leads to high operational costs. Furthermore, the simulation of network parameters such as latencies between peers require virtualized network infrastructure.

[1] One set of experiments has been performed in the Bitcoin testnet, the other experiments have been performed in the Bitcoin mainnet.

[2] E.g. `http://hackingdistributed.com/2017/02/10/miniature-world/`

Emulation approaches still run the actual client implementations, however, the client implementations interact with an emulated operating system instead of the real operating system. This reduces operational costs and simplifies management of the simulated network. For example, Shadow [JH11] executes the Tor application in a simulation environment and supports Bitcoin using a plugin [MJ15].

Furthermore, a discrete-event simulation model of Bitcoin for the network simulator ns-3 [RH10] has been published [GKW+16]. This model focuses on the transmission of blocks. Finally, analytical models exist, which were already discussed in Section 2.3, e.g., [GKL15].

5.2 Client Behavior Models

In order to serve in the analysis of network layer aspects, the used simulation model has to fulfill two main requirements. First, a precise behavior model of selected parts of the client implementation (e.g., transaction forwarding) is required. Other aspects of the client implementation, such as block validation, might be modeled less precise, hence improving simulation performance. The second requirement is the ability to run large-scale simulations with several thousand peers within a reasonable duration and with available hardware. We will now present two approaches for obtaining a behavior model. Because both requirements are conflicting to a certain degree, both approaches result in simulation models that differ in the degree that each requirement is fulfilled.

5.2.1 Top-Down Model

The starting point of our modeling approach is the source code of the Bitcoin reference client *bitcoind*. The idea is to utilize existing source fragments, and run them within a discrete event simulation. In contrast to emulation approaches, we do not emulate the operating system, but modify the application source code so that it interfaces with the underlying simulation engine. The modeling approach presented in this subsection has been previously published in [NAH15].

We start by briefly describing the software architecture of bitcoind 0.10.0 and the operating principle of discrete-event simulations. The difference between those two create the challenge of our modeling approach, which will be discussed thereafter.

Bitcoind

The bitcoind client is a multithreaded application written in C++. Communication between threads is mainly achieved via message queues and global locks, limiting access to global data structures. Each thread may call blocking functions, such as reading data from a socket. We will now sketch the functional and temporal behavior of the three main threads used for networking in bitcoind:

ThreadOpenConnections/ThreadOpenAddedConnections These two threads try to establish connections to other peers in the network, until the maximum number of outgoing connections (8 by default) is reached. The selection of which hosts to

connect to is based on the received IP addresses from the peer discovery protocol described in Section 3.3. Connection attempts are performed using blocking *connect()* calls. Therefore, the timing behavior of this thread depends on how fast connections can be established, or timeouts on unsuccessful connections occur.

ThreadSocketHandler As described in Section 4.1, this thread reads data from sockets and writes them into message queues for later processing. It also sends data that was previously stored in designated outgoing queues to remote peers. Again, the calls in socket functions are blocking so timing is affected by the blocked duration (e.g., sending a packet on a saturated link). The thread iterates over all connections at most every 10 ms. Our measurements show that blocking calls can increase this interval time to up to around 60 ms.

ThreadMessageHandler Messages that were stored in incoming queues by ThreadSocketHandler are processed by this thread. It cycles every 100 ms through all connection's queues and performs the protocol handling itself (e.g., reacting to messages, checking for timeouts). The timing of this thread can be affected by cryptographic processing delays for instance, during the validation of a block or transaction.

Discrete-event simulation

A discrete-event simulation (DES) is characterized by a system state that is modified by *events* occurring at discrete points in simulated time [Law14]. Each event is associated with an *event handler*, which is a piece of code that executes the logic of the event. A simulator maintains an event queue containing the timestamps and event handlers of future events. Future events can be inserted into the event queue during simulation initiation or as an effect of the execution of event handlers. The execution of the simulation consists of continuously selecting the next event (i.e., the event with the smallest timestamp) from the event queue, and executing its event handler.

In the context of distributed systems, each node of the system usually maintains a local state, which can be modified by events. Activities spanning an interval of simulated time must be modeled using multiple events that represent the start and end of an activity, respectively. For instance, if a message is transmitted from one peer to another peer, the sending of the message is initiated in one *send* event, which schedules a receive event at the receiving peer in the simulated future, accounting for transmission delay.

Simulator Design

We will now first discuss three main requirements of the simulator design. Then the architecture of our simulator is presented.

First, the simulator has to be able to simulate a large number of interacting peers within one simulation. The simulation of each peer can require the simulation of several threads per peer. In order to satisfy both requirements, state that is local to peers and state that is local to threads has to be stored individually per peer and thread, respectively.

Secondly, the real execution of the original application implicitly defines its temporal behavior by the variable durations of computations and waiting times for blocking

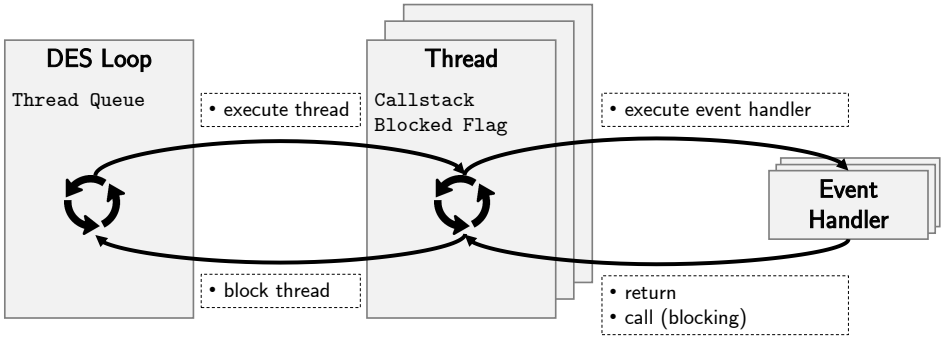

Figure 5.2: Event architecture of the simulator.

function calls, such as network communication or disk access. Because the simulated time is independent of the runtime of the executed event handlers of a DES, delays must be reflected explicitly in the DES model. This requires splitting contiguous code segments into several distinct event handlers, which in turn requires maintaining the local state of threads across events.

Finally, all interaction of the application with the underlying operating system (e.g., network stack, disk access) has to be substituted by calls to corresponding functionality provided by the simulator.

Figure 5.2 depicts the high-level architecture of the resulting simulator. In addition to the main simulator event loop shown on the left-hand side, another event loop dedicated to each thread is used to enable the simulation of threads.

The code executed by each thread has to be transformed into two or more event handlers. Splitting the code into multiple event handlers is required, because most threads invoke blocking function calls (otherwise, the thread would continuously run at full CPU utilization). The event handlers are not directly executed by the DES loop, but instead by a *per-thread* loop, which is used to keep track of the current execution of the thread. Each event handler ends its execution with either returning or by calling a (blocking) function. If an event handler ends without calling a blocking function, the per-thread loop will immediately select and execute the next event handler to be executed according to the stored *callstack*. Only if a blocking function is called, an event for the modeled timestamp of return of the blocking function is enqueued in the main event queue, and control is returned to the main DES loop.

This architecture implies that the main event queue only holds one type of event (*continue thread*). Furthermore, in addition to global state and state held per peer, there is also state held per thread.

Transformation

We will now discuss the transformation steps from application source code into the presented simulator design.

Timing behavior The main challenge arises from the conversion of implicitly defined timing behavior in native applications to explicitly defined timing behavior

in a DES model. We have to split the original code into several event handlers, so that code that is executed at different points in time in the real application is also executed at different points in simulated time. Technically, every single instruction of the native application is executed at a different point in time by the CPU. However, we only care about significant execution delays, such as the call to a blocking operating system function or a very expensive calculation. We define an *execution delay point* (EDP) as such a point in the application source code.

This leads to the following method for splitting application code into event handlers:

1. One event handler is created for each entry point of each thread of the application. The event handler, for now, contains the unmodified code segments required for the execution of the thread.

2. Every EDP within an event handler splits the event handler into two parts – one event handler containing all code segments leading up to the EDP, and one event handler containing all code segments after the EDP. This transformation has to be done for all EDPs.

3. If an EDP is located within a function that is called from an event handler, the event handler also has to be split into two event handlers at the function call. This transformation has to be performed recursively.

Listing 5.1: Example: Timing behavior transformation - source model.

```
1  void threadA() {
2    // do stuff A...
3    sleep(100);
4    // do stuff B...
5    doWork();
6    // do stuff C...
7  }
8
9  void doWork() {
10   // do stuff D...
11   int i ← readBlocking();
12   // do stuff E...
13 }
```

Listing 5.2: Example: Timing behavior transformation - simulation model.

```
1  void threadA1() {
2    // do stuff A...
3  }
4  void threadA2() {
5    // do stuff B...
6  }
7  void threadA3() {
8    // do stuff C...
9  }
10
11 void doWork1() {
12   // do stuff D...
13 }
14 void doWork2() {
15   // do stuff E...
16 }
```

Consider the example sketched in Listings 5.1 and 5.2. Step one of the transformation does not change the source code but keeps both functions as-is. In step 2, any EDP has to be located and the created event handler has to be split at the EDP. The source model contains two EDPs, `sleep(100)` in line 3 and `readBlocking()` in line 11. This means that both functions have to be split into two single event handlers each. Because the function `doWork()` contains an EDP, all event handlers calling that function also have to be split at the point of calling `doWork()`. The resulting event handlers are shown in Listing 5.2.

Splitting the source code into event handlers in the demonstrated way enables a return of the control flow at each EDP to the simulator engine, thus allowing the advancement of simulation time. However, several aspects were not covered by the approach so far. First, the original program flow has to be preserved across the split code fragments. For instance, the event handler `threadA1()` has to ensure that after returning, the thread becomes blocked for 100ms, and continues with the execution of `threadA2()`. This is achieved by adding a code segment to the end of `threadA1()` that sets the blocked flag of its thread to true and adds `threadA2()` to the callstack (cf. Figure 5.2)

Secondly, return values of blocking function calls have to be made available to the calling function. Because every blocking function call splits the original function into two event handlers, the return value of a function can be provided to the second event handler as a parameter. For instance, the source model from Listing 5.1 reads an `int i` in line 11 from the blocking function `readBlocking()`. Therefore, the event handler `doWork2()` in Listing 5.2 requires access to the read variable, and will be provided the variable as a parameter (`doWork2(int i)`).

Furthermore, the behavior of the blocking function has to be implemented in the simulation. For example, a blocking function that reads data from a network socket has to actually read data received by the peer in previous events. Finally, because single functions were split into separate event handlers, local state must be carried across multiple events by the simulator. We will now discuss this aspect.

Program state The programmer of the original application has several options to keep state during the execution of the client. First, global variables can be used, which can be accessed from any function of the program. Secondly, local variables within one function or scope can be used. Furthermore, the program can dynamically allocate memory to store data. Finally, data can be transferred between functions in the form of function arguments and return values.

In order to simulate several client instances, the original source code has to be modified so that (1) each peers accesses its own state and does not interfere with the state of other peers, and (2) state is kept persistent across the execution of multiple events.

The use of *global variables* in the original application has to be changed in the simulation model, because all executed instances would access the same global variable. A straightforward approach is to move all global variables into one data structure, which is instantiated once per simulated peer. Of course, all references to the global variable in the application code must be modified to access the variable within the instance's data structure. Listings 5.3 and 5.4 give an example for such a transformation. The example shows the use of a global variable (`int counter`), which is stored within a `struct globals` in the simulation model. In order to access the variable, a parameter containing the ID of the simulated peer is added to the function signature of the function `inc()`.

Listing 5.3: Example: Global variables - source application.

```
1 int counter;
2
3 void inc() {
4   counter++;
5 }
```

Listing 5.4: Example: Global variables - simulation model.

```
1 struct globals {
2   int counter;
3 };
4 globals[NPEERS];
5
6 void inc(int peerID) {
7   globals[peerID].counter++;
8 }
```

The *dynamic allocation of memory* (e.g., using malloc, the C++ new operator, or C++ containers) in source applications does not impose any problem during a simulation, because the memory of every simulated peer is independent of other peers and the simulator. Therefore, dynamic memory allocation can be used without modification in a simulation model. Only in cases where memory consumption is high, and peers maintain redundant state (e.g., all peers store the complete blockchain, with only minor exceptions), a modification that reduces memory consumption can be appropriate.

As previously discussed, *local variables* have to be made persistent when splitting single functions into multiple event handlers. A similar approach as used for global variables can be used: One data structure is defined for all functions that have to be split into more than one event handler. This data structure contains all variables that are used in the corresponding function. When the first event handler of a function is executed during simulation, an instance of the data structure is created and stored along with the call stack in the thread. Subsequent event handlers modeling later segments of the same function can then access the data structure. After the last event handler of the function has been executed, the data structure is removed from the call stack.

Discussion

We will now discuss advantages and drawbacks of the presented method for transforming a native application's source code into a discrete-event simulation model. The alternatives to the presented method are the use of emulation based simulation (e.g., Shadow [JH11]) on the one hand, and the use of more abstract simulation models created from scratch on the other hand.

Obviously, the main advantage compared to models built from scratch is the lower degree of abstraction, i.e., the model precisely matches the client behavior. However, there are also three main drawbacks: First, the performance of our model is quite limited. While we were able to run simulations consisting of 6,000 peers (i.e., the number of reachable peers during the time of execution), the simulation of one hour of simulated time took about three hours of wall clock time consuming 22 GB of memory. The main reason for the limited performance is the continuous creation of new events by our thread model. These threads may cycle continuously without actually changing the simulation state (e.g., when no new data has arrived).

Secondly, the manual effort for the presented transformation method is extremely high, even though the actual application logic can be used from the original application's source code. Creating a simulation model from scratch, and only porting

the required application logic into the DES model can substantially reduce the required manual effort, especially if only a small part of the original application logic needs to be simulated.

Finally, while the precision that can be achieved by the behavior model might be close to perfect, there are inherent inaccuracies in the parametrization of the simulation (cf. Section 5.3), which can nullify all efforts for a precise simulation model. Therefore, a behavior model that models execution delays probabilistically (e.g., as an additional delay for the transmission of a packet) can achieve a similar overall precision as a behavior model that explicitly models execution delays as described.

Compared to emulation based approaches, the main advantage of the used approach is that the resulting simulation model is a *pure* DES model, i.e., there is no reliance on specific threading libraries or system architectures. This allows such a model to be used in existing simulators and enables the combination of the simulation model with existing models and the use of existing approaches for parallelization using standard HPC infrastructure.

However, compared to emulation approaches that require only minimal changes to the application's source code, the effort for the manual transformation is enormous. Furthermore, the transformation changes the source code in a way that makes it hard to follow the application logic from the DES model, because logically contiguous source code fragments (e.g., single functions) are broken into several independent event handlers.

The discussion shows that only in distinct cases the presented approach can be better suited than a bottom-up behavior model or emulation based approaches. We believe that a reduction of the manual effort is possible using (semi-)automatic source code transformation, e.g., using LLVM and Clang [Lato8]. For instance, automatic source code transformation has been successfully used for benchmarking of parallel applications [SDHD13]. However, because a model of only parts of the client application suffices our requirements and because we require a better runtime performance, we opt to create another behavior model *bottom-up*, i.e., starting from scratch and only modeling the required pieces of application logic.

5.2.2 Bottom-Up Model

We will now present the bottom-up simulation model for Bitcoin. Instead of modifying the source code of the client bitcoind, we start with the protocol specification of the Bitcoin network and model the required event types and handlers.

The first events often required for the simulation of the Bitcoin network are *join* and *leave* events. Although simulations with a static network topology are also possible and useful in some scenarios, modeling churn is required for the analysis of the peer discovery mechanism in Section 6.4. A *join* event models a new peer that enters the network and establishes connections to other peers. Modeling the default behavior of bitcoind, a joining peer establishes outbound connections to 8 randomly selected reachable peers that maintain fewer connections than their configured maximum number of connections. A *leave* event models a peer disconnecting from the network.

Table 5.1: Event types of bottom-up model.

Event Type	Parameters
Join	peer
Leave	peer
(INV	sender, receiver, hashes)
(GETDATA	sender, receiver, hashes)
TX	sender, receiver, transaction
GETADDR	sender, receiver
ADDR	sender, receiver, addrlist

On a *leave* event all connections from and to the leaving peer are removed from the network topology. Furthermore, peers that established on of their 8 outgoing connections to the leaving peer establish a new connection to another, randomly selected peer, so that the total number of outgoing connections remains at the previous level.

In contrast to the previously presented top-down model, the establishment and termination of connections happens atomically in one event. While this does not accurately model the real behavior, it drastically improves simulation performance. Furthermore, the analysis of most aspects does not rely on such a precise model of the connection establishment.

In order to simulate transaction propagation, the flooding process described in Section 2.2.3 has to be modeled. As the protocol contains three different message types (`INV`, `GETDATA`, and `TX`), an obvious way to model the flooding process is be to introduce one event type for the reception of each message type and model the behavior according to the client's behavior upon reception of a corresponding message. However, all three messages can also be modeled as one single event (*tx*) that models the reception of a new transaction. This model represents a push strategy as described in Section 3.3. In order to correctly model behavior and timing of the original announce-and-request flooding protocol, several adoptions are required: First, the time between one peer sending an `INV` message announcing a new transaction to another peer until that peer receives the `TX` message containing the transaction has to be modeled correctly. Therefore, a *tx* event will be scheduled at a simulated time that accounts for three times the latency to the remote peer includes potential client delays (e.g., because of trickling). Secondly, all client behavior that has been separated into handling the reception of different messages has to be merged into one event handler. When a client receives an `INV` announcement, it checks whether it has already received the announced transaction and, if the transaction is new to the client, requests it via a `GETDATA` message. This checking is done in the *tx* event in the simulation model, i.e., the incoming transaction is discarded, if it has already been receive earlier.

The rationale behind modeling all three steps of the message flooding process in one event is the improved performance, reduced complexity, and no requirement for explicitly modeling all three steps. Furthermore, the increase in required bandwidth, which discourages the use of a push flooding protocol, is no issue in a simulation, as all

communication between peers are modeled within local memory. A simulation model that models each step as one event would be required if simulated peers would deviate from the standard message flow and would for example send `GETDATA` messages without the prior reception of an `INV` message.

Finally, two event types representing the reception of a `GETADDR` and an `ADDR` messages are required. These events are only used for the analysis of the peer discovery mechanism in Section 6.4 and are explained in detail in that section. An overview of all used event types is given in Table 5.1.

Although the bottom-up behavior model is much simpler and more abstract than the previously presented top-down model, it enables the realistic simulation of the transaction propagation of Bitcoin and will be used in Chapter 6.

5.3 Network and Client Parametrization

In addition to the behavior of clients, the parametrization of the network and peers has to be modeled. We base our parametrization on our measurements of the Bitcoin P2P network described in Chapter 4. The parametrization of our simulation model contains the aspects churn, delay, network topology, and client behavior parametrization, which will be discussed now.

Churn

A model of churn is required in order to appropriately schedule *join* and *leave* events of peers. If the simulation assumes a static network topology, no churn model is required. We model churn based on our observation of connection establishments and terminations at our monitor nodes (cf. Section 4.2.1). This means, we schedule a join *event* for every successful connection attempt, and schedule a *leave* event for every connection termination.

Several caveats for this approach should be noted. First, there is a time difference between a reachable peer joining the network (i.e., establishing its first connection to any other peer) and the establishment of a connection to one of our monitor peers. This time difference includes the time until the peer's IP address is announced to our monitor node and the time it takes for the establishment of the connection. Secondly, our observations showed that a large number of very short living connections were established by a small number of peers. We suspect that these peers are not continuously joining and leaving the network (e.g., by continuously restarting the client), but rather terminate the connection to our monitor nodes quickly after establishment. Although this introduces an inaccuracy in our parametrization, the implications are mostly a reduced simulation performance, because the effect of a small number of peers on a simulated network consisting of several thousand peers is negligible. Finally, our parametrization does not account for unreachable peers, e.g., peers behind NAT routers. We will address the modeling of unreachable peers in detail later in this section.

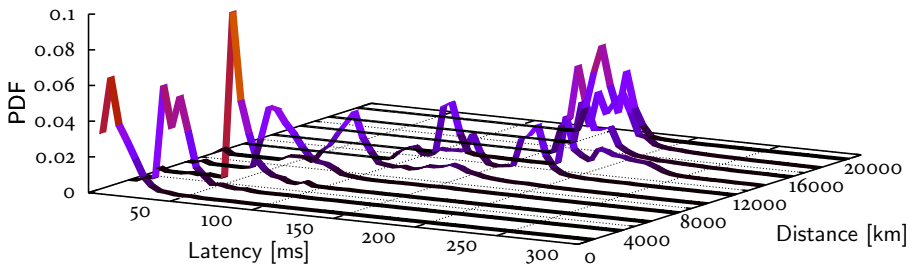

Figure 5.3: Latency distribution broken down by geographical distance between measurement node and foreign peer. Binsize = 2000km [NAH16].

Delay Model

There are two main sources for delays in a distributed system like a permissionless blockchain. First, there are communication delays caused by the latency between two peers. Secondly, the execution of code at the client introduces delays, which can be intentional or unintentional. We will now model both sources of delay. The delay model presented in this subsection has been previously published in [NAH16].

Network Latency While it is easy to measure the latency distribution from one location to other peers, it is much harder to estimate the latency between two foreign peers. In order to perform simulations, however, these peer-to-peer latencies have to be modeled. Approaches like iPlane [MIP+06] provide a latency estimate for pairs of IP addresses by modeling the Internet's routing structure. However, iPlane only provides predictions for a subset of the IP addresses that participated in the Bitcoin P2P network. Additionally, iPlane only estimates a mean latency and not a latency distribution. Therefore, we chose to rely on our own latency measurements to Bitcoin peers, and estimate the latency between peers based on their geographical distance.

We have already shown in Figure 4.10 that the measured latency between our monitor peer and remote peers strongly depends on the geographical distance to the remote peer. Figure 5.3 shows the observed delay distribution from the monitor nodes to all peers broken down by the geographical distance to the remote peer. As the location of peers can be approximated using the Maxmind GeoIP Database[3], every peer can be simulated according to its presumed geographical location. In order to determine the latency for a specific message in the simulation, the distance between sender and receiver of the message is calculated, and a measured latency from the corresponding latency distribution (i.e., the latency distribution for the calculated distance bin) is selected randomly according to the measured probability distribution.

[3] `http://dev.maxmind.com/geoip/`

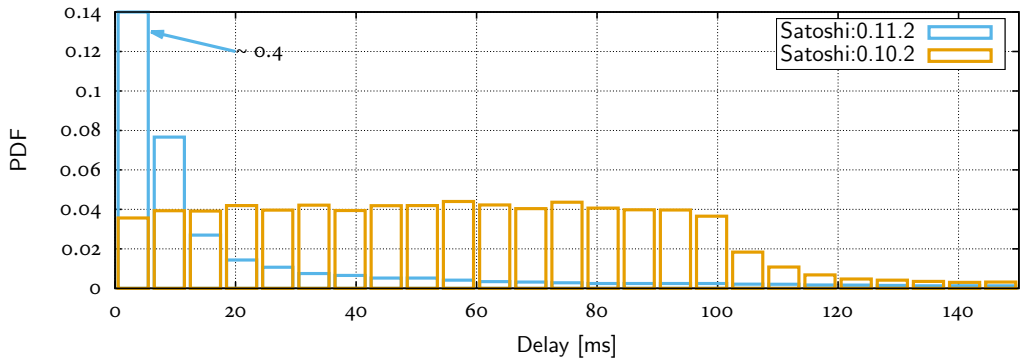

Figure 5.4: Comparison of the unintentional client delay for the Bitcoin reference client versions 0.10.2 and 0.11.2 [NAH16].

We acknowledge that this model has certain limitations: First, the model does not account for latency introduced by temporal behavior such as link saturation. Secondly, while the latency model is well suited for estimating the latency between peers located at well-connected hosting services, it falls short of modeling peers with slow connections to their ISP. However, as most Bitcoin peers are located at hosting services, we expect the resulting error to be limited (cf. Section 4.2.3). Finally, the distance between two peers on the earth's surface may not reflect the routed distance, e.g., through submarine optic fiber cables. A possible approach to improving the used latency model could include a combination of the collected measurement data with other latency estimation approaches such as iPlane.

Client Delay We define the time between the reception of a message and sending of subsequent messages to its peers as the *client delay*. These delays can be unintentional, such as computation delays or delays introduced by blocking behavior of the software architecture. However, the client may also intentionally delay forwarding of messages in order to impede timing analysis or reduce network load. We will first model these unintentional delays empirically and model the intentionally introduced delays in the Bitcoin client analytically.

For the empirical model of the unintentional client delays we leverage the possibility of sending two different types of ping messages to peers: those that are processed by the operating system's network stack (ICMP and SYN) and those that are processed by the application code (Bitcoin protocol `PING`). Although processing by the operating system can be severely delayed, it usually takes only a negligible amount of time in the microsecond range [BRE+15]. By subtracting the averaged network latency from the observed delay for answering a Bitcoin `PING`, we receive an estimate of the application's processing delay.

Figure 5.4 shows the observed unintentional client delay distributions for clients using one of two exemplary deployed client versions. Whereas the older client version (0.10.2) employs a blocking message processing architecture with a fixed 100 ms sleep, the newer version (0.11.2) uses an event driven message processing that minimizes

delays. Our delay estimates clearly reflect these changes; especially the behavior of the older version can be easily modeled with a uniform delay distribution between 0 and 100 ms. Please note that although most delays are less than 100 ms, a substantial number of much longer delays were observed: More than 7.5 % of all delays were longer than one second, 2.6 % of all delays were even longer than 10 seconds.

Two different causes for these very long delays could be identified: On the one hand there are peers that constantly have a very high client delay (i.e., more than 1 second). These peers are probably permanently overloaded. On the other hand there are peers that exhibit small client delays in most cases, but long delays for a few measurements only. These peers are probably only temporarily busy, e.g., they could be verifying a block at this particular point in time. In order to model this dependency, we use our measurements to generate one client delay distribution per peer. Every time a client delay has to be simulated, a client delay measurement is chosen randomly from the peer's client delay distribution.

The Bitcoin reference client implements a trickle mechanism to intentionally delay the forwarding of `INV` messages to neighboring peers. The following description of the trickling mechanism applies to version 0.10.x. We will later discuss changes to current versions. Upon reception of a transaction, the client randomly decides whether to apply trickling on this transaction. Trickling is performed for 75 % of all transactions, 25 % of all transactions are immediately forwarded. Transactions that were not immediately forwarded to all neighboring peers are forwarded to one neighbor at a time during each of the following message processing cycles, which are executed every 100 ms.

This means, the trickling delay probability mass function (PMF) for a peer with c neighbors is given as

$$D_{\text{TRICKLE}}(t) = \begin{cases} 0.25, & \text{if } t = 0 \\ 0.75 \cdot \left(\frac{c-1}{c}\right)^{(r-1)} \cdot \frac{1}{c}, & \text{if } t = r \cdot 100, r \in \mathbb{N} \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

The first case models the immediate forwarding of a transaction in 25 % of all cases. The second case models the trickling case, which is a Bernoulli experiment, where in each round r one neighbor of the peer is selected for message forwarding. With the source code changes in version 0.11.x, the fixed 100 ms interval vanished, effectively increasing the number of processing cycles and, therefore, accelerating message forwarding even with trickling.

As of version 0.12.x, this behavior was changed again so that trickling is performed on a per-neighbor basis with sending times chosen according to an exponential distribution.[4] The mean sending interval (i.e., parameter μ in the exponential function) is set to 5 seconds for incoming connections, and to 2 seconds for outgoing connections. As the composition of client versions in the Bitcoin network changes over time (cf. Figure 4.5), the modeled behavior also has to be adapted to changes made to the client behavior.

[4] https://github.com/bitcoin/bitcoin/commit/5400ef6bcb9d243b2b21697775aa6491115420f3#diff-7ec3c68a81efff79b6ca22ac1f1eabba

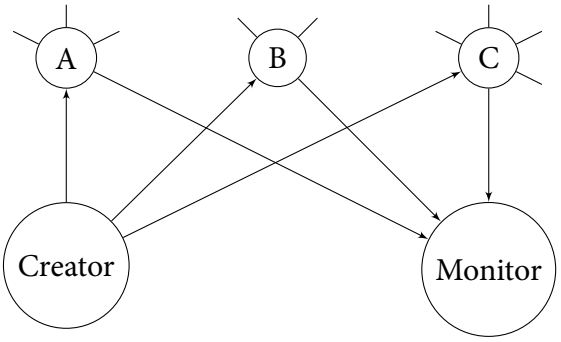

Figure 5.5: Experiment setup for the observation of the 0-hop transaction propagation delay.

Network Topology

The network topology model presented in this section has been previously published in [NAH16].

In order to simulate a P2P network, connections between peers have to be modeled. As the real topology of the Bitcoin network is unknown, the topology has to be statistically modeled by assuming a certain *node degree distribution*, i.e., a PMF which represents the probability of a peer having a certain number of connections. The node degree distribution affects the overall propagation in the network, and is also important for the analysis of attacks on the network. In order to generate network topologies following a given node degree distribution during simulation, we use the *configuration model* as described in [New10].[5] We will now present a method for approximating the node degree distribution of the reachable peers of the Bitcoin P2P network. Later, we will also approximate the number of unreachable peers.

Node Degree Distribution Although most Bitcoin network peers maintain a relatively small number of connections, some peers maintain an extremely large number of connections [MLP+15]. As many similar networks were shown to resemble a scale free network [B+09] and previous data [MLP+15] also supports this assumption, we assume that the node degree distribution follows a power law: $P(k) \sim k^{-\gamma}$. However, as the client is configured to establish eight outbound connections, we set the minimum node degree to be eight as well (i.e., $P(k) = 0, k < 8$).[6] In order to find a suitable parameter γ for the Bitcoin network, we simulate several possible values for γ and compare the simulated transaction propagation speed to the observed transaction propagation speed.

One challenge with this approach is that transaction propagation speed is not only affected by the node degree distribution, but also by the number of unreachable peers, and by peers with deviating behavior. Therefore, we do not take the overall

[5] The configuration model consists of two steps: In the first step, each peer is assigned its number of connections according to the node degree distribution. In the second step, connections between peers are randomly created so that each peer actually has the number of connections assigned in step 1.

[6] Obviously, the probabilities for $P(k), k \geq 8$ have to be normalized accordingly so that $\sum_k P(k) = 1$.

transaction propagation speed into account, but only the *0-hop* propagation speed, i.e., the time it takes for peers directly connected to the creator of a transaction to announce the transaction to their direct neighbors. Figure 5.5 depicts the used experiment: In addition to our monitor node (cf. Chapter 4) we ran an additional peer (*Creator*), which runs a slightly modified version of bitcoind and connects to reachable peers. The only modification made to the Creator client was to remove any delays in the forwarding of transactions, i.e., we disabled trickling. Then, transactions were created at the creator peer and immediately sent to its direct neighbors. The announcements of these transactions by the neighbors of the creator peer were then monitored by the monitor peer.

We chose to focus on the 0-hop transaction propagation delay for two reasons: First, the trickling function (Equation 5.1) depends on the number of connections a peer has. Therefore, the 0-hop transaction propagation delay is strongly affected by the node degree distribution. Secondly, the 0-hop transaction propagation is not affected by the behavior of other peers (except for other peers increasing the number of connections). This makes it possible to ignore unreachable peers for now, which is important, because it reduces the number of parameters that have to be concurrently considered.

The bottom part of Figure 5.6 shows the measured propagation delay distribution between the creation time of a transaction and the reception of the corresponding `INV` messages by our monitor node from the neighbors of the creator of the transaction (*0-Hop*). Additionally, the simulated delay distribution for the same number of reachable peers and a node degree distribution following the adapted power law parametrized with $\gamma = -2.3$ is shown. Various values for the parameter γ were simulated with $\gamma = -2.3$ resulting in the smallest deviation between measurements and simulation, i.e., showing the least square error sum between the two distributions. Therefore, we chose $\gamma = -2.3$ for the node degree distribution of the reachable nodes of our simulation model.

Unreachable Peers With the node degree distribution of the reachable peers given, the number of unreachable nodes has to be approximated in order to simulate the network. Without an estimation of the number of unreachable peers it is not possible to determine how many connections any reachable peer has to other reachable peers, and how many connections it has to unreachable peers.

In the Bitcoin network there are at least two known *classes* of unreachable peers: standard clients that maintain a low number of connections (i.e., eight) and peers that are specifically used to maintain a high number of connections and perform fast message forwarding. The first class of peers could represent a standard user behind a NAT, whereas the second class of peers could be run by an exchange service, for example. We added peers of these two classes to our model and, again, varied the parameters (i.e., the number of peers per class and the number of connections held by the second class of peers) and compared the simulated transaction propagation delay distribution to the measured one. In contrast to the first measurements, that were restricted to direct neighbors of the generating peer, the transaction propagation delay now covers propagation through all reachable peers. Although additional peers

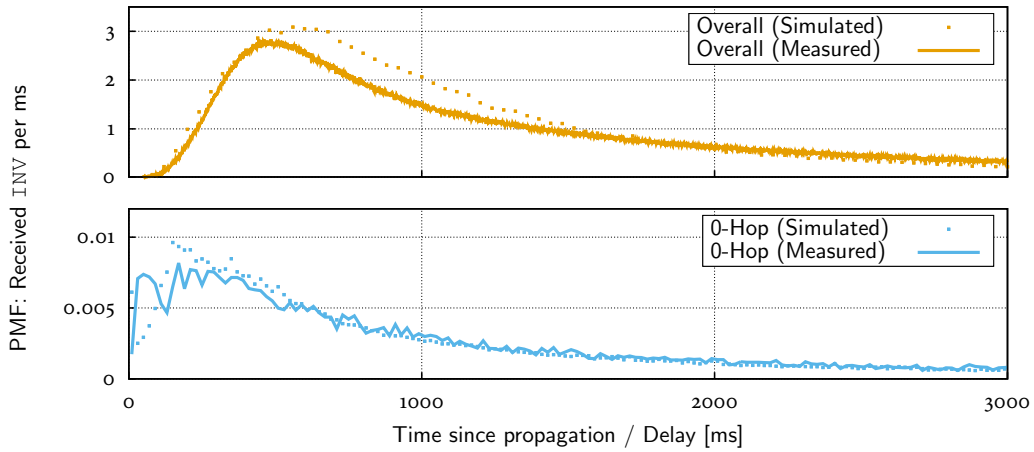

Figure 5.6: Comparison between measured and simulated `INV` propagation delay as histogram data; limited to direct neighbors of originating peer (bottom) and for the complete network (top). Both networks parametrized with $\gamma = -2.3$ [NAH16].

and connections are added to the model, the node degree distribution of the reachable peers does not change, as connections to additional peers substitute already existing connections in the simulation.

The upper part of Figure 5.6 shows the measured transaction propagation delay distribution of all reachable peers compared to the simulated one with 16,000 unreachable standard peers and 70 unreachable peers with 200 connections each. Again, we simulated a broad range of values and the parametrization shown resulted in the smallest square error sum between measurements and simulation. Although this specific parametrization represents the transaction propagation in the real network well, one cannot draw the conclusion that the real network consists exactly of these types of peers in these quantities. It is possible that there are numerous other parametrizations (including additional peer classes or anomalous behavior) that also lead to the same transaction propagation delay distribution.

In a recent study the number of unreachable clients was estimated to be „at least 155,000 at any given 6-hours interval“ [WP17]. While this sounds far off of our estimation, the authors do not state the number of unreachable peers that are *concurrently* connected to the Bitcoin network. However, the authors state that 93.9 % of connections were maintained for a duration of less than 60 seconds. Assuming that these 145,545 connections were uniformly distributed across their 6-hour interval results in an average of less than 7 concurrent connections from these peers. The paper does not state the distribution of connection durations above 100 seconds, however, assuming these peers are connected for the entire 6-hours interval results in an additional 9,455 connections from unreachable peers, which is in rough correspondence to our estimation.

We emphasize that Figure 5.6 also serves as an empirical validation of our simulation model. Overall, the deviation between simulated transaction propagation and observed transaction propagation is reasonably low. We suspect that the re-

maining deviation in the overall propagation delay is caused by anomalous clients that were not modeled here.

5.4 Discussion

In this chapter we presented a simulation model for the Bitcoin P2P network, which will be used in Chapter 6. Two methods for developing behavior models were discussed: While the top-down approach enables high-precision simulations, it turned out to be too costly, from a development as well as operational perspective. Contrary, the presented bottom-up simulation model deliberately accepts a lower precision at lower overall costs. The parametrization based on our real-world measurements enables the simulation of the Bitcoin network with a reasonable precision.

We will now discuss lessons learned and possible future work regarding the behavior model and parametrization of the simulation model. Although the top-down behavior model turned out to be inappropriate for our purpose, the transformation method presented showed the steps required to transform a client application into a DES model. Making these steps explicit illustrates the differences between native application code and DES model code. One measure to reduce these differences is using an event-driven architecture in the native application, as used in newer versions of bitcoind. Using (semi-)automatic source code transformation, it might be possible to further reduce the required effort for the transformation of application code into a DES model.

Although the prerequisites for the parametrization of our simulation model seem to be very promising (e.g., comprehensive monitoring of public network, public source code), there is still an *inherent ground-truth problem* associated with modeling the network layer of permissionless blockchains. For instance, parameters such as the number of unreachable peers or the network topology can only be approximated based on observations. We emphasize that while a lack of knowledge of such parameters impedes research, hiding such parameters also improves the security of the system against several attacks (cf. Chapter 3). Despite the existing ground-truth problem, we provided a parametrization of our simulation model based on approximations derived from our real-world measurements. Furthermore, we validated our model and its parametrization by comparing the simulated information propagation to the observed one (cf. Figure 5.6). Overall, the ground-truth problem stresses the *importance of topology inference* as a basis for research, and not only as a intermediate goal for adversaries.

While the behavior of reachable peers of the Bitcoin network is well analyzed, more work on unreachable peers similar to [WP17] is required. We acknowledge the difficulty of this kind of research without affecting unreachable peers on the network, especially as this kind of research usually requires operating a large number of reachable peers. Knowing more about unreachable peers could enable a better parametrization of our simulation model.

Finally, the source code of most commonly used client implementation is publicly available, which makes it possible to create behavior models for these clients. However, even a very small number of peers with anomalous behavior can significantly affect

the overall network. Therefore, the identification and modeling of such anomalous behavior is required (cf. Chapter 7).

6
Topology Inference

The topology of the P2P network of permissionless blockchains is an important aspect in ensuring anonymity of users [FV17] and in ensuring robustness against denial of service attacks [HKZG15], double spending attacks [KAC12], and attacks on mining [ES14, NKMS16] (cf. Chapter 3). Furthermore, knowledge of the topology can facilitate the analysis of P2P networks for research purposes (cf. Chapter 5). In this chapter we present and analyze four different methods for inferring the topology of the Bitcoin P2P network.

Table 6.1 gives an overview of the used methods and which aspect of the network layer is being exploited. As can be seen the methods exploit various aspects, which can be located at different *positions* of the network layer of permissionless blockchains: *Transaction accumulation* is a client implementation specific aspect, which can be easily changed in a client's implementation. The handling of *double spends* on the network layer is predetermined by the handling of double spends on the consensus layer, therefore, the effects of changes to this aspect are far more severe than changes to an implementation specific aspect. Finally, the methods targeting timing and peer discovery exploit the communication and connectivity, respectively, between peers itself. Preventing these analysis methods comes with inherent tradeoffs negatively affecting communication delay and connectivity between peers.

Before the methods are presented and analyzed in detail in Sections 6.1 to 6.4, we define the considered topology inference problem and briefly cover related work in the field of topology inference.

Scenario Definition The presented scenario definition has been previously published in [GNH18]. We will now define the general considered scenario that is valid for all considered topology inference methods. Additional assumptions and restrictions required for single topology inference methods will be discussed later.

Table 6.1: Overview of the analyzed topology inference methods.

Inference Method	Layer	Exploited aspect	Countermeasures
Section 6.1	Implementation	Transaction accumulation	Modify implementation
Section 6.2	Application Logic	Double spends	Relay double spends
Section 6.3	Communication	Timing	Increase TX delay
Section 6.4	Connectivity	Peer discovery	Randomize peer discovery

Let $G = (V, E)$ be the undirected graph modeling the peers (V) and connections (E) of the Bitcoin network. Given a subset $R \subseteq V$ of the *reachable* peers of the network, the adversary[1] tries to infer all connections between all peers in R. The inference can lead to false positives (i.e., inferring a connection although no connection exists) and false negatives (i.e., not inferring a connection although a connection exists). We will use precision (i.e., the share of inferred connections that are true positives) and recall (the share of existing connections that were inferred) as metrics to describe the success of the inference.

We assume that the adversary can run a small number of peers, which can connect to as many other peers as possible. This number is limited by the number of reachable peers and the network capabilities of the adversary.[2] We also assume that the adversary is able to precisely estimate the latency between its own peers and remote peers, e.g., based on the observation of Bitcoin `ping` messages or ICMP ping messages. The adversary is not assumed to have information that an ISP or state actor organization might have about connections and traffic of other peers. We do not consider stronger adversary models (e.g., ISPs), as these adversaries could simply monitor the network traffic in order to infer the network topology.

Related Work Topology inference in Bitcoin has been the subject of several previous works. Peer discovery in Bitcoin allows clients to query their neighbors for IP addresses of other peers in order to establish connections to them. The queried neighbor then sends a list with IP addresses along with a `lastseen` timestamp. Until March 2015 the timestamp was not randomized sufficiently and allowed Miller et al. [MLP+15] to exploit this mechanism and infer the network topology. Peer discovery can also be exploited for topology inference by sending IP addresses that do not correspond to reachable peers, but are sent to remote peers so that the announcement of these IP addresses by other peers can be observed [BKP14].

Furthermore, a comparison of the node degree distribution of peers on the Bitcoin and Bitcoin Cash network is performed in [JW18]. However, the paper does not state the used method for finding out the node degrees of remote peers.[3]

[1] As stated, topology inference is a dual-use technology, which can be used for research purposes as well as for adversarial purposes. For the sake of simplicity, we refer to the entity performing the topology inference as adversary, independent of their intentions.

[2] Our measurements show that maintaining connections to ≈10,000 peers consumes about 20 Mbit/s.

[3] One of the authors also claimed to be Satoshi Nakamoto without providing a verifiable proof for

6.1 Exploiting Transaction Accumulation for Topology Inference

The topology inference approach presented in this section exploits an implementation aspect in the forwarding of transactions of the Bitcoin implementation Bitcoin Core (*bitcoind*). Hence, we will first describe the exploited client behavior and refine the adversary model for the presented topology inference method. Then, the inference method will be presented, discussed, and evaluated. The content presented in this section has been previously published in [GNH18].

6.1.1 Fundamentals & Assumptions

As described in Section 2.2.3, transactions and blocks are flooded through the Bitcoin network using three types of messages (`INV`, `GETDATA`, and `TX`/`BLOCK`, respectively). After receiving and validating a transaction, `INV` messages are not sent out immediately to a client's neighbors, but are delayed according to a non-deterministic function. Bitcoin Core maintains one outgoing queue for each connected peer for storing these delayed transactions. When a new transaction is received or created, this transaction is added to the queues for all neighbors. Therefore, each queue contains all transactions that are to be announced to that peer. At certain times all messages in a queue are announced to the neighbor via a single `INV` message.[4] Every time the elements of the queue are sent to the neighbor, a new sending time is determined. These times are chosen according to an exponential distribution with a mean of 5 seconds for incoming connections and 2 seconds for outgoing connections. This mechanism has the property that all transactions received between two sending timestamps are sent in one single `INV` message.

We will demonstrate how to exploit this *transaction accumulation* for topology inference by creating and publishing transactions in a certain way. In addition to the assumptions made previously regarding the adversary model, we now also assume that the adversary is able to create a large number of transactions. These transactions can transfer funds between addresses controlled by the adversary, however, transaction fees still have to be paid.

6.1.2 Topology Inference Method Description

Assume for now that the adversarial monitor peer v_M is connected to all peers $v_i \in V$ of the network. The adversary creates one transaction $t_i \in \tau$ for each connected peer v_i. All transactions are independent and not conflicting in any way (i.e., they are spending different outputs). All transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the same time. Afterwards, the adversary monitors the first `INV` messages that will be received by v_M from all connected peers, and infers information about the topology by using the following inference rules:

that claim.

[4] If there are more than 35 transactions in the queue (which occurs only infrequently), only 35 transactions are announced at once.

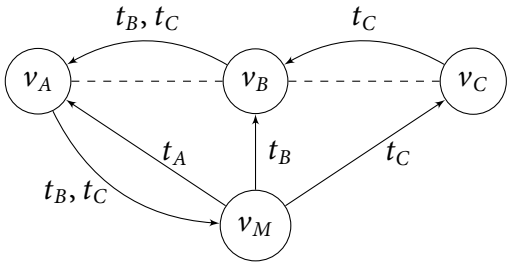

Figure 6.1: Exploiting transaction accumulation for topology inference. Dashed lines indicate existing connections. Solid lines indicate the transmission of transactions [GNH18].

1. If the first `INV` message that peer v_A sends to v_M contains only t_B (i.e., the transaction sent to v_B) and no other transaction from the set of created transactions τ, then v_A and v_B are directly connected.

2. If the first `INV` message that peer v_A sends to v_M contains more than one transaction from the set of created transactions τ, at least one of the peers associated with the announced transactions is connected to v_A.

Let us consider the scenario depicted in Figure 6.1 to demonstrate that the presented inference rules do not lead to false positives if all assumptions are met. A formal proof is given in the Appendix (A). v_M is connected to v_A, v_B, and v_C. v_A is connected to v_B, v_B is connected to v_C. After the transactions were sent by the adversary, each peer has only the transaction designated for itself (and transactions created by other participants, which can be ignored). Statement 1 is equal to *If v_A and v_C are **not** directly connected, then v_A will **not** send an `INV` message that contains only t_C and no other transaction from the set of created transactions τ.*

Because v_A and v_C are not connected, t_C has to be relayed by another peer (v_B) to v_A. As we assumed that the adversary is connected to all peers, the adversary is also connected to v_B and has sent a transaction t_B to v_B. Because of the queuing mechanism of *bitcoind*, v_B's queue for v_A already contains t_B. Therefore, t_B and t_C will be announced together to v_A, which announces them together to v_M. It is also possible that t_B will be sent earlier than t_C, because the queue at v_B is sent between the reception of t_B and t_C by v_B. However, it is not possible that t_C arrives earlier than t_B at v_A.

This scenario also explains the second statement: If v_A sends an `INV` message that contains t_B and t_C, the adversary does not know whether v_B, v_C, or both are directly connected to v_A. The transactions initially sent to all peers serve as identifiable flags that the remote peers attach to the first group of transactions they forward after receiving their transaction. This allows the adversary to reconstruct the path of transactions and thereby infer connections between peers.

6.1.3 Discussion & Variants

While this topology inference approach is possible under perfect conditions, there are several issues that can arise when not all assumptions are met.

If there are peers on the network that the adversary is not connected to, false positives can occur. Consider again the scenario depicted in Figure 6.1, but let us assume that v_M is not connected to v_B. Then, v_B would not have received a transaction t_B, and the `INV` message sent by v_A would only include t_C, which would lead to the wrong conclusion that v_A and v_C are directly connected.

False positives can also occur when the adversary cannot guarantee that all transactions arrive at all peers at the same time. While the latency measurement might be precise in general, temporal changes, e.g., due to bandwidth peaks, are possible and hard to foresee by the adversary. Furthermore, sending several thousand transactions within a few hundred milliseconds in a coordinated way can require much bandwidth and computational effort.

The reception of `INV` messages containing multiple transaction from τ does not advance the performed topology inference, because it does not allow to draw any definite conclusions from the observation. The transactions included in the `INV` messages of remote peers is determined by the sending times of the respective queues and is unknown to the adversary. Therefore, even when all assumptions are met, the success of the approach depends on the order in which the remote peers forward transactions to their neighbors. This means that repeating the approach (possibly very often) is required in order to infer a large number of connections.

Another issue with the approach is that it is not possible to explicitly target a specific remote peer in order to infer the connections of that peer only. Instead, the inferred connections are a subset of all existing connections, which cannot be influenced by the adversary.[5]

Variant *DS*: We will now present a variant of the discussed approach that reduces the cost by reducing the incurring transaction fees. Assuming 10,000 peers on the network and transaction fees of $1 per transaction, the cost for one run of the approach is $10,000.[6] A possibility to reduce this cost is to still create one transaction per peer, but to create these transactions so that they are all double spends of only a few different outputs. The number of different inputs among all transactions is a parameter freely chosen by the adversary (e.g., DS_3 denotes variant *DS* with three different inputs). Each transaction is still unique (e.g., by having different outputs), which enables the mapping of one transaction to one remote peer. That way, the adversary has to pay only for those transactions that get included in the blockchain. However, this approach can cause transactions to be dropped, which can cause false positives. We will evaluate the effect of double spendings on this approach in the next subsection.

6.1.4 Simulation Results

We will now briefly describe the used simulation setup before the results of the simulation are presented and discussed. Simulations are performed using a discrete event simulation with an adapted version of the simulation model presented in Chap-

[5] The subset is influenced by the times at which peers flush their queues and the latencies between peers.

[6] Due to fluctuations in transaction fees and exchange rates, this calculation is just an example.

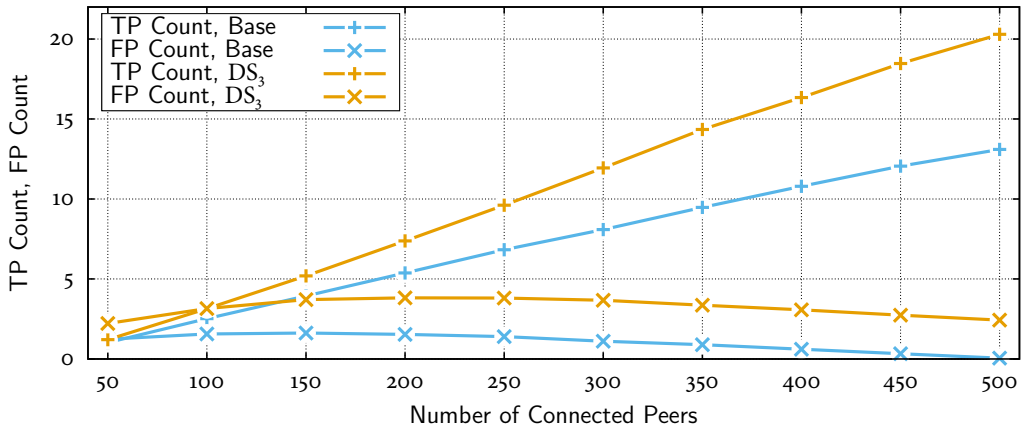

Figure 6.2: Number of true positives and false positives per run for the base approach and the variant *DS* with three different inputs [GNH18].

ter 5. The total simulated number of peers on the network is 500, which roughly corresponds to the number of peers on the Bitcoin testnet. The network topology is generated by creating eight outbound connections to uniformly chosen peers for each simulated peer. This results on average in eight incoming and 16 total connections per peer. The adversary is modeled as a specific peer that establishes a large number of connections (depending on scenario) and sends and receives the transactions according to the presented inference strategy.

While the simulation matches the general behavior of the Bitcoin client, several simplifications were made. First, we model the three-step transaction propagation process (`INV` - `GETDATA` - `TX`) as one single event. Secondly, the latencies between peers are chosen according to a normal distribution (μ = 100 ms, σ = 50 ms, truncated to [1 ms, 6000 ms]). Thirdly, when peers forward transactions and have more than 35 transactions in their queue, they choose the transactions to forward uniformly at random, but prefer transactions created by the adversary[7]. Therefore, our simulation is not a precise model of the Bitcoin network or testnet and the results should be seen as a proof of concept.

Figure 6.2 shows the true positive (TP) and false positive (FP) count depending on the number of connected peers for the base variant and variant DS with three different inputs for one run of the approach. If the adversary is connected to all 500 remote peers, one run of the approach results in about 20 correctly detected connections for variant DS, and in about 13 correctly detected connections for the base variant. Reduction of the share of connected peers leads to a decline in the true positive count. While we expected the false positive count of variant DS to be higher than that of the base variant, surprisingly, variant DS also results in a higher true positive count compared to the base variant. Double spends limit the propagation of individual transactions, because they are dropped at all peers that already received

[7] This models the scenario that the adversary pays higher transaction fees than the fees for the other transactions.

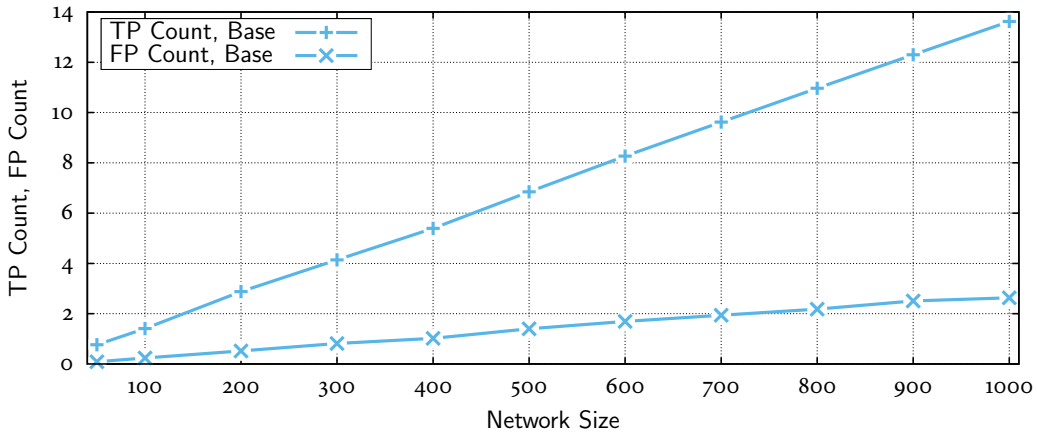

Figure 6.3: Number of true positives and false positives depending on the network size for v_M being connected to half of the peers [GNH18].

another transaction with the same input. This limitation of propagation is actually beneficial for the approach, because only single-hop propagation of each transaction (i.e., from one remote peer to another and back to v_M) is required and leads to the correct detection of a connection.

As we performed the simulation with a fixed number of 500 peers, the question of the effect of the network size on the inference quality arises. Hence, we simulated the network with a varying number of peers for an adversary connected to half of all network peers. Figure 6.3 shows a that linear relationship between the number of peers and the TP and FP counts exists. Therefore, a network with twice the number of peers results in about twice the number of true positives at the same false positive rate.

6.1.5 Experimental Results

In order to perform a ground truth validation of our simulation results, we set up several peers on the Bitcoin testnet: Two peers perform the role of the adversary peers and connect to all reachable public peers (around 520 connections during the experiments in November 2017[8]). Another five peers running Bitcoin Core (0.15.0.1) serve as validation targets. These peers establish eight outgoing connections and are reachable to the adversary peers via IPv4 and IPv6. In this setup the adversary peers are connected to all neighbors of the validation targets, which is a best-case scenario for inference.

During the experiments, one of the adversarial peers sends transactions to other peers so that they all arrive at the same time at their destination. The latency to remote peers was measured using ICMP ping, TCP SYN packets, and Bitcoin ping messages as described in Chapter 4.

We performed 50 runs of variant *DS* of the described inference approach using transactions with three different inputs. A total of 632 unique connections were detected, which roughly conforms to our simulation results. Out of these 632 connec-

[8] Peers were found using `https://github.com/ayeowch/bitnodes/`

tions, only 9 connections were connections from or to one of our validation peers. From these 9 detected connections, only 6 actually existed, which corresponds to an observed precision of 67 %.[9] Roughly estimating the total number of connections on the testnet to be 4,160[10], and assuming a precision of 67 % results in a recall (with respect to all connections of the network) of about 10 % after 50 runs for a total cost of $50 * 3 = 150$ transaction fees.

6.1.6 Discussion

In this section we presented and analyzed two variants of a topology inference approach that exploits the accumulation of transactions by the Bitcoin client *bitcoind*. Simulation results show that although the base variant is technically feasible, the costs for actually performing the variant are unbearable high (i.e., one transaction fee per network peer for one run). Contrary, the costs for performing the variant *DS* are very low (i.e., constant in the size of the network). Furthermore, simulation results suggest a decent inference quality and show that the number of inferred connections scales linearly with the size of the network. Although the small sample size of the experimental results only allows very rough estimates of the inference quality to be expected for variant *DS*, the expected recall is in the range of 10 % at the cost of 150 transaction fees for a network with 500 peers.

While these results sound promising, there are two main limitations of the presented approach. First, because it is not possible to infer the connections of a specific peer only, rather than inferring connections of random peers of the network, it is hard to thoroughly validate the approach in real-world networks. For adversarial purposes, this lack of influence on which connections are inferred prevents targeted attacks, especially taking into account that topology inference is only an intermediate goal for further attacks. For scientific purposes, the lack of validation of the approach leads to results that are hard to justify and which should not be used in any models without careful consideration.

Secondly, the variant *DS* produces false positives even if all assumptions are met. False positives can also occur when the adversary is not connected to all peers. Additionally, false positives can occur when other client implementations are used that do not exhibit the exploited transaction accumulation behavior. All these causes for false positives typically exists in real-world networks. Hence the reliability of the approach is limited.

Finally, as transaction accumulation is implementation specific, the implementation could be easily modified to counter this topology inference method. For instance, if each transaction that is stored in an outgoing queue is sent with a certain probability only, transactions could *overtake* each other in a client's queue, making the assumptions, on which the discussed method relies, invalid. As the presented approach shows that transaction accumulation does leak some information, such countermea-

[9] Because of the small sample size, the real precision can strongly deviate from the observed precision.

[10] 520 peers with 8 connections each.

sures could also prevent possible advanced approaches that also exploit transaction accumulation for topology inference or for deanonymization of users.

6.2 Exploiting Double Spends for Topology Inference

The content presented in this section has been previously published in [GNH18]. One major drawback of the approach presented in Section 6.1 is that it is not possible to infer the connections of a specific peer only, rather than inferring connections of random peers of the network. This is not only problematic for adversaries, but also makes validation a challenge. In this section we will describe and analyze a topology inference method that relies on the transaction validation behavior of clients with regard to *double spending* transactions.

We briefly recap the transaction validation behavior described in Chapter 2: When a peer receives a transaction, it validates the correctness of the transaction. This includes checking the correct format, checking whether the sum of input values is at least as large as the sum of output values, and checking whether the inputs of the transaction are actually spendable. Because every transaction output can only be spent once, a transaction with an input that was already spent by a transaction received earlier is regarded as invalid and dropped silently. We will now demonstrate how to exploit this behavior regarding *double spends* for topology inference. As in the previous section, we assume that the adversary is able to create and publish own transactions on the network.

6.2.1 Topology Inference Method Description

Again, assume for now that the adversarial monitor peer v_M is connected to all peers $v_i \in V$ of the network. One of the connected peers is the target peer v_T, the connections of which the adversary wants to infer. The adversary creates one transaction $t_i \in \tau$ for each connected peer v_i, except for the target peer v_T. All transactions have the same input, i.e., they are double spends, but all transactions are unique, e.g., by specifying different output addresses. Again, all transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the same time. Then the adversary monitors which transaction the target peer v_T forwards to the monitor peer v_M and can conclude that the peer associated with the forwarded transaction is directly connected to the target peer v_T.

Let us consider the scenario depicted in Figure 6.4 to demonstrate that the proposed strategy reveals existing connections of the target peer if all assumptions are met. A formal proof is given in the Appendix (A). The monitor peer v_M is connected to v_A, v_T, v_B, and v_C. The target peer v_T is connected to v_A and v_B, while v_B is also connected to v_C. After the transactions were sent by the adversary, every peer only has the transaction designated for itself, and v_T has no transaction received yet. Every peer will only accept and forward *exactly one* of the created transactions, because they are all double spends of the same output. Therefore, if v_C forwards t_C to v_B (dotted line), v_B will drop t_C because of the earlier reception of the conflicting transaction t_B.

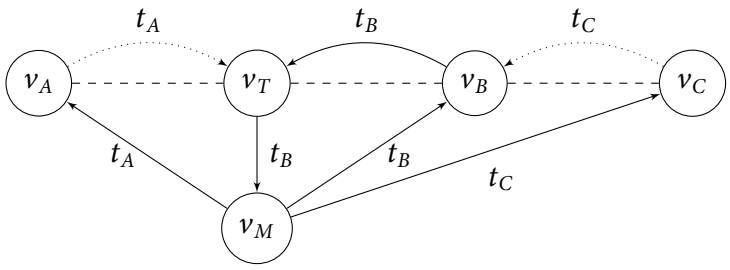

Figure 6.4: Exploiting double spends for topology inference. Dashed lines indicate existing connections. Solid lines indicate the transmission of transactions. Dotted lines indicate dropping of transactions by the receiver because of an earlier reception of a conflicting transaction [GNH18].

Because the target peer v_T has not yet received any of the conflicting transactions, it will accept exactly one transaction forwarded by one of its neighbors (transaction t_B in Figure 6.4). This transaction gets forwarded to the monitor peer v_M and indicates a neighbor of the target peer v_T.

6.2.2 Discussion & Variants

If the adversary is not connected to all peers of the network, or if the transactions are not received by all peers at the same time, false positives can occur. The reason is basically the same as for the approach exploiting transaction accumulation discussed in Section 6.1: A neighbor of v_T that did not receive its double spending transaction from v_M will accept another double spending transaction t_i from another neighbor v_i and forward that transaction to v_T, which may forward t_i to the adversary causing the false inference of a connection between v_T and v_i. Obviously, if the adversary cannot establish a connection to v_T, the connections of v_T cannot be inferred using the discussed approach. We will now discuss three variants of the presented approach that aim at optimizing the inference even when not all assumptions are met.

Variant *Count*: When repeating the approach several times, one would expect the transactions associated with real neighbors (true positives) to be sent to the adversary by v_T more often than those of peers that are not connected to v_T (false positives), because those transactions have to be relayed by another peer and should be slower. In order to reduce false positives, the approach can be repeated and connections are only identified, if the number of transactions indicating a specific peer as a neighbor of v_T is larger than a certain threshold.

Variant *Ignore*: Assume that t_A is forwarded by v_T to v_M. If the adversary was unable to synchronize the reception of all transactions at all remote peers (e.g., due to bad latency estimation or bandwidth limitation), it is possible that t_A is *also* forwarded to v_M by another peer, say, v_B. As such a reception indicates the violation of a key assumption and v_T might have received t_A from v_B rather than directly from v_A, the adversary can opt to ignore the result without concluding a connection between v_T and v_A.

Variant *Suppress*: The cost for a single run of the approach is one transaction fee.

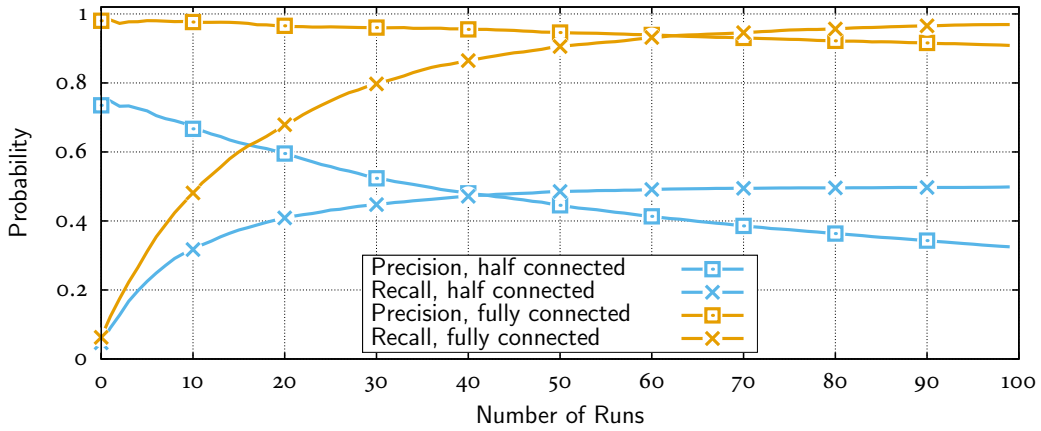

Figure 6.5: Precision and Recall depending on the number of runs with v_M being connected to 250 (half connected) and 500 (fully connnected) of 500 peers [GNH18].

However, one single run reveals at most one connection of the target peer. In order to infer more connections, additional runs are necessary, which each come at the cost of one transaction fee. Which connection can be inferred depends on which transaction arrives first at v_T, which is determined by the sending times of the remote peers and the latencies between peers. With bad luck (or single clients being very fast), multiple runs of the approach can all result in inference of the same, already known, connection. Variant *Suppress* slightly modifies the approach to eliminate the repeated inference of the same connection. Consider again the example depicted in Figure 6.4 and assume that the adversary inferred the connection between v_T and v_B in the first run of the approach. For the next run, we (1) want the transaction t_B to be dropped at v_T and (2) we do not want v_B to forward any other transaction t_i. While simply not sending any transaction to v_B would satisfy the first requirement, it would make v_B a hidden node and violate the second requirement. Therefore, we modify the way the double spending transactions are created. Assume there are two unspent outputs i_1 and i_2 that will be used as inputs to the transactions in the following way:

- All peers v_i, except for v_T and v_B, receive transactions t_i spending i_1 only.
- v_T receives a transaction t_T spending i_2 only.
- v_B receives a transaction t_B spending i_1 and i_2.

This approach satisfies both requirements: v_T will drop t_B because it is a double spend of i_2. Any transaction t_i will be dropped by v_B because they are double spending i_1. Yet, any transaction t_i will be accepted by v_T because they are spending different outputs (i_1 and i_2).

6.2.3 Simulation Results

We simulated the approach exploiting double spends with the same simulation setup as described in Section 6.1.4. Figure 6.5 shows how recall and precision develop

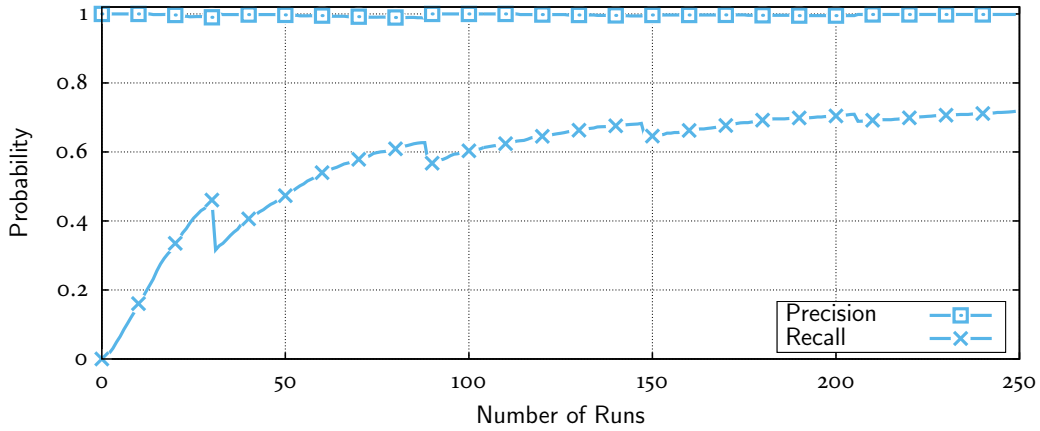

Figure 6.6: Precision and recall depending on the number of runs for variant *Count* and v_M being connected to 375 of 500 peers[GNH18].

depending on the number of runs for the base version of the approach. Please note that the statistical population of connections to be inferred is now limited to the connections of the target peer v_T. Hence, perfect recall and precision here mean that all connections of the target peer have been correctly identified, and not that all connections of the complete network have been correctly identified. If the monitor peer v_M is connected to all peers of the network, the recall reaches 95% after 100 runs while the precision decreases slowly. The precision decreases because the latency between the adversary peer and other peers is simulated probabilistically, i.e., the simulated adversary cannot ensure that all generated transactions are received at the same point in time. Therefore, it is possible that remote peers receive a double spending transaction from one of their neighbors before receiving *their* double spending transaction from the adversary peer.

If the adversary is connected to only half of the peers of the network, the expected maximum possible recall is 50 %, because the adversary is on average only connected to half of the neighbors of v_T. As described above, the target's neighbors being not connected to the adversary cause false positives and thus the precision is lower than for the fully connected scenario.

Figure 6.6 shows precision and recall for the variant *Count* of the approach exploiting double spends. As can be seen, the recall increases in steps. These steps are caused by adjusting the threshold for the required number of receptions. While this variant can be used to reach high precision, the recall is limited even after more than 200 runs.

Figure 6.7 shows precision and recall of the variant *Suppress* with v_M being connected to all peers. Using only this variant results in the recall growing faster, because this variant prevents neighbors from being detected multiple times. However, not only true neighbors are detected faster, but also false positives, which results in a faster declining precision. If v_M is not connected to all peers, the precision falls even faster, because the likeliness that a detection is a false positive is higher.

The precision can be improved by combining the variants *Suppress* and *Ignore*, for

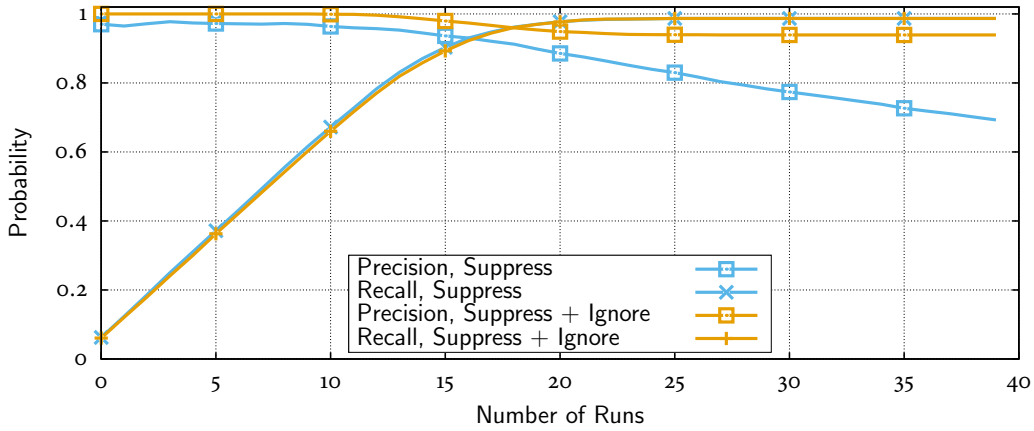

Figure 6.7: Precision and recall depending on the number of runs for variants *Suppress* and *Suppress + Ignore* with v_M being connected to 500 of 500 peers (fully connected) [GNH18].

which precision and recall are also shown in Figure 6.7. Combining both variants results in a recall of 96 % after 25 runs with a precision of about 94 % if the monitor is connected to all peers.

6.2.4 Experimental Results

We validate the approach in the Bitcoin testnet with the setup described in Section 6.1.5 with the exception that the adversarial peers do not send any transactions to the IPv6 addresses of the validation targets. The reason for this exception is that otherwise the presented approach infers connections between the IPv4 and IPv6 addresses of the validation target. While this might also be an interesting application for the approach, it would impair our validation.

We ran the approach six times against each of the five validation targets with 50 runs each using the combination of the variants *Suppress* and *Ignore*. Analyzing the data generated during the experiments using different combinations of variants results in various combinations of precision and recall. Two of them using *Suppress* and *Ignore* are shown in Figure 6.8. The combination of the variants *Suppress* and *Ignore* results in a recall of 60 % and a precision of 97 %. The recall can be improved though by relaxing the restrictions imposed by *Ignore* by using only the variant *Suppress*. This combination results in a recall of 87 % and a precision of 71 % (also shown in Figure 6.8) for a total cost of 99 transaction fees. Again, note that precision and recall refer to the connections of the target peer only.

6.2.5 Discussion

In this section we presented and analyzed a topology inference method that exploits the handling of *double spending* transactions by Bitcoin clients. Our simulation as well as experimental results in the Bitcoin testnet indicate a high inference quality

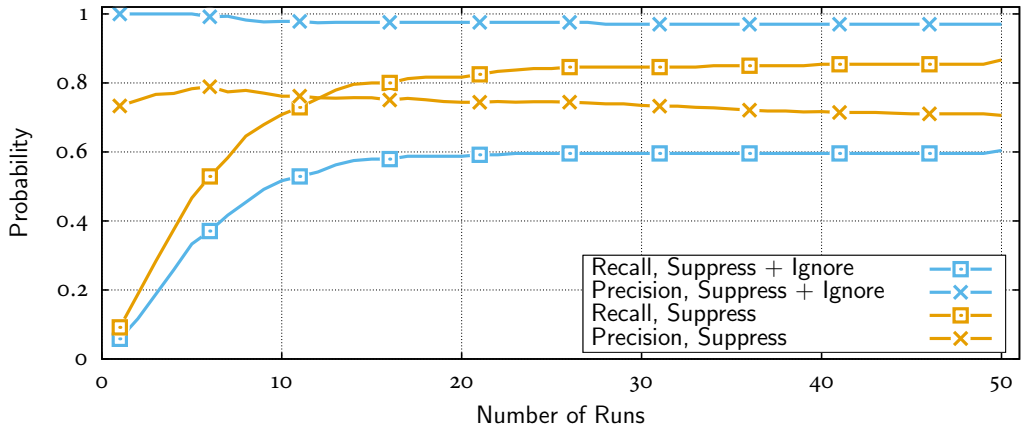

Figure 6.8: Experimental Results: Precision and recall depending on the number of runs using variant *Suppress* and *Suppress+Ignore* [GNH18].

at low costs (i.e., 99 transaction fees per target peer). However, we emphasize again that the experiments were performed using a favorable scenario, i.e., the adversarial peer was connected to all neighbors of the validation peers. The existence of *hidden* neighbors as possible in reality could severely impair the inference quality.

Several countermeasures against the approach are possible, however, each countermeasure comes with certain drawbacks. First, an obvious countermeasure would be to forward double spends, which, however, would create the potential for DoS attacks: Malicious peers could publish large numbers of double spending transactions at the cost of only one transaction fee on the network. These transactions would all be flooded through the network, consuming a large amount of bandwidth at participating peers.[11]

Another countermeasure could be to not always forward the transaction that was received first, but randomly deciding which double spending transaction will be forwarded. This countermeasure, however, could affect security against double spending attacks in zero-confirmation payments [KAC12, DW13].

Furthermore, individual peer operators may choose to deny incoming connections, which prevents the discussed approaches from working, but is not desirable from an overall network's perspective. On the other hand, operating a reachable peer with a large number of incoming connections from unreachable peers also impedes the presented inference approaches. Finally, because of the large number of transactions created, the proposed topology inference method can be observed by monitoring large parts of the network (cf. Chapter 4).

[11]The Bitcoin Cash client *Bitcoin Unlimited* has been recently modified to relay double spending transactions in order to improve the detection of double spending attempts when accepting zero-confirmation payments (`https://github.com/BitcoinUnlimited/BitcoinUnlimited/pull/1109`).

6.3 Exploiting Timing for Topology Inference

While the topology inference method presented in Section 6.1 exploited an implementation specific aspect of the network layer, the method presented in Section 6.2 exploited network layer behavior that is required because of consensus layer requirements. In this section, we present a topology inference method that is completely independent of the implementation and the consensus layer. Instead, it targets the information propagation of the P2P network itself. Hence, the presented method can be applied to all open flooding P2P networks. The only required assumptions are that messages are flooded through the whole network, that each message is uniquely identifiable, and that it is possible to connect to arbitrary peers of the network in order to receive the propagated messages from remote peers.

An overview of the presented method is depicted in Figure 6.9. A monitor peer connected to (almost) all reachable peers of the network observes the message propagation process by logging the timestamp of reception of each message from each remote peer. For each message, a set of tuples *(reception time, sending peer)* is observed – one tuple for each forwarding peer. Intuitively, the reception times in the observations caused by one message correlate with the network topology. Therefore, this observation is then compared to a *propagation delay model*, which enables us to assign likelihoods for each pair of IP addresses for being directly connected or not. Finally, the likelihoods derived from the observation of multiple messages are combined using a *maximum likelihood estimation*.

The remainder of this section is structured as follows. First, the timing based inference method will be presented and validated using an analytical model and simulations. Then, we present a real-world validation using experiments in the Bitcoin network. Finally, we analyze the trickling countermeasure against the presented topology inference method. The content presented in this section has been previously published in [NAH16] and in [NH18].

6.3.1 From Observations to Network Topology

We will now show how to infer the topology of a network based on the comparison between observations of the information propagation delay in the network and a propagation delay model. For this, we will first formalize the problem considered. Then, the approach is presented and validated. Finally, limits of the approach are discussed.

Observations

Every time a message is forwarded to an adversary's monitor, a tuple *(reception time tr, sending peer v)* is created at the monitor. Therefore, for each unique message m, the adversary observes a set of tuples $O_m = \{(tr_0, v_0), (tr_1, v_1), ...\}$. As we assume the adversary to be aware of the latency from the monitor node to other peers, the adversary can subtract this latency from the *reception time* and get an estimate of the *sending time*: $O'_m = \{(t_0, v_0), (t_1, v_1), ...\}$, where the first tuple (t_0, v_0) represents the message's sending by the originator (v_0) of the message.

We convert the absolute timestamps of O'_m to time differences relative to the creation

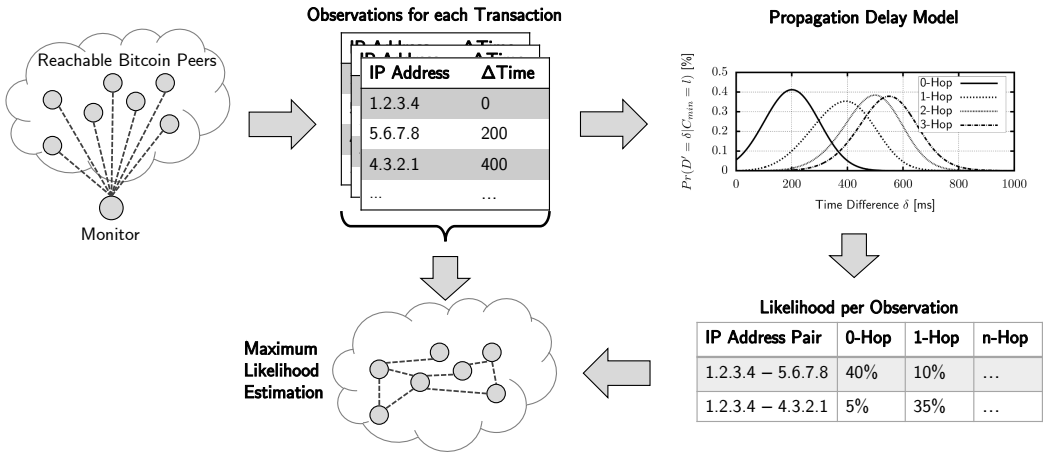

Figure 6.9: Timing-based topology inference method.

time t_0 ($\delta_1 = t_1 - t_0, \delta_2 = t_2 - t_0, ...$). Each of these time differences δ_i is a sample of the delay between the originator of the message and the peer v_i, which forwarded the message to the adversary's peer. Grouping all time differences of all messages by these two peers results in a set of measured delays for each pair of peers $\Delta_{v_1,v_2} = \{\delta_1, \delta_2, ...\}$. The set contains one time difference for each message that was created by v_1 or v_2. Therefore, the set is empty for all pairs of peers that both did not create a message during the observation period.

We will now focus on how to estimate the shortest path length C_{min} between two peers in the network (i.e., the shortest sequence of edges between both peers) based on the observations made.

Inferring the Shortest Path Length

The following estimation compares the observations made to the analytical propagation delay model described in the Appendix (B). The parameters of the modeled network are the number of peers and the probability of existence of each possible connection. The network in our delay model is assumed to match a random graph model [ER59]. Furthermore, we assume that the zero-hop delay distribution (i.e., the latency between any pair of peers) is known.

Using the model, we can calculate the a priori probability that the shortest path C_{min} between two randomly chosen peers has length l (i.e., $P(C_{min} = l)$). The discrete random variable D models the propagation delay between two randomly chosen peers (discretized to e.g. milliseconds[12]). The model enables us to calculate the probability of observing a specific delay δ: $P(D = \delta)$. It also allows calculation of the probability of observing a specific delay δ assuming that the shortest path length between sender and receiver equals l: $P(D = \delta | C_{min} = l)$.

We are now looking for a method to assess how likely it is to observe a specific set of time differences, depending on the shortest path length between the two ob-

[12] Although a delay could be modeled as a continuous random variable, we opt for a discrete model to enhance readability and closely match our simulation model.

served peers. A relationship between the unknown shortest path length C_{min} and the observed time difference δ is given by $P(D = \delta | C_{min} = l)$.[13] Evaluation of this formula for each possible shortest path length and all observations allows a comparison between the resulting probabilities and lets us decide, which shortest path length has the maximum likelihood.

The likelihood function for a set of observed time differences Δ_{v_1,v_2} and a length of the shortest paths l follows from the definition of a likelihood function as

$$L(C_{min} = l | \Delta_{v_1,v_2}) = P(C_{min} = l) \cdot \prod_{\delta \in \Delta_{v_1,v_2}} P(D = \delta | C_{min} = l).$$

The maximum likelihood estimation of the shortest path length between v_1 and v_2 is computed by selecting the largest likelihood among all shortest path lengths, resulting in

$$\hat{l} = \arg\max_{l} L(C_{min} = l | \Delta_{v_1,v_2}).$$

For an asymptotically large number of observations, $\hat{l}$ converges to the real value of C_{min}. However, the estimated shortest path length can differ from the real shortest path length, if, for instance, the observation contains only a few values and many of them are outliers. Therefore, some measure of confidence in the guess is required. The quotient of the likelihood function of $\hat{l}$ and the sum of all likelihood functions gives the probability that the guess is in fact correct (*certainty*)

$$P(C_{min} = \hat{l} | \Delta_{v_1,v_2}) = \frac{L(C_{min} = \hat{l} | \Delta_{v_1,v_2})}{\sum_l L(C_{min} = l | \Delta_{v_1,v_2})}. \tag{6.1}$$

Actually, this equation can be calculated not only for $C_{min} = \hat{l}$, but also for all other values of C_{min}, denoting the probability that each C_{min} is in fact correct. This corresponds to assigning probabilities to each shortest path length.[14]

Simulation Results & Validation

We will now show the effectiveness of the proposed timing-based topology inference method by examining the resulting error rates. A flooding network was simulated that generated the observations as input for the timing analysis. The timing analysis resulted in a guess which edges of the network exist. By comparing the estimate to the simulated network, we can judge the quality of the presented technique.

[13] This problem can be formulated as a very simple Hidden Markov Model (HMM): Each hidden state represents one minimum path length C_{min} between two peers in the network. The observable states of the HMM are the time differences δ. The transition probabilities from each hidden state l to the observable states equal the probability of observing a delay of δ assuming a minimum path length of l: $P(D = \delta | C_{min} = l)$.

[14] The shortest path length C_{min} can also be seen as a probabilistic information source, which outputs l with a probability of $P(C_{min} = l | \Delta_{v_1,v_2})$. The entropy of this information source then equals the uncertainty in the estimation. The difference $P(C_{min} = l) - P(C_{min} = l | \Delta_{v_1,v_2})$ is an upper bound for the information content of the observation Δ_{v_1,v_2}.

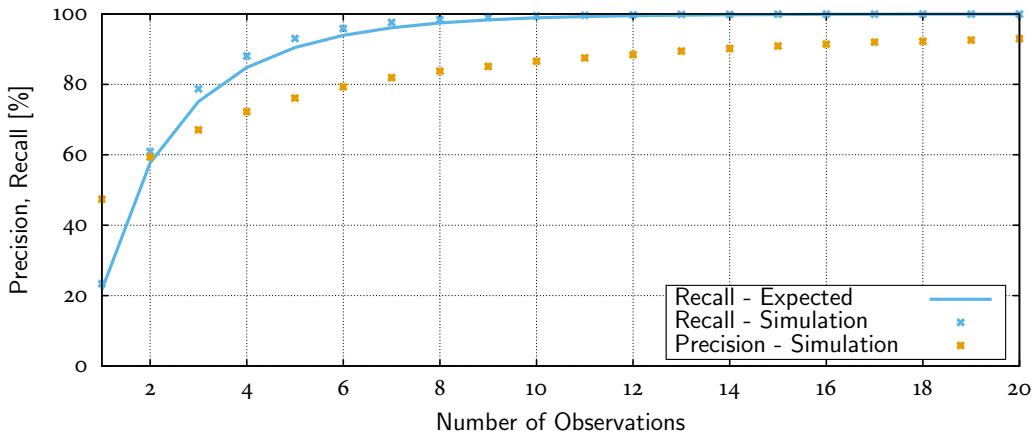

Figure 6.10: Precision and recall in a simulated network wrt. the number of observations per pair of peers [NAH16].

The estimation can lead to two kind of errors: false positives and false negatives. A false positive occurs if a specific edge is postulated although it does not exist. A false negative occurs, if an existing edge is not detected by the analysis. Obviously, the more observations are in Δ_{v_1,v_2}, the less likely are both kinds of errors, as each observation originates from the *correct* distribution.

As before, we use precision (true positives divided by the sum of true and false positives) and recall (true positives divided by the number of elements that should have been detected) as measures for the quality of the topology inference method. Figure 6.10 shows how precision and recall increase with the number of observations in the performed simulation. Additionally, the calculated expected recall is depicted. The recall converges quickly to 100 %, whereas the precision rises much slower and reaches 90 % after 12 observations. Both, expected and experimental recall match very well.

Although the error rates look extremely promising, it should be noted that the simulation experiment makes some idealized assumptions that cannot be matched in the real world: First, the delay distribution as assumed by the adversary equals the real delay distribution used in the simulation. In reality, an adversary has to estimate the delay distribution, which will only be an approximation (cf. Section 5.3). Additionally, the network and the delay distribution is static in the simulation, whereas churn and jitter are known to occur in real-world networks. We will leave a sensitivity analysis of the delay distribution estimation used by the adversary as future work and give a proof of concept of the proposed method in the real Bitcoin P2P network in Section 6.3.2.

Limits

As just shown, there exists a relationship between the number of observations and the quality of the estimation. Intuitively, the shape and especially the overlap of the conditional delay distributions for each shortest path length also affect the estimation's quality: highly overlapping delay distributions impede correct estimations, whereas observations from non-overlapping distributions are easy to map to shortest path lengths. We will first illustrate this relationship before analyzing the effectiveness and

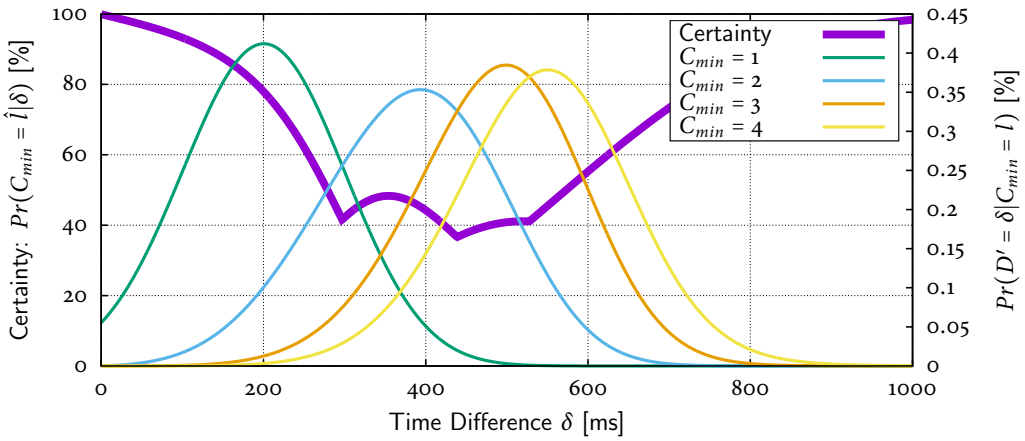

Figure 6.11: Conditional delay distributions and certainty wrt. to observed delay. Scenario: 6,000 nodes, 16 connections per node on average, zero-hop latency distribution according to a normal distribution (μ = 200 ms, σ = 100 ms) [NAH16].

tradeoffs of a countermeasure against timing analysis.

Figure 6.11 shows the probabilities $P(D = \delta|C_{min} = l)$ depending on the observed time difference δ for $C_{min} \in \{1, 2, 3, 4\}$ for a given scenario. Additionally, the *certainty*, as calculated by Equation 6.1, of such an observation is shown. It can be seen that observing small time differences (below 200 ms) leads to the highest certainty, because of the fact that these delays result from $C_{min} = 1$ with overwhelming probability. As the conditional probabilities overlap between 200 ms and 600 ms, such observations do not help much in reconstructing the network, as various minimum path lengths are almost equally likely. For delay differences higher than 600 ms the certainty rises again. However, this is only because of the limited considered shortest path length of 4 for this calculation. Larger shortest path lengths result in conditional probabilities similar to $C_{min} = 4$ but slightly shifted, similar to the small difference between the conditional probabilities for $C_{min} = 3$ and $C_{min} = 4$. We will exploit the fact that small delays cause a higher certainty in the real-world validation in Section 6.3.2.

6.3.2 Experimental Validation in the Bitcoin P2P Network

We already showed that the proposed timing analysis method is feasible in theory and simulation under idealized conditions. In order to analyze its real-world feasibility, we now apply the proposed method to the Bitcoin P2P network. We will first describe our parametrization of the propagation delay model, then explain the used experimental setup, and finally present and discuss our results.

The propagation delay model is the core of the presented topology inference method, as it provides the mapping between observed time differences and hop count. The presented analytical model assumes a random graph model, however, in Section 5.3 we have seen that the node degree distribution of the Bitcoin P2P network follows a power law. Hence, the analytical delay model cannot be used to obtain the required probabilities $P(D = \delta|C_{min} = l)$ for the real network. Instead, we simulated the

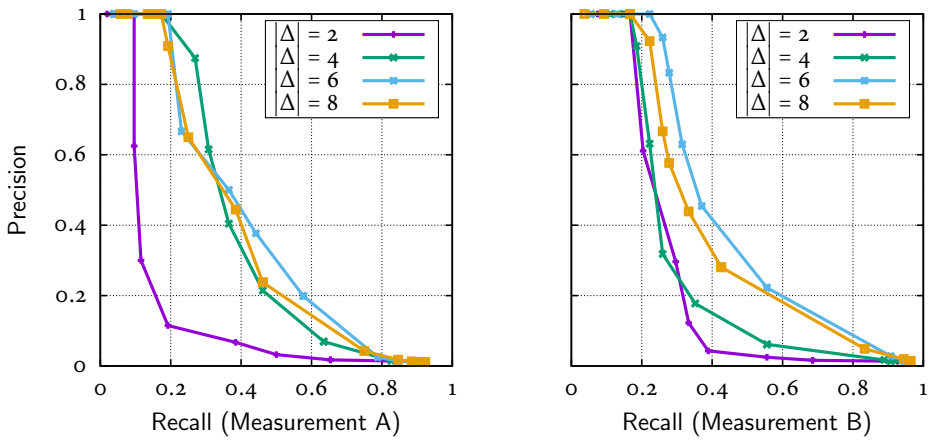

Figure 6.12: Precision vs. recall of two estimations for measurements performed on Jan 26th (A) and Jan 28th (B), 2016 for varying number of observations $|\Delta|$ [NAH16].

transaction propagation in the Bitcoin network using the presented simulation model, and measured the resulting delays for each hop distance. Based on this measurement, the delay distribution can be approximated. We emphasize the importance of a validated simulation model for this approach (cf. Figure 5.6).

In order to carry out an experimental ground truth validation, two off-site peers running *bitcoind 0.11.0* with ~ 50 neighboring peers each were used to create transactions and publish them to the Bitcoin network. As we operate these peers, we know their direct neighbors and can thus compare the inferred connections against the actually existing connections. Our monitor peers, which are connected to all reachable Bitcoin peers, observed the propagation of transactions through the network.

It turned out that the maximum likelihood estimator presented in Section 6.3.1 does not deliver satisfying results because of a large number of very long delays, even for directly connected peers. These large delays increase the likeliness of longer shortest path length, although there were a few observations that indicate a direct connection between a certain peer and the originating peer. As the high certainty of observations with small delays has already been pointed out (cf. Figure 6.11), we now use an estimation that focuses on the smallest observed delay: assuming the set of time differences for a pair of peers Δ_{v_1,v_2} contains n observations, an edge between the two peers is detected, iff the theoretical probability (i.e., the probability as derived from the model) that all n observations are larger than the smallest observation δ_{min} is higher than a certain threshold s ($P(D > \delta_{min}|C_{min} = 1)^n > s$). The threshold represents the sensitivity of the estimation and affects the false positive and negative rates.

Figure 6.12 shows the resulting precision and recall depending on the chosen sensitivity and the number of observations for two different measurements. Each data point corresponds to one setting of the sensitivity threshold s. Depending on the analysis' goal, the threshold can be configured achieve either a higher recall or a higher precision. An increasing number of observations also increases the quality of the estimation up to about 6 estimations, where no further improvement can be

seen. With the appropriate sensitivity, the estimation can achieve a recall of 40 % while maintaining a precision of also about 40 %.

Precision and recall can be combined into the F_1-Score, which is a common measure of accuracy and is calculated as $F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. A perfect predictor results in $F_1 = 1$, whereas worse predictors result in smaller F_1-Scores. The results of our experiments correspond to an F_1-Score of 0.4, which is substantially lower than the theoretically achievable scores, however, much better than simply guessing the connections of a peer, which would result in an F_1-Score of 0.0125. Furthermore, our data shows that 44 % of peers that were online for at least one hour publish at least 5 transactions per day, making a passive topology inference possible. For the remaining 56 % of peers, an adversary has to actively insert transactions similar to the approaches presented in Subsections 6.1 and 6.2.

Although the topology inference quality is much lower in the real-world validation than in the simulation-based validation, the results still show the feasibility of the presented method.

6.3.3 Countermeasure: Trickling

A common countermeasure against timing analysis is deliberately delaying the forwarding of messages (*trickling*) instead of instantaneously rebroadcasting all messages. This countermeasure is implemented in Bitcoin. The idea is to improve anonymity by making it harder to identify the originator of a transactions, and to impede topology inference by increasing the overlap in the conditional delay distributions, which reduces the certainty and hinders the adversary from reconstructing the network. However, this also increases the overall propagation delay in the network, which is not desirable for most applications, for example Bitcoin, where reaching consistency is the main purpose of the network.

Analysis of Trickling Delay Distributions

The existence of general tradeoffs between the objectives DoS resistance, anonymity, topology hiding, performance, and cost of participation have already been discussed in Chapter 3. We will now apply the presented propagation delay model and the timing analysis technique in order to quantify the tradeoff between topology hiding in terms of precision and recall, and performance in terms of *consistency delay*, i.e., the delay until a message has been flooded through a certain share of the network. We assume that the analyzing adversary is aware of the fact that trickling is performed and how it is parametrized.

Figure 6.13 illustrates the effect of applying the trickling countermeasure on the consistency delay and the precision and recall, given as the F_1-Score, of an adversary for our exemplary scenario. In the exemplary scenario, trickling is performed by randomly delaying the forwarding of messages for a certain length of time according to a) a uniform distribution and b) an exponential distribution. Both distributions were parametrized with a set of mean values: at $\mu = 0$, effectively no trickling happens, therefore, the result corresponds to what has been shown in Figure 6.10. With increasing mean values, both the time until information is propagated to 90 % of peers, as

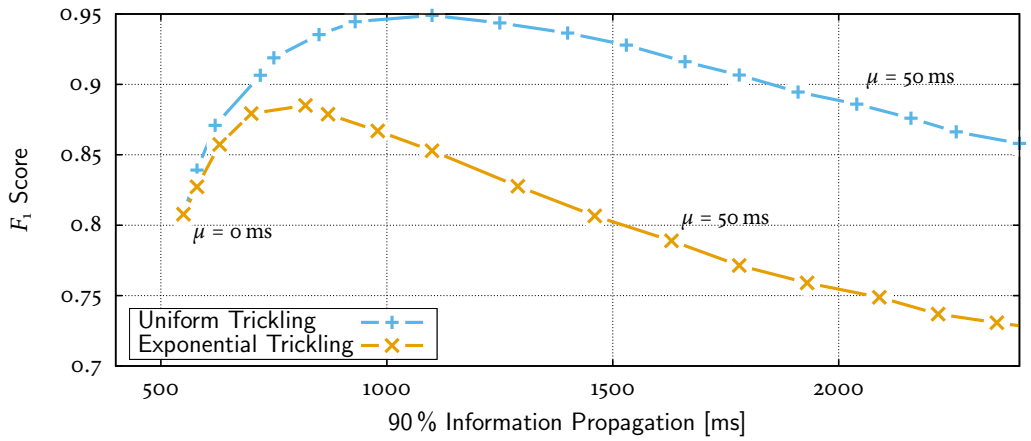

Figure 6.13: Tradeoff between low consistency delay and topology inference resistance when applying *trickling*. Trickling is performed using *a)* a uniform distribution of varying size, and *b)* an exponential distribution with varying mean. The x-Axis shows the overall delay until 90 % of peers received the propagated message. F_1-Score after 4 observations [NAH16].

well as the F_1-Score increases. Only for higher delays, the F_1-Score starts to decline.

Although one might expect trickling to always have a positive effect on topology hiding, the results show that trickling, if inappropriately parametrized, can actually reduce the resistance to topology inference, i.e., it can improve an adversary's precision and recall. This is caused by trickling's negative effect on propagation speed. Trickling's goal is to increase the overlap of the transition probability distributions. On the one hand, trickling broadens the shape of the conditional delay distribution, on the other hand it also increases the difference between the mean of the different distributions. For example, a constant trickling distribution that delays all packets by one second makes it much easier for an adversary to guess the packet's hop-count, as the constant delay only increases the gap between the different conditional probabilities, but does not broaden each distribution's shape. Figure 6.13 also shows that trickling according to an exponential distribution can increase the resistance against topology inference if properly parametrized, whereas trickling with a uniform distribution has a negative effect for the parameters considered.

Please note that although trickling may be detrimental for preventing topology inference, it also can have positive effects on the general timing analysis resistance that were not discussed here. For example, trickling makes it substantially harder for an adversary to identify the originator of a message in the network, which improves anonymity in the network (cf. Chapter 7) and also makes timing based topology inference harder, as the adversary has to actively create transactions.

Optimal Trickling

The example shown in Figure 6.13 indicates that delaying messages according to an exponential delay distribution is Pareto-better (i.e., results in worse topology inference quality at the same performance) than delaying messages according to a uniform

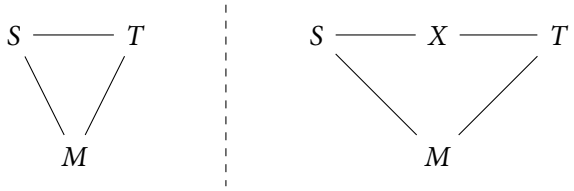

Figure 6.14: Considered scenario: The adversary M wants to infer whether S and T are directly connected (left side), or whether S and T are not directly connected (right side) [NH18].

delay distribution. This raises the question what an *optimal trickling delay distribution* is. In this subsubsection we will address this question by numerically optimizing the trickling delay distribution for a simplified scenario.

The content presented in this subsubsection has been previously published in [NH18].

Considered Scenario & Adversary Model We consider the simplified scenario sketched in Figure 6.14: An adversary M is connected to two peers S and T and wants to infer, whether S and T are directly connected or not. The adversary creates a message (e.g., a transaction) and sends it to S so that S receives the message at time 0. The adversary then waits and measures the duration δ until T sends the message to M. δ is therefore the delay from S receiving the message until T sends the message to M.

Consider the case that the latency between any two peers is one time unit, i.e., the transmission of a message over one link takes one time unit, and each peer immediately rebroadcasts each message to its neighbors. Then, the adversary knows that if $\delta = 2$ then S and T are directly connected, if $\delta = 3$ then S and T are not directly connected. In reality, the latency between two peers is not constant but follows a probability distribution. We assume that the latency between any two peers follows the same distribution $\lambda(t)$, which is known to the adversary.

We will now adapt the maximum likelihood estimator presented in Subsection 6.3.1 that can be used by the adversary to infer whether a direct connection between two peers exists. We consider a simple relay delay strategy that delays every message independently using a given delay function d. All notation in this section is discrete, hence d is a probability mass function (PMF) that defines the probability that a message is delayed by a certain duration. The duration is a discretized representation of time, e.g., time slots of millisecond precision. Let $f * g$ denote the convolution of the (discrete) functions f and g, and let f^{*n} denote the n-th convolution power of a function f. Let C be the random variable modeling the path length between S and T. $C = 1$ if S and T are directly connected, $C = 2$ if there is one hop between S and T. The resulting distribution for the overall delay δ equals time t conditional to the path length C is then given by

$$P(\delta = t | C = c) = (\lambda^{*c} * d^{*(c+1)})(t). \tag{6.2}$$

For example, if S and T are directly connected ($C = 1$), the message is delayed by one link latency (the link between S and T) and two relay delays according to d (at peers S and T).

The probability that the distance between S and T is c, given an observed time difference of t, is given by

$$P(C = c|\delta = t) = \frac{P(\delta = t|C = c) \cdot P(C = c)}{P(\delta = t)}. \tag{6.3}$$

$P(\delta = t|C = c)$ can be calculated using equation (6.2), $P(\delta = t)$ can be calculated using the law of total probability, $P(C = c)$ is assumed to be known to the adversary based on statistic properties of the network. The MLE maximizes $P(C = c|\delta = t)$, i.e., the adversary guesses the path length C that is most likely based on the observation δ.

Optimization Methodology Based on the adversary model we will now derive a delay function d that maximizes the expected error of the adversary, i.e., which makes topology inference as hard as possible. The expected error e_d of the guess of the adversary depends on the delay function d, and can be calculated using equation (6.3) as

$$e_d = \sum_t [P(\delta = t) \cdot (1 - \max_c P(C = c|\delta = t))]. \tag{6.4}$$

The expected error is the objective function that should be maximized. The variable in the optimization problem is not a scalar, but the delay function d. The optimization has to ensure that d is a PMF (i.e., $d(t) \geq 0 \forall t, \sum_t d(t) = 1$). Furthermore, we want to limit the expected value $\mathbb{E}(d)$ of d to be less than some constant μ. The choice of the parameter μ reflects performance constraints in the system: A small choice of μ ensures fast message propagation, a large μ allows for slower message propagation. The resulting optimization problem is

$$\begin{aligned} \underset{d}{\text{maximize}} \quad & e_d \\ \text{subject to} \quad & \mathbb{E}(d) < \mu \\ & d \text{ is a PMF.} \end{aligned} \tag{6.5}$$

Because d is a discrete function, Equation (6.5) is a multidimensional optimization problem, where each time step of the delay function d is one dimension (i.e., one free variable) of the optimization problem. The optimization problem is also constrained ($\mathbb{E}(d) < \mu$ and d is a PMF). However, the optimization problem with both constraints can be transformed into an unconstrained optimization problem (e.g., using gradient projection [Ros60]). The optimal solution to the transformed problem can be approximated using common software for optimization (e.g., we used the BFGS search algorithm implemented in the Dlib toolkit [Kin09]).

Results We assume a scenario with a fixed latency of one time unit between all directly connected peers ($\lambda(1) = 1, \forall t \neq 1 : \lambda(t) = 0$), equal a priori probabilities ($P(C = 1) = P(C = 2) = 0.5$), and a maximum expected value of 10. The top part of Figure 6.15 shows the approximated optimal delay function $\hat{d}(t)$. For a delay of 0 the probability peaks at around 0.42 and rapidly declines to less than 0.02, where

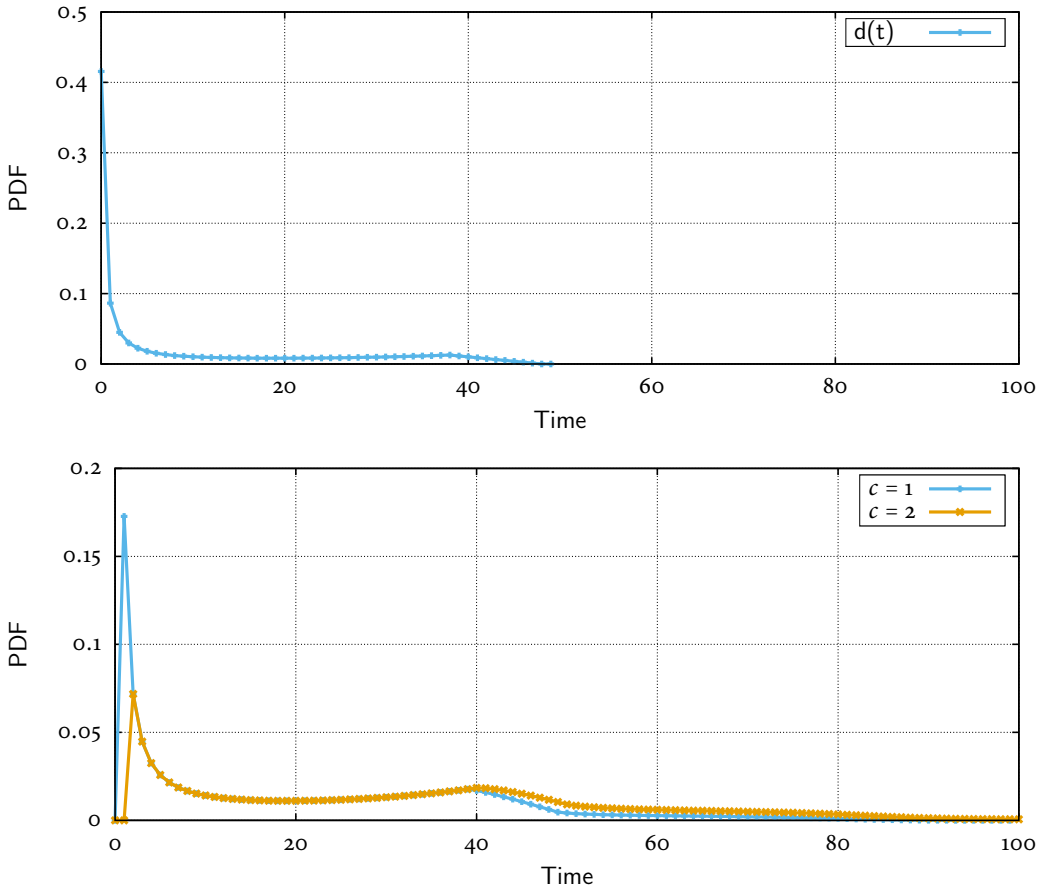

Figure 6.15: Top: Optimal $\hat{d}$. Bottom: Resulting $P(\delta = t|C = c)$ for $c \in \{1, 2\}$. Parameters: $\mu = 10$, $\lambda(1) = 1$ (0 else), $P(C = 1) = P(C = 2) = 0.5$ [NH18].

it stays until t=39. The expected value of $\hat{d}$ is 10, $e_{\hat{d}}$ is 0.41. Please note that $\hat{d}$ does not decrease monotonically over time and $\hat{d}$ does not resemble any common PMF (e.g., the PMF of a binomial distribution).

The bottom part of Figure 6.15 shows $P(\delta = t|C = c)$ for the same scenario with the optimal function $\hat{d}$ for $c \in \{1, 2\}$. $P(\delta = t|C = c)$ is used by the MLE to derive the probability for $C = 1$ and $C = 2$, based on the observed time difference δ. We can see that $P(\delta = t|C = 1)$ and $P(\delta = t|C = 2)$ are exactly congruent between $\delta = 2$ and $\delta = 19$. This implies that an observation within that range is completely useless for the adversary. However, if the adversary observes $\delta = 1$, he can be sure that $C = 1$ because $P(\delta = 1|C = 2) = 0$ and $P(\delta = 1|C = 1) > 0$. If the adversary observes $\delta \geq 40$, he learns that $C = 2$ is more likely than $C = 1$ because $P(\delta = t|C = 2) > P(\delta = t|C = 1)$.

Especially the observation of $\delta = 1$ is valuable for topology inference, because it allows the definite conclusion that both remote peers are directly connected. While $P(\delta = 1|C = 1)$ is only around 17 % for one single observation, the probability that at least one out of ten observations for two directly connected peers is $\delta = 1$, is already at 85 %. This property makes $\hat{d}$ a non-optimal delay function when the adversary

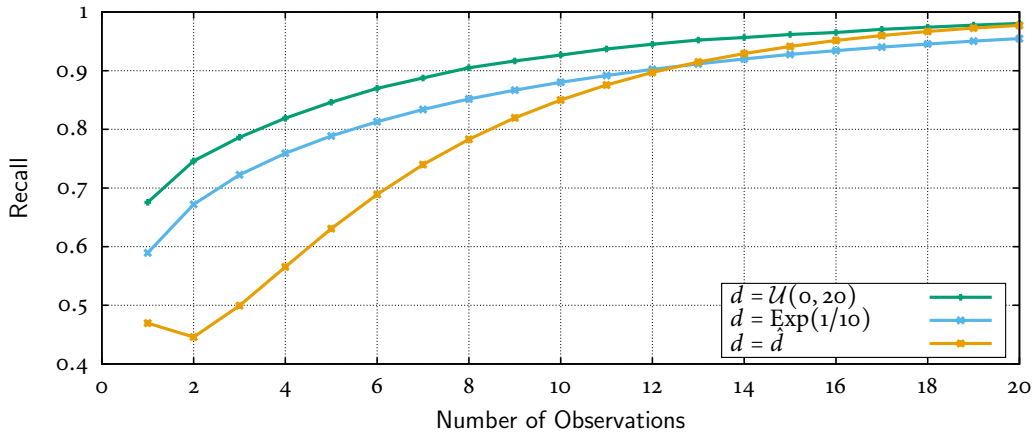

Figure 6.16: Recall depending on the number of observations for $d \in \{\hat{d}, \text{Exp}(1/10), \mathcal{U}(0, 20)\}$ [NH18].

combines multiple observations.

Figure 6.16 shows the recall (i.e., the probability that an existing connection is correctly inferred) depending on the number of observations for d being $\hat{d}$, a uniform distribution, and an exponential distribution (all with a mean of 10), obtained by simulation. Although $\hat{d}$ results in the lowest recall for a small number of observations, for large numbers of observations other distributions are better for topology hiding.

All shown delay functions result in a recall of more than 95 % for 20 observations. However, these values are a result of the strong adversary model with perfect knowledge of all network properties (e.g., latency, node degree distribution). Imperfect estimation of these properties causes a decline in inference quality.

Although the considered scenario is simplified, it shows the general possibility of optimizing the delay function and, thereby, calculating the *optimal* delay function according to the given constraints. However, the considered scenario is so simplified that it neither accounts for realistic latency distributions nor for realistic network topologies. Furthermore, the optimization problem only covers the tradeoff between performance and topology hiding, but does not incorporate the requirement anonymity (cf. Chapter 7).

6.3.4 Discussion

In this section we presented and analyzed a timing-based topology inference method. The real-world validation in the Bitcoin network shows that the proposed method could be used to infer network links at a substantial (~ 40%) recall and precision at the time of conducting the experiments in January 2016. We also showed that randomly delaying the forwarding of messages can, if inappropriately parametrized, actually reduce the resistance to timing-based topology inference. Finally, we show how such a parametrization can be optimized regarding performance and topology hiding.

As the flooding process is inherently observable in permissionless blockchains, delaying the forwarding of messages is the only possible countermeasure. While

transactions were hardly delayed in 2016, the transaction delay has increased substantially since then (cf. Figure 4.12), rendering the discussed topology inference method much more difficult to execute. On the other hand, there were discussions[15] in the Bitcoin Cash community to reduce transaction propagation delay in order to accelerate zero-confirmation payments.

Finally, we would like to point out the similarity between timing analysis for topology inference in flooding P2P networks on the one hand and timing analysis for deanonymization in anonymous communication (AC) networks: A common adversary model in the analysis of AC networks is the global-passive adversary (GPA, e.g. [MD05]), which is able to observe the inter-packet intervals on all links between nodes of the network. This adversary model is similar to the one used in this work, as an adversary that participates in a flooding network receives all messages from all of its neighbors and, therefore, can reconstruct message flows. One difference to the assumptions made in our work is that in AC networks, an adversary is not able to link messages received by a peer to messages sent by that peer, as these message appear indistinguishable to the adversary because of encryption. Because timing analysis attacks on AC networks like Tor have been extensively studied (e.g., [DMMK17]), knowledge from this area of research might also be applicable in the area of topology inference in permissionless blockchains.

6.4 Exploiting Peer Discovery for Topology Inference

As discussed in Chapter 3, some form of peer discovery is required for every peer-to-peer network in order for peers to find other peers to connect to. In this section, we explore how the design of the in-band peer discovery strategy affects the requirements topology hiding, DoS resistance, and performance of a network, and demonstrate a method for quantitatively assessing the quality of a peer discovery strategy. In contrast to the previous sections, this analysis does not aim at any specific system (e.g., Bitcoin), and no real-world validation is performed. Instead, we use simulations to quantitatively asses the inherent tradeoffs in the design space. The content presented in this section has been previously published in [NH18].

6.4.1 Peer Discovery: Requirements & Tradeoffs

It is important that a client is able to quickly establish outgoing connections, not only from a performance perspective but also for DoS resistance. In the event of an eclipse attack a client should be able to react on the loss of connections caused by the attack by the establishment of new connections. Metrics that reflect this ability are the total number of IP addresses in a client's address list and the number of reachable addresses in a client's address list. The total number of reachable addresses gives an upper bound on the number of successful connections a client can establish, the share of reachable addresses indicates the probability of successfully establishing a connection per connection attempt, assuming the client tries to connect to randomly chosen addresses.

[15] https://twitter.com/PeterRizun/status/980224151242784768

Another requirement that is affected by the peer discovery strategy is topology hiding as an adversary can infer connections between peers based on the address messages peers send to their neighbors [MLP+15]. Metrics that indicate the success of an adversary are precision and recall for the classification problem of whether a direct connection between two peers exists.

Intuitively we expect that the choice of parameters of the peer discovery strategy has an oppositional effect on the two requirements. For instance, a configuration that sends a large number of IP addresses at short time intervals with precise timestamps of connected IP addresses will result in a good DoS resistance, but will also make it easy for adversaries to infer the network topology. We will quantitatively analyze the tradeoff between these two requirements in the next sections.

Finally, adversaries should be unable to eclipse peers by filling their address list with IP addresses under the adversary's control and making the victim peer connect exclusively to attacker's peers [HKZG15]. We will discuss this requirement in Section 6.4.6.

6.4.2 Peer Discovery Strategy Description

We analyze a basic in-band peer discovery strategy that periodically exchanges reachable IP addresses between connected peers.

Every client maintains an *address list* l containing tuples consisting of an IP address a_i and an associated timestamp t_i ($l = \{(a_1, t_1), (a_2, t_2), ...\}$). Every δ_s seconds a client sends an *address message* containing n randomly (uniform) selected entries of its address list to each neighbor. On reception of such a message from a neighbor a client updates its own address list: new addresses (and their timestamp) are added to the list, and the timestamp of known addresses is updated if a newer timestamp than the one stored is received. When a new connection is established to or from a client, the client adds the foreign IP address to its address list l and randomly selects a timestamp for that IP address from a uniform distribution $\mathcal{U}[t - \delta_d, t]$ where t is the current time and δ_d is a configured value. When the timestamp of a connected peer becomes smaller than $t - \delta_d$, a new timestamp is set according to the same uniform distribution ($\mathcal{U}[t - \delta_d, t]$). This strategy ensures that the timestamps of all connected peers are always newer than the current time minus δ_d, hence δ_d represents the maximum age of connected IP addresses in a peer's address list. Finally, addresses with timestamps smaller than $t - \delta_x$ are removed from a client's address list, with δ_x being a configurable parameter.

The described strategy is very simple and can be configured using only the parameters $n, \delta_s, \delta_d, \delta_x$. However, there are many more changes possible, e.g., the timestamp of connections could follow other probability distributions than the used uniform distribution, the subset of addresses to be sent to neighbors could be biased based on the timestamp, or the number of addresses to be sent could depend on the total number of entries in a client's address list or the connection duration. However, we will limit our analysis to the described strategy with its parameters and leave a more detailed assessment as future work.

6.4.3 Adversary Model

The adversary wants to infer the topology of the network based on information leaked by the peer discovery mechanism. For now we assume a passive monitor adversary that establishes connections to peers and receives the announced addresses from its neighbors. A discussion of other adversary models is made in Section 6.4.6. We assume that the adversary knows all chosen parameters as well as all required parameters of the network (e.g., node degree distribution). Consider the example of an adversary that wants to know whether two peers, p_1 and p_2, are directly connected. Let us consider the case that the adversary is connected to p_1 only. One observation $o \in O$ by the adversary is the reception of one address message from p_1 containing a subset of p_1's address list. One observation can be either the age of the IP address of p_2 or the fact that the IP address of p_2 is not contained in the sent list i.e., $o \in \mathbf{R}^+ \cup \{\perp\}$ ($o = \perp$ implying that the IP address of p_2 is not contained in the sent list).

Let C be the random variable modeling the existence of a connection between two peers (i.e., $C = 1$ if both peers are directly connected, $C = 0$ otherwise). The maximum likelihood estimator (MLE) for a set of observations O maximizes the likelihood function

$$L(C = c|O) = P(C = c) \cdot \prod_{o \in O} P(o|C = c)$$

for $c \in \{0, 1\}$. In order to utilize the MLE, an adversary requires knowledge of the probability distributions $P(C = c)$ (i.e., the a-priori probabilities of two peers being connected) and $P(o|C = c)$ (i.e., the probability of making a specific observation o conditional to both peers being connected or not connected, respectively). Combining knowledge about the client source code with statistic properties of the network into a simulation model allows approximation of both probability distributions for real-world systems.[16]

6.4.4 Methodology

In order to analyze the discussed peer discovery strategy, we implemented a model of a P2P network as a discrete event simulation (cf. Chapter 5). The simulation model has three types of events: a peer joins the network, a peer leaves the network, and a peer sends an address message to a neighbor. The churn of the network (i.e., join and leave events) was taken from a real-world measurement on the Bitcoin network: our monitor peer establishes connections to all reachable peers on the Bitcoin network. Every new connection to the monitor peer translates to a *join* event, every disconnect translates to a *leave* event. The simulation was performed with a one week snapshot

[16] $P(o = \perp|C = 1)$ is calculated by dividing the parameter n (the number of IP addresses a client sends) by the total number of addresses in the client's address list. Both values can be approximated by the adversary from the client source code. $P(o = \perp|C = 0)$ is calculated as $P(o = \perp|C = 1)$ multiplied by the probability that a client has the IP address in question in its list. An adversary with knowledge of the client source code and basic statistic properties of the network (e.g., number of nodes, churn) can approximate that probability by simulation. That way, an adversary can also derive $P(o|C = 1)$ and $P(o|C = 0)$ for $o \neq \perp$.

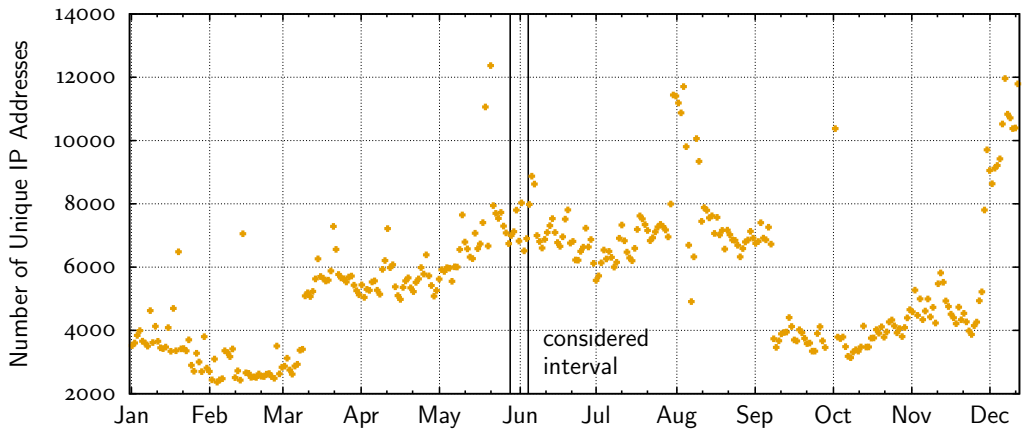

Figure 6.17: Measured number of unique IP addresses of the Bitcoin P2P network to which connections were established per day during the year 2017 [NH18].

from May 29th, 2017 until June 5th, 2017. During that period, our monitor peer established connections to 35,000 unique IP addresses. On average around 9,000 peers were concurrently reachable during that period, which is also the size of the simulated network. All simulations were performed for the duration of one simulated week.

Although the use of a specific snapshot from the Bitcoin network for parametrization of the simulation reduces generality of the obtained results, the selected snapshot is a representative sample of the Bitcoin network in 2017: Figure 6.17 shows the number of unique IP addresses to which connections were established per day during the year 2017. While there were anomalous events during that year (e.g., several thousand Sybil peers on August 1st), no such event occurred during the considered time frame. Furthermore, the observed connection duration distribution during that time frame is consistent with the distributions observed during most of the year. Finally, other measurements of the Bitcoin network[17] are in correspondence to our measurements. We will discuss whether the results can be generalized to other networks in Section 6.4.6.

When a peer joins the network in the simulation, it establishes 8 outgoing connections to randomly selected peers. When a peer leaves the network, other peers that established an outgoing connection to the leaving peer establish new outgoing connections to other randomly selected peers so that the total number of outgoing connections remains 8. This behavior matches the behavior of the Bitcoin client.

The *address send* event implements the described strategy. The simulation does not account for latencies between peers because these are typically in the milliseconds range and, therefore, irrelevant for our analysis. We also ignored unreachable peers during the simulation but will discuss the effect of unreachable peers on the results later. The simulation also allows us to directly observe all probability distributions (cf. Sec. 6.4.3) required for the adversary. We simulate the adversary by providing

[17] Publicly available data sources include `https://bitnodes.earn.com/` and `http://bitcoinstats.com/network/propagation/`.

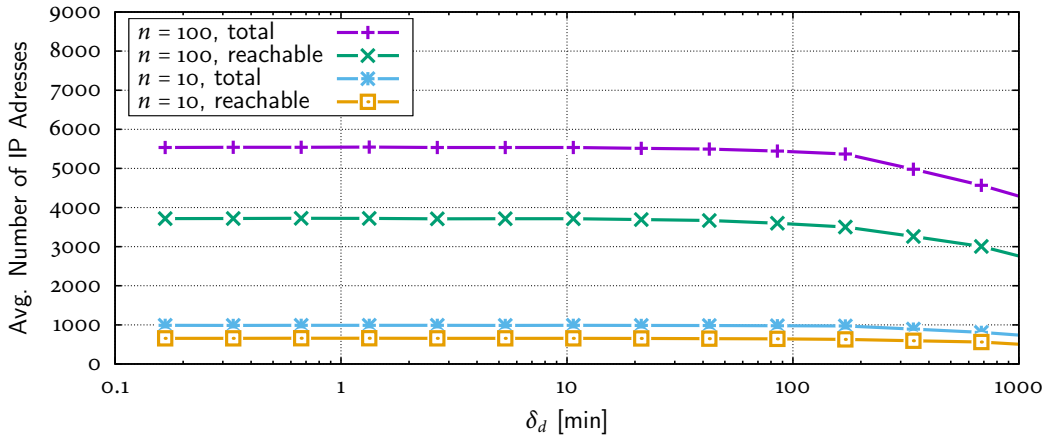

Figure 6.18: Average number of IP addresses in all client's address lists [NH18].

all these probability distributions and observations to the simulated adversary and letting the adversary guess whether peers are connected or not. This implies that our adversary has perfect knowledge, which is almost impossible to achieve in reality.

6.4.5 Results

We will now first discuss simulation results regarding DoS resistance and then discuss results regarding topology hiding. We fixed the address send interval δ_s to one hour and the address deletion age δ_x to 24 hours for all simulation runs. Parameters that were varied were the number of addresses to send n and the maximum age of connected IP addresses δ_d.

Figure 6.18 shows the average number of IP addresses (total and reachable only) in all client's address lists for $n \in \{10, 100\}$ depending on δ_d. For $n = 100$ the average total number of IP addresses is around 5,500 for $\delta_d < 100$ minutes. The average number of reachable IP addresses is around 3,700 for $\delta_d < 100$ minutes. For very large choices of δ_d, the total and reachable number of IP addresses decline. With $n = 10$ a similar behavior can be seen, although the overall numbers are much smaller (less then 1,000).

The results indicate that the discussed peer discovery strategy works well over a wide range of parameter choices. Even when configured to sending only 10 addresses per hour per neighbor ($n = 10$), the address list of peers still contains around 600 reachable IP addresses on average. Furthermore, the effect of the choice of δ_d is negligible for $\delta_d < 100$ minutes. Please note that the given values are averages over all peers after one week of simulated time. Peers that remain in the network for a long time and have established many connections have more entries in their address list than peers that joined the network just a short time ago. Furthermore, the session length distribution of peers affects the share of reachable IP addresses: a large number of short living peers would increase the total number of IP addresses but would contribute less to the number of reachable IP addresses.

Based on these results we chose three parameter sets for analysis of a topology inference attack. We fixed the number of addresses to send to $n = 100$ and simulated

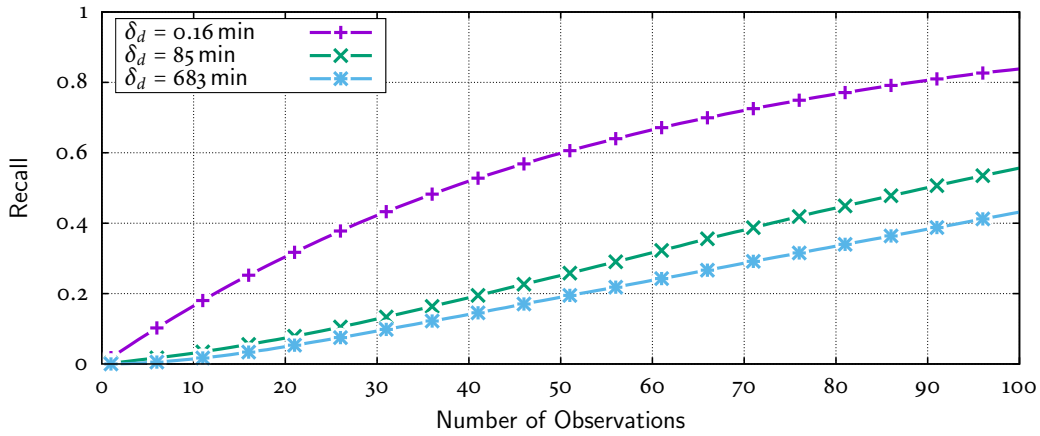

Figure 6.19: Average recall depending on the number of observations $|O|$ [NH18].

a topology inference attack for $\delta_d \in \{0.16\,\text{min}, 85\,\text{min}, 683\,\text{min}\}$. $\delta_d = 0.16\,\text{min}$ and $\delta_d = 683\,\text{min}$ represent extreme choices (corresponding to the outermost points in Figure 6.18), $\delta_d = 85\,\text{min}$ corresponds to a choice where, based on the results from Figure 6.18, no significant deterioration of DoS resistance can be expected.

Figure 6.19 shows the expected recall depending on the number of observations $|O|$. Recall is defined as the quotient of true positives and relevant elements, i.e., the share of existing connections the adversary infers. As expected, the recall rises in the number of observations and also rises with decreasing δ_d.

Figure 6.20 shows the expected precision depending on the number of observations $|O|$. Precision is defined as the quotient of true positives and all inferred elements, i.e., the share of inferred connections that actually exist. For $\delta_d = 0.16\,\text{min}$ the precision is close to 1 regardless of the number of observations. For the other δ_d, the precision increases with the number of observations.

6.4.6 Discussion

We will first discuss the implications of the shown results before we address various aspects of the adversary model and the discussed strategy.

Results

Assume a configuration with $\delta_d = 85\,\text{min}$ and $n = 100$. Our results then show that an adversary that is able to obtain 100 observations from a certain peer is able to identify 55 % of the neighbors of that peer with very high precision (i.e., almost no false positives). With the setting of δ_s to one hour, an adversary that only connects to that peer would have to wait 100 hours. A monitor adversary that connects to all reachable peers would only need to wait 50 hours to obtain 100 observations for each pair of peers because he receives one address message per hour from every peer. By using more than one monitor peer an adversary can easily reduce the required time to collect the desired number of address messages.

One important requirement for the adversary is that the network topology does

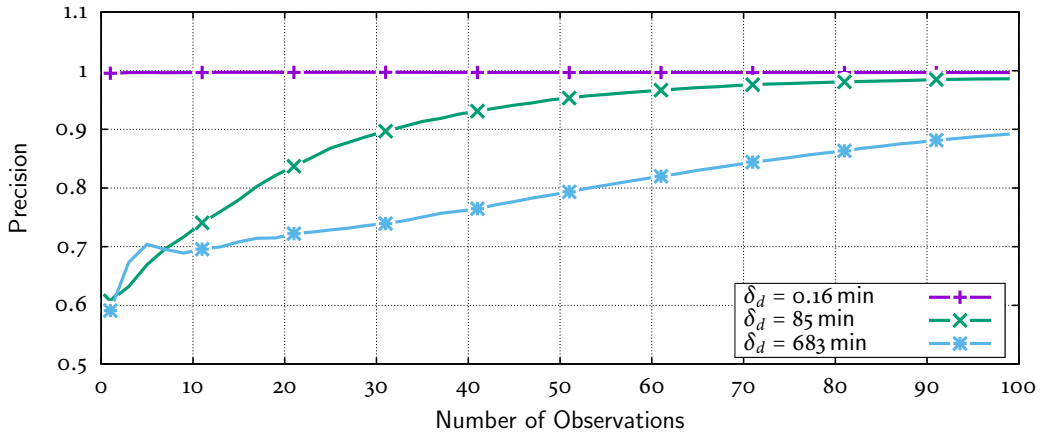

Figure 6.20: Average precision depending on the number of observations $|O|$ [NH18].

not change while making observations. Measurements on the Bitcoin network show that roughly 40 % of all connections exist for periods longer than one week (cf. Section 4.2.1). In these cases even an adversary with one monitor peer would be able to collect enough address messages to identify a substantial share of connections of peers.[18]

Adversary Model

The adversary model we considered in this analysis is in some aspects stronger than what we expect a real adversary to be (e.g., perfect knowledge of all relevant parameters and distributions), however, there are other aspects where a stronger adversary model might be appropriate. First, we assumed the adversary to be passive. In reality, an adversary can easily actively transmit its own address messages to other peers.

A very simple attack is to flood unreachable IP addresses to other peers so that the share of reachable IP addresses decreases. The number of IP addresses an adversary can send per connection is limited by the choice of n/δ_s and the expiration duration δ_x. For instance, with δ_s and δ_x as before and $n = 100$, an adversary can send up to 2400 unreachable IP addresses per connection in 24 hours. Even with 100 adversarial connections, the share of reachable IP addresses would still be around 1.5 %, implying that a connection can be established within a few minutes or less. Furthermore, the required memory to store all addresses would be less than 10 MB. Hence, this attack is not viable for most scenarios.

A known attack is to eclipse peers by filling their address list with IP addresses under the adversary's control and making the victim peer connect exclusively to attacker's peers [HKZG15]. If a client selects IP addresses from its address list randomly for establishing connections, the adversary has no other option than to announce the IP addresses of his peers and hope that the victim peer connects exclusively to his peers. In the analyzed case, an adversary has to spawn more than 40,000 peers and announce their IP addresses in order to eclipse a peer with 8 outgoing connections

[18] Although some parameters were chosen in accordance with those in Bitcoin, our results are not directly applicable to the Bitcoin network because neither the discussed peer discovery strategy nor the network topology matches the ones in Bitcoin.

with a probability of 50 %.[19]

Furthermore, the considered MLE is only optimal assuming independence between several observations. Adversaries could exploit further information such as the differences in timestamps of certain IP addresses between several observations. Also, partial knowledge of the network topology might enable an adversary to not only exploit information received from peers p_1 and p_2 in order to infer whether a connection between p_1 and p_2 exists, but also to facilitate information sent by other peers (e.g., the neighbors of p_1 and p_2).

Peer Discovery Strategy

Although the analyzed peer discovery strategy is very simple, the results indicate that an adequate parametrization of the strategy could already satisfy the considered requirements to a reasonable extend. In scenarios with stronger requirements, several changes to the strategy are possible: In order to prevent too many connections to peers from small IP ranges or the same AS, simple checks can be implemented after the random selection of IP addresses from the address list. Selecting IP addresses biased towards new addresses might improve performance, however, it can be easily exploited by adversaries to increase chances of connections to its own peers. Therefore, a random selection is preferable. A possibility to improve performance and DoS resistance at the cost of higher bandwidth cost proposed in [HKZG15] and implemented in Bitcoin[20] is to continuously check whether IP addresses are reachable. This drastically reduces the share of unreachable IP addresses.

6.5 Topology Inference - Discussion

In this chapter we presented and analyzed four different topology inference methods. Figure 6.21 summarizes the results for all methods. The first method targets the accumulation of transactions in the Bitcoin client *bitcoind*, however, the lack of validation possibilities makes an assessment of the real-world inference quality challenging. The second analyzed method exploits the handling of double spending transactions and resulted in an inference quality sufficient for academic and adversarial purposes. The third method is based solely on passively observing the timing of message propagation. It resulted in a reasonable inference quality in 2016, but is hardly feasible as of 2018 because of deliberately increased and randomized transaction propagation delays. The fourth method targets the peer discovery mechanism, which turned out to be exploitable only for certain parametrizations.

We emphasize that topology inference is usually only an intermediate goal (cf. Chapter 3). Therefore, whether the quality of an inference method is sufficient depends on the actual goal of the adversary. Topology inference can be seen as a dual use technology, which can be used for research purposes as well as for adversarial purposes. First,

[19] Calculated as: (number of adversarial peers / total number of reachable IP addresses in I)^8 = 0.5: $40000/43700^8 \approx 0.49$.

[20] `https://github.com/bitcoin/bitcoin/blob/5114f8113627791b871c88998bd/src/net.cpp/#L1756`

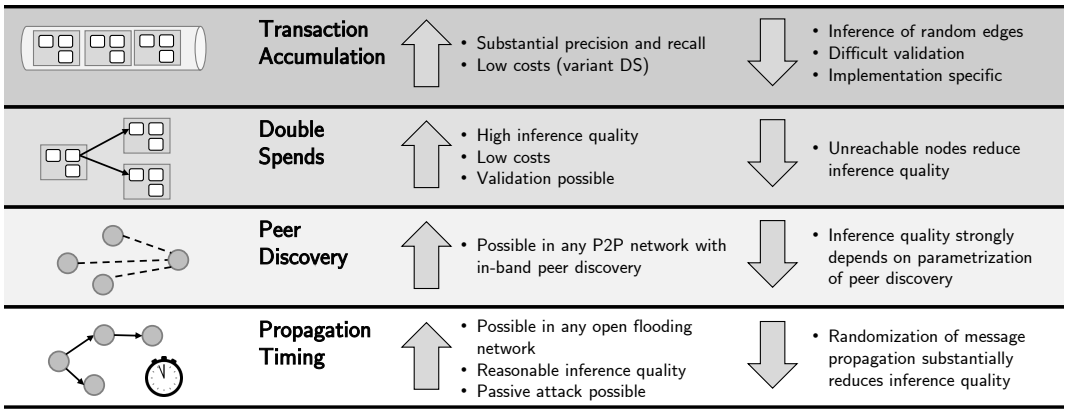

Figure 6.21: Overview of considered topology inference methods. Advantages and drawbacks of each method are given from the perspective of *enabling* topology inference.

it can be used as a basis for a deanonymization of users [FV17]. Further research is required in order to asses, whether the topology inference quality provided by the analyzed methods is sufficient for deanonymization.[21] Secondly, topology inference can be used as a basis for eclipsing attacks. While it is hard to estimate the capabilities and motivation of an adversary wishing to perform an eclipsing attack, the quality of the analyzed topology inference methods seems sufficient to substantially reduce the costs for eclipsing attacks. Finally, topology inference can be used as a basis for research. Using the analyzed methods it is not possible to get a view of the complete network graph, which might be desirable for a precise parametrization of simulation models. However, insights on properties such as the node degree distribution or on single, conspicuous peers can be gained using the analyzed methods.

All analyzed topology inference methods have two general limitations in common. First, connections to unreachable peers cannot be inferred using any approach.[22] Although unreachable peers to not contribute to the P2P network by providing connection slots, they contribute to the overall security because they are hidden to most other peers (except for those peers they are connected to). Secondly, churn changes the network topology over time, limiting the time frame available for gathering information for topology inference.

The fact that the topology of a changing network is harder to infer suggests the idea to deliberately disconnect from neighbors after a certain period and regularly establish new connections to other peers.[23] If topology hiding is considered extremely important in a certain system, clients could apply a notion similar to the *privacy budget* known from differential privacy [Dwo08] to its own connections. Sending

[21] `https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-July/016216.html`

[22] Although peer discovery can leak IP addresses of unreachable peers, those IP addresses could be filtered by checking the reachability of an IP address before announcing it.

[23] This idea has been proposed for structured P2P networks [CKS+06].

messages that allow an adversary to infer that connection decreases the budget of the connection. When the budget is exceeded, the connection is terminated. One drawback of this approach is, however, that an unstable network topology not only increases bandwidth cost, but also reduces DoS resistance against short term Sybil attacks. An adversary that enters the network with a large number of Sybil peers is able to *thin out* connections between honest peers much faster in a changing network, as peers will establish connections to the Sybil peers faster. Therefore, a thorough analysis of all implications of such an approach is required.

Several countermeasures were discussed for all analyzed topology inference methods. All countermeasures have two aspects in common: First, they all *randomize* when actions are performed by a client (e.g., delay of transaction propagation) and how these actions are performed (e.g., accumulated vs. individually). We could show that the parametrization of the random process (i.e., the probability distribution used for transaction trickling) is crucial for the effectiveness of the countermeasure. Secondly, all countermeasures have in common that they make actions initiated by a client independent of what an adversary can influence. This includes making decisions independent of information provided by other, untrusted peers.

A possible way to improve inference quality is to combine the information gathered using several of the discussed methods. For instance, passively acquired timing information might be used to establish guesses, which can be confirmed by active methods with higher precision such as the method exploiting double spends. Furthermore, the method exploiting double spends might be optimized by making use of more monitor peers, by a continuous sending of transactions, or by further combination of double spending inputs. In this dissertation we analyze the feasibility and quality of several topology inference methods, but do not actually apply the methods with the goal of inferring the topology of a complete real-world network. By further improving the discussed methods, an approximation of node degree distribution or the identification of single, well connected peers might be possible. Such knowledge could facilitate further research by improving the available simulation model, hence making simulation-based assessments more accurate. However, such knowledge could also facilitate attacks on the network. Therefore, it can be reasonable to design a network so that the effectiveness of topology inference is limited, although knowledge of the topology is desirable from a scientific perspective.

7
Anonymity

We have identified anonymity as a security objective of permissionless blockchains in Chapter 3. In this chapter we empirically address the question whether observations on the Bitcoin P2P network can be used to link the creators of transactions to IP addresses of Bitcoin peers. The content of this chapter has been previously published in [NH17].

Several previous works suggest that a network-based deanonymization can be possible in certain scenarios and assuming certain adversary models (e.g., [Kam11, KKM14, FV17, BKP14]). However, with users using dynamically assigned IP addresses, operating from clients behind NAT routers or using wallet services, it is not clear whether information obtained by participating in the network and observing the *normal* message flow could be used for the deanonymization of Bitcoin users.

In addition to network-based deanonymization, heuristics for the clustering of Bitcoin addresses based solely on blockchain data have been proposed (e.g., [RH13]), which can enable the linking of several addresses to one user. It was also shown that it can be possible to establish a link between one of a user's addresses and information from additional sources that reveals the user's identity. In the worst case, this knowledge can be used to learn about all financial transactions of an identifiable user.

One fundamental challenge for the analysis of deanonymization approaches is the lack of ground truth data. Therefore, neither for a blockchain based clustering nor for a network-based deanonymization a ground truth validation can be performed.[1] However, both approaches operate on disjoint data (blockchain vs. network data) but aim at computing the same result (addresses controlled by one user). Therefore, a correlation between the results of both approaches can serve as a validation of both approaches.

Figure 7.1 illustrates our approach: First, we apply blockchain based clustering

[1] An exception to this is was presented by Nick [Nic15], who was able to extract ground truth data because of a client implementation bug.

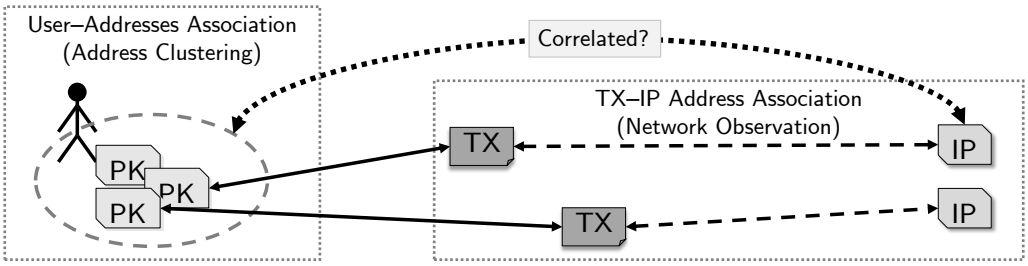

Figure 7.1: High-level overview of the used approach: Addresses are clustered using known heuristics; transactions are assigned to IP addresses based on network observations and to clusters based on their content. We then check whether single clusters are conspicuously often associated to a single IP address, and whether single IP addresses are conspicuously often associated to a single cluster [NH17].

heuristics that result in disjoint sets of Bitcoin addresses. Then, we assign transactions to IP addresses based on network observations. Furthermore, each transaction is assigned to a cluster, based on the input addresses used in the transaction. Finally, we validate our approach by analyzing the correlation between IP addresses and clusters.

The remainder of this chapter is structured as follows: After briefly addressing related work in Section 7.1, we describe and apply all clustering heuristics that are known to us in Section 7.2. The network based deanonymization approach is presented in Section 7.3, which also contains the validation of the approach. Finally, a discussion of the results is given.

7.1 Related Work

The anonymity of users in Bitcoin has been analyzed in several ways in the past. The fact that all transactions are publicly available facilitated clustering approaches with the goal to group Bitcoin addresses by the controlling user. We will review all published heuristics known to us in detail in Section 7.2 and therefore sketch related work only briefly here. The first analysis of anonymity in Bitcoin was performed by Reid et al. [RH13] and already made use of the most commonly used heuristic that links the addresses used by multiple inputs of one transactions to one user.[2] Meiklejohn et al. [MPJ+13] proposed additional heuristics based on the behavior of standard clients.

Blockchain information has not only been used for clustering but also for large scale analysis of the distribution of wealth, common transaction patterns, behavior analysis, etc. [RS13], and for an evaluation of user privacy [AKR+13]. More recently, Nick was able to use ground truth data of consumer wallets due to a bug in a client implementation [Nic15]. This work also proposes a heuristic specific to the behavior

[2] Satoshi Nakamto already mentioned the basis for this heuristic in [Nako8]: „Some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same owner.“ It was later shown that transactions can actually be created by multiple users that each sign only one input [Max13].

of consumers in Bitcoin. Reasons for the effectiveness of clustering have been given by Harrigan et al. [HF16], e.g., the incremental growth of clusters.

Network based information has also been used previously to attack the anonymity of users. For instance, the observation of anomalous relaying behavior has been used to map Bitcoin addresses to IP addresses [KKM14]. Furthermore, it was shown that the creation time of transactions can be used to infer the user's time zone [DS15], which might provide a link to the user's identity given background information. Biryukov et al. [BP14] performed a man in the middle attack on clients using Tor by becoming the only possible Tor exit node by banning all other exit nodes in the Bitcoin network. Because all connections of a victim's peer are then routed through adversarial peers, the adversary can link the IP address used to publish Bitcoin transactions to the used Bitcoin addresses.

7.2 Clustering based on Blockchain Information

Each Bitcoin user can create a practically unlimited number of distinct public/private key pairs and use each of them only for one transaction. Hence, each Bitcoin address can be seen as one pseudonym of the user. The goal of address clustering is to partition the set of Bitcoin addresses into subsets (*clusters*), so that each subset contains the addresses under the control of one user. Several heuristics for address clustering in Bitcoin have been proposed in the past. We will now briefly describe the general procedure for clustering, which uses one or more heuristics, and then describe and discuss the used heuristics.

7.2.1 Clustering Procedure & Heuristics

For the definition of the heuristics used in clustering, we use the following notation, which loosely follows the notation used in [MPJ+13]: Let $t \in T$ be a Bitcoin transaction. Let $\mathcal{P}$ be the set of all addresses specified in all transactions in T. Let the set $\text{inputs}(t) \subseteq \mathcal{P}$ include all addresses referenced by the inputs of a transaction t and the set $\text{outputs}(t) \subseteq \mathcal{P}$ include all addresses contained in the outputs of a transaction t. Let $o_j(t) \in \text{outputs}(t)$ be the j-th output of a transaction t ($j \leq |\text{outputs}(t)|$), and let $i_j(t) \in \text{inputs}(t)$ be the j-th input of a transaction t ($j \leq |\text{inputs}(t)|$) .

The clustering procedure computes a partition $\Pi = \{C_1, C_2, ..., C_n\}$ of the set of all addresses $\mathcal{P}$ with $C_1, ..., C_n$ denoting the resulting clusters. Ideally, each cluster contains all addresses used by one user. The clustering process is depicted in Figure 7.2: All transactions are processed in their temporal order. For each transaction t, a cluster $\hat{C}_t$ is computed that contains all addresses that belong into one cluster, according to the evaluated heuristics. This transaction specific cluster $\hat{C}_t$ encodes which addresses used in the transaction are controlled by one user.

The heuristics are applied in a predefined order, each heuristic further altering $\hat{C}_t$. $\hat{C}_t$ is then used to merge existing clusters in Π, and to add new addresses to Π. Several cases are possible: If $\hat{C}_t$ contains only new addresses (i.e., addresses which are not yet in any cluster in Π, $\hat{C}_t$ is added as a single new cluster to Π. If $\hat{C}_t$ contains one

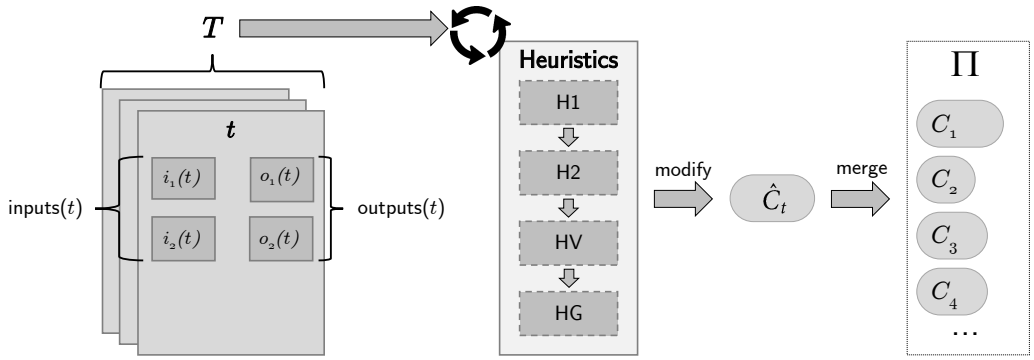

Figure 7.2: Overview of the clustering process. The set of transactions T is chronologically processed using all selected clustering heuristics. The heuristics modify the temporary cluster $\hat{C}_t$ for each transaction t. After processing of each transaction, $\hat{C}_t$ is merged into the partition Π.

or more address from exactly one existing cluster in Π, $\hat{C}_t$ is added to that cluster. If $\hat{C}_t$ contains addresses from more than one existing cluster in Π, all those existing clusters are merged into one cluster, and remaining addresses from $\hat{C}_t$ are added to that merged cluster. This transitively connects all addresses controlled by one user (according to the applied heuristics).

We will now describe the heuristics used for the clustering process.

Heuristic 1 (H1): Multi-Input

If a transaction spends more than one input, the transaction needs to be signed using the private keys corresponding to the public keys from *all* inputs. Assuming that the transaction was created by a single user, that user controls all addresses that are input to the transaction. This heuristic was first used in [RH13] and [MPJ+13].

For a transaction t the cluster determined by this heuristic is $\hat{C}_t = \text{inputs}(t)$. This heuristic is always applied first and is used for all used combinations of clustering heuristics. This heuristic only produces false positives (i.e., clustering addresses that are not controlled by the same user into the same cluster), if the assumptions are not correct. This can be either the case if users give services access to their private key (e.g., Mt.Gox) or if transactions are assembled by multiple users in a decentralized fashion (e.g., CoinJoin [Max13]).

Heuristic 2 (H2): Change Address

Every output of a transaction can only be spent in its entirety. Hence, if Alice controls an unspent output worth 2 BTC and wants to pay Bob 1 BTC, Alice creates a transaction claiming the 2 BTC as an input with two outputs: One output of 1 BTC to Bob's address and one output of 1 BTC to a *change address* [MPJ+13] under the control of Alice (assuming no transaction fees). Since the change address as well as the addresses of the inputs (cf. *H1*) are all controlled by Alice, they should be clustered together. The challenge is to identify which output is the change address and which output is the address of the payee, which should be in a different cluster. Meiklejohn et al. [MPJ+13]

proposed the following heuristic to identify the change address: An output $o_j(t)$ is the change address if these four conditions are met:

1. This is the first appearance of the address $o_j(t)$.
2. The transaction t is not a coin generation transaction.
3. There is no address within the outputs, which also appears on the input side (self-change address).
4. Condition 1 is only met for exactly one $o_j(t)$ and not also for some $o_k(t)$ with $j \neq k$.

For a transaction t the partition determined by this heuristic (based on $\hat{C}_t$ from *H1*) is

$$\hat{C}_t = \text{inputs}(t) \cup \{o_j(t)\}.$$

The rationale behind this heuristic is that the reference Bitcoin client creates a new key pair for change addresses and only uses these addresses once when the received change is spent again. Ancient version of bitcoind used to send change to an address that was also used as input (self-change address).

Obviously, this heuristic can lead to false positives and false negatives: In a transaction with two outputs that have not appeared before, it is not possible to determine the change address (cond. 4), although there might be one. Also, a transaction could spend money to two payees without any change and the heuristic could mistake one of the payees' addresses for the change address.

Heuristic 2 exceptions

In order to capture changing wallet behavior, two exceptions to Heuristic 2 have been proposed in [MPJ+13]: There is no change address in a transaction t if there is an output address in t that…

- had already received exactly one input (**H2a**).
- had been used in a self-change transaction before (**H2b**).

These exceptions captured common behavior in 2013, however, it is not clear whether the exceptions are useful anymore.

We now define an additional exception to heuristic *H2* that makes use of blockchain information that is newer than the current processed transaction t. The behavior for change addresses is that they are only used once. In *H2* we demand that, in order to qualify as a change address, an address must not occur before t. However, with **H2c** we demand that the address also does not occur in later transactions (except for one occurrence as an input).

Value based (HV): Optimal Change

If a transaction has only one output, whose value is smaller than any of its inputs, this output address is likely the change address. This heuristic is based on the behavior of Bitcoin clients to minimize the transaction size, i.e., the number of inputs and outputs: If the change was larger than any input, the input could be omitted and the change could be reduced by this input. This heuristic was used in [Nic15].

Table 7.1: Comparison of all heuristics. Total number of addresses: 196,963,722, total number of transactions: 172,868,721 [NH17].

Heuristics	# Cluster	∅Size	Max Size	#Cluster w/ Size 1
H1	88 m	2.24	12 m	65 m
H1+H2	46 m	4.25	92 m	29 m
H1+H2a	51 m	3.89	87 m	32 m
H1+H2b	63 m	3.10	66 m	40 m
H1+H2c	48 m	4.13	85 m	30 m
H1+HG$_{10}$	146 m	1.34	0.1 m	123 m
H1+HG$_{100}$	121 m	1.62	0.25 m	97 m
H1+HG$_{1000}$	108 m	1.83	1 m	84 m
H1+HG$_{10000}$	104 m	1.88	8 m	81 m
H1+HV	72 m	2.71	76 m	62 m

Consumer based: Redeeming Transaction

Nick [Nic15] proposed a heuristic that uses properties of the redeeming transaction of a possible change output (i.e., the transaction with the change output as an input). For a change address it requires that the redeeming transaction has at most two outputs. The heuristic was used specifically for clustering consumer wallets that show this characteristic. The approach used by Nick made it possible to identify consumer wallets. As we cannot distinguish between consumer wallets and other wallets, we omit this heuristic from further analysis.

Cluster Growth (HG)

In [HF16] it has been shown that clusters normally grow in steady, but small steps. Especially the merger of two already large clusters by a new transaction is unlikely and might hint at a false positive from one of the applied heuristics. This observation can be formulated as a heuristic that can be applied after other heuristics have already established a transaction specific partition.

HG$_k$: If updating Π with $\hat{C}_t$ would cause the largest affected partition in Π to grow by more than a constant number of k addresses, then set $\hat{C}_t = \varnothing$.

Discussion

To our knowledge, we list all heuristics that were published. However, there is a whole class of heuristics that we barely cover. Although most described heuristics only consider single transactions, heuristics could exploit knowledge of the complete transactions graph and base their decisions on any property derived from the graph. The consumer based heuristic and the cluster growth heuristic use simple transaction graph information, but much more sophisticated methods, e.g., facilitating metrics such as connectivity or centrality are possible.

Furthermore, we acknowledge that a lot of manual effort can be put into a better clustering by carefully inspecting special cases, modeling specific behavior and man-

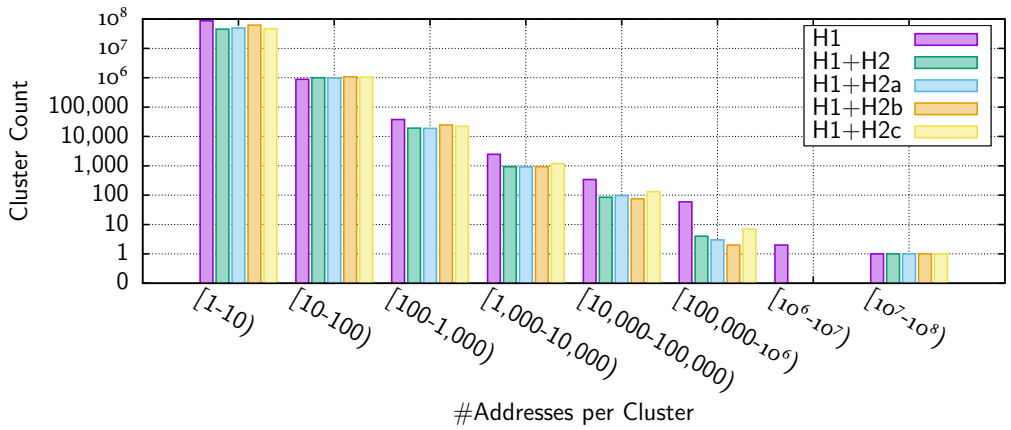

Figure 7.3: Histogram of the resulting cluster sizes for heuristics *H1* and variants of *H2* [NH17].

ually merging or splitting clusters. For the sake of comparability, we chose not to do any manual intervention in our clustering process. We also acknowledge that many heuristics were proposed several years ago and might depend on client behavior, which could have changed in the meantime. For instance, between May 2015 and April 2017, more than 78,000 CoinJoin could be identified [GKRN17].

7.2.2 Results

We will now compare the results of the clustering process with different combinations of heuristics. The clustering was performed at blockchain height 440,349 (November 24th, 2016). Using machines equipped with a Xeon E7-8837 and 512 GB memory, one run of our implementation[3] of the clustering process took about 30 minutes to complete. Prior to clustering we generated the transaction graph as a pointer-based data structure. This process takes several hours but requires less memory and only has to be done once. The generated data structure is then read to memory by the clustering process, which is run completely in-memory and requires no further hard disk accesses.

Table 7.1 lists a comparison of key properties of the resulting clusterings for the heuristics *H1*, all discussed variants of *H2*, *HV*, and several variants of *HG*. The shown properties include the total number of clusters (*# Cluster*), the average number of addresses per cluster (*∅Size*), the number of addresses of the largest cluster (*Max Size*), and the number of clusters that consist of only one single address (*#Cluster w/ Size 1*).

Applying only heuristic *H1* results in a clustering with 88 m clusters. Additionally applying *H2* causes more clusters to be merged, hence resulting in fewer, but bigger, clusters. Additionally applying variants of *HG*, however, causes fewer clusters to be merged, hence resulting in more, but smaller, clusters.

The different variants of heuristic *H2* lead to 46 m to 63 m clusters. The three exceptions to *H2* cause fewer clusters to be merged than by applying *H1* and *H2* only.

[3] https://github.com/tillneu/bitcoin-clusterer

The strongest effect on the resulting clusters has *H2b*, which reduces the average cluster size from 4.25 for *H2* to 3.1 addresses per cluster for *H2b*. Figure 7.3 shows a histogram of the resulting cluster sizes for heuristic *H1* and for variants of the heuristic *H2*. Interestingly, all variants of *H2* cause the number of clusters for all sizes except 10 to 100 addresses to decrease when compared to applying *H1* only.

The value based heuristic *HV* has only a small effect on the average cluster size (grows to 2.71 addresses per cluster) but a large effect on the size of the largest cluster (from 12 m to 76 m). A possible explanation for the result is that a disproportionately large share of transactions that originated from that super-cluster have a combination of input and output values that makes *HV* applicable to them, thus merging more addresses into the super-cluster.

A small choice of the parameter k for the heuristic *HG* causes fewer clusters to be merged as the threshold is easily exceeded. This causes the average cluster size to decrease down to 1.34 addresses per cluster for HG_{10}. Figure 7.4 shows a histogram of the resulting cluster sizes for variants of the heuristic *HG*. Notably, there are only minor changes in the number of clusters with a size of 10 to 100,000 addresses. However, the number of clusters with 100,000 to 1 million addresses is reduced to 1 using HG_{10} compared to 59 using only *H1*. Most likely, transactions that cause a false positive in *H1* are less likely to occur in these medium sized clusters.

In all variants the largest identified cluster contains between 100,000 and 92 m addresses. This cluster contains among others the addresses of the former exchange Mt.Gox. The existence of this super-cluster was also discussed in [HF16]. The size of that cluster is substantially increased by application of variants of *H2* and *HV*, whereas the application of *HG* can limit the growth of that cluster.

The differences in the resulting cluster sizes, counts, and composition of the clusters depending on the used heuristics illustrate the challenges in the validation of clustering approaches without ground-truth data. While it is still possible that one combination of heuristics performs a perfect clustering (i.e., one cluster per user), it is much more likely there are errors in all resulting clusterings - we just cannot identify these errors because of a lack of ground-truth data. For our approach, it is not required to identify the *best* heuristic and assess its quality. Instead, we compare *all* resulting clusterings with our collected network-based observations.

7.3 Network Information

We will now explain how network based information was acquired and how that information is compared to the blockchain information based clustering results. The main idea is to associate IP addresses to transactions based on observations on the Bitcoin P2P network and then use the previously established linking between clusters and transactions in order to determine the correlation between clusters and IP addresses.

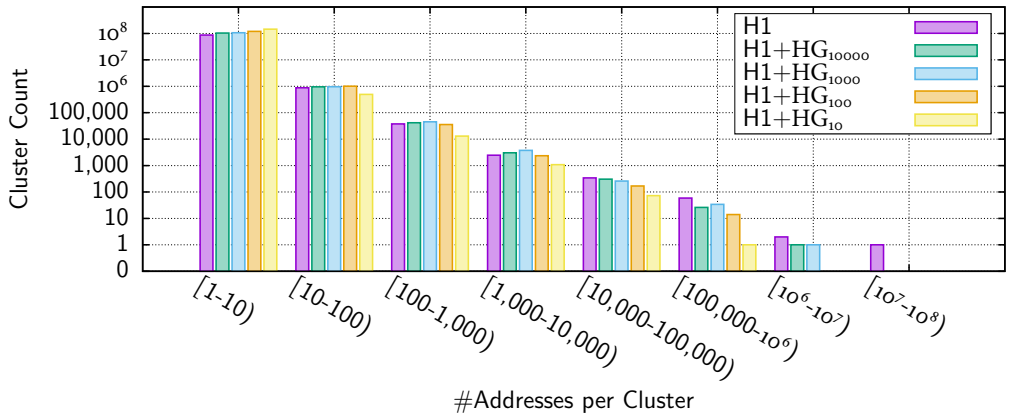

Figure 7.4: Histogram of the resulting cluster sizes for heuristics *H1* and variants of *HG* [NH17].

7.3.1 Association of Transactions and IP Addresses

In order to observe transactions being flooded through the network, we deployed two monitor peers that maintain connections to all reachable peers in the network and log for each transaction, when it is received from each peer in the network (cf. Section 4). For each transaction there is one peer (*originator*) which first sent the transaction to our monitor peer. We want to associate one IP address to each transaction. However, we cannot conclude that the first peer we received a transaction from has really first brought the transaction to the network, nor can we conclude that the peer generated the transaction. First, the user could connect to any reachable peer in the network, send the transaction to that peer and leave the network afterward. Secondly, due to trickling, the transaction can be sent to other network peers, which might forward the transaction to our monitor peers before we receive the transaction from the creating peer. Therefore, we apply several heuristics that aim at reducing the number of obviously false mappings:

- If both monitor peers first received a transaction from different peers, we discard both possible originators.
- If the time difference at which the transaction is received from the originator by both monitor peers differs by ≥100 ms, the originator is discarded.
- The subsequent receptions of the transaction from other peers must not be faster than what the speed of light in fiber allows. By using the Maxmind GeoIP services[4], we can approximate the location of the other network peers and establish a lower bound on the time it takes for a transaction to be transmitted from the originator to our monitor peer *via* any other network peer. If we receive a transaction faster than that lower bound, we discard the originator.

[4] `http://dev.maxmind.com/geoip/`

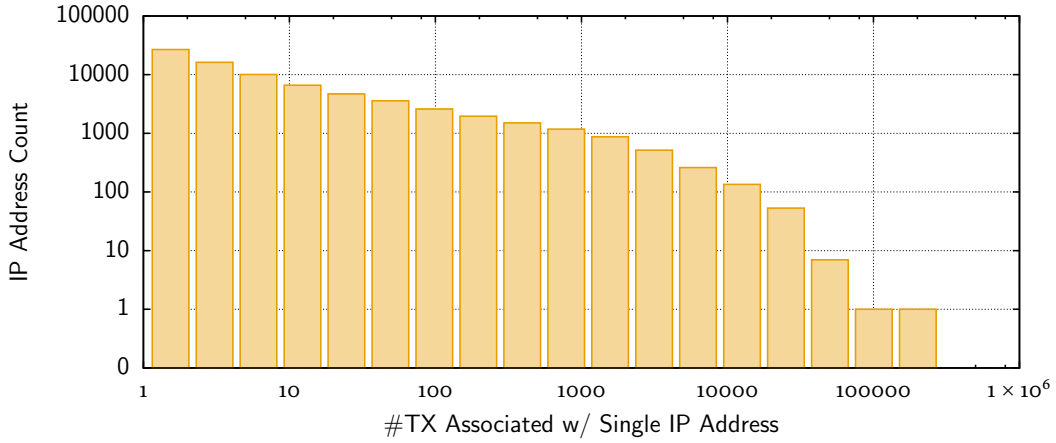

Figure 7.5: Histogram of the number of unique transactions associated per IP address. Read as: *There are 10,000 IP addresses that are associated to 4 to 8 transactions each* [NH17].

During the monitored period between block 366,000 (2015-07-19) and block 440,349 (2016-11-24), 96,520,958 transactions were added to the blockchain. For 9,934,056 of these transactions (≈ 10 %), we identified an originator IP address using the heuristics described above. In total, 79,079 unique IP addresses appeared as originators. This leads to an average of about 125 transactions per IP address. However, the number of transactions associated per IP address follows a heavy tailed distribution. Figure 7.5 shows the distribution of how many transactions were associated with each IP address. Most IP addresses were an originator address only for a small number of transactions. However, two IP addresses were originators for more than 65,000 transactions each. Interestingly, both of these IP addresses (one of which IPv4 and one IPv6) are in IP ranges assigned to the same hosting provider.

Although we are able to associate IP addresses to transactions, we do not know whether the mapped IP addresses in fact identify the user that issued the transaction and simply regard the IP address as a piece of information that might be linked to the user. In order to validate that linking, we will now compare the results from the clustering based on the transaction graph to the collected IP address information.

7.3.2 Methodology

We will now introduce the notation used for the association of clusters with IP address information. For the association between transactions and clusters we use the following notation: Let $c(t)$ describe the cluster that issued a transaction t according to the evaluated heuristics.[5] Let the set of transactions issued by a cluster C be $T_C := \{t \in T : c(t) = C\}$. For the association between transactions and IP addresses as described in Section 7.3.1 we use the following notation: Let $\mathcal{A}$ be the set of all

[5]Each transaction is associated with exactly one cluster, because each transaction is processed individually, and the resulting transaction specific cluster $\hat{C}_t$ is merged with all existing clusters with the same addresses.

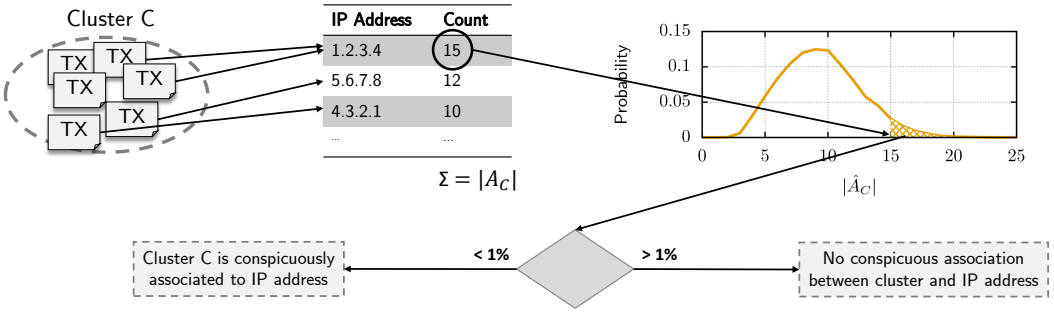

Figure 7.6: Decision process for flagging the association between clusters and IP addresses as *conspicuous*.

observed IP addresses. Let $a(t) \in \mathcal{A}$ describe the IP address of the originator (if any) of a transaction t. Finally, we define the tuple of all IP addresses associated with a cluster C as $A_C = (a(t) : t \in T_C)$. A_C is defined as a tuple instead of a set because single IP addresses can occur multiple times in A_C and we are interested in that count.

The main question now is whether there is a correlation between clusters and IP addresses or whether for each transaction the originator is simply a random IP address. Both, IP addresses and clusters, are nominal variables that cannot be ranked in any way. Standard statistical methods (e.g., [MBB12]) would suggest to fill a contingency table with all observed IP addresses as one dimension and all clusters as the other dimension. Then, for each tuple (IP address, Cluster) the expected frequency and the observed frequency could be compared using the chi-squared test. However, a problem with the data is that the contingency table is very sparsely populated. In order to perform the chi-squared test, no more than 20% of the expected frequencies should be less than 5 and all individual expected frequencies should be 1 or greater [YMM96], which is not the case for our data. Even if the frequencies were sufficient, the large sample size would cause biased results [LLJS13].

Therefore, we analyze each cluster C separately in order to see whether the associated IP addresses A_C are independent. Figure 7.6 depicts our approach: First, for each cluster, the IP address $\hat{A}$ with the highest observed frequency $|\hat{A}_C|$ in this cluster is selected. Then, we calculate the probability that such a high frequency $|\hat{A}_C|$ (or even higher) can be observed if IP addresses and clusters are independent. If this probability is smaller than 1 %, we reject the independence hypothesis, conclude that the IP address and the cluster are conspicuously associated, and add the cluster to the set of conspicuous clusters C^+.

The only information required for this approach is the probability that a certain frequency $|\hat{A}_C|$ is observed given the total number $|A_C|$ of IP addresses associated with this cluster and assuming independence. This translates to a simple *urn problem*, where a number of $|A_C|$ IP addresses are selected at random from an urn (with replacement). The probability to choose an IP address A would be $P(A) = |A| / \sum_{A' \in \mathcal{A}} |A'|$ (with $|A|$ being the total observation count of A, i.e., the share of an IP address in all

observations, cf. Figure 7.5).[6] The question to be answered with the urn problem is the probability of selecting one IP address at least $|\hat{A}_C|$ times.

Obviously, in addition to checking for each cluster whether the associated IP addresses were randomly chosen, we can also check for each IP address whether the associated clusters are randomly chosen. This analysis has been also performed using the same method as described above for the opposite direction with T_A denoting the set of transactions associated with an IP address A, and $\mathcal{A}^+$ denoting the set of conspicuous IP addresses according to the hypothesis testing.

7.3.3 Results & Discussion

From our data we selected all clusters with at least two IP addresses associated ($|A_C| \geq 2$), determined $|\hat{A}_C|$ for these clusters, and calculated the set of conspicuous cluster C^+. Table 7.2 shows the number of clusters with at least two associated IP addresses ($|\{C : |A_C| \geq 2\}|$) and the number of conspicuous clusters $|C^+|$ for various heuristics. The number of clusters with at least two associated IP addresses varies between 283k and 456k clusters. Comparing these numbers to the total number of clusters (cf. Table 7.1) shows, that only a small percentage of all clusters has two IP addresses associated, with the highest percentage for the *H1+H2c* combination.

The number of clusters $|C^+|$ with a conspicuously large $|\hat{A}_C|$ varies between 15k and 35k, which corresponds to 5 % to 8.3 % of the considered clusters. For comparison, when randomly selecting IP addresses based on their a-priori probability, the share of conspicuous clusters is around 1 %. The results indicate that the highest correlation between clusters and their associated IP addresses exists, when clustering using variants of *H2*. For the value based heuristic, the growth based heuristic, and the base heuristic *H1*, fewer conspicuous clusters were found.

Table 7.2 also shows the share of conspicuous IP addresses $\mathcal{A}^+$ among those IP addresses with at least two associated transactions. The share varies between 6.2 % and 20.2 % with the smallest percentages for clusterings with variants of *H2*. This is caused by the extremely large super cluster that is created by these heuristics (cf. Table 7.1): The probability to randomly select that cluster very often (assuming independence) increases with the number of transactions associated with that cluster. Therefore, the independence hypothesis gets accepted for more IP addresses.

Only for a small share of clusters and IP addresses, a correlation between clusters and network information could be shown. At least for these clusters, information obtained by observing the network could also be used in a constructive way during the clustering process. For example, the set of candidate clusters for a transaction could be reduced based on networking information. Also, the information could be used for tie breaking when having multiple change address candidates.

For the majority of clusters and IP addresses, we did not observe any correlation to network information. This could mean that there is no correlation, or that the used method did not reveal a correlation. For example, a more powerful observer

[6] For large values of $|A_C|$, the distribution can be approximated with the binomial distribution with p being the probability of the most likely IP address ($p \approx 0.02$ for our data).

Table 7.2: Comparison of the number of clusters with at least two associated IP addresses ($|\{C : |A_C| \geq 2\}|$) and the number and share of conspicuous clusters (C^+), and the share of conspicuous IP addresses ($\mathcal{A}^+$) for various heuristics [NH17].

Heuristics	$\lvert\{C : \lvert A_C\rvert \geq 2\}\rvert$	$\lvert C^+\rvert$	$\frac{\lvert C^+\rvert}{\lvert\{C:\lvert A_C\rvert\geq 2\}\rvert}$	$\frac{\lvert \mathcal{A}^+\rvert}{\lvert\{A:\lvert T_A\rvert\geq 2\}\rvert}$
H1	282,950	14,879	5.26 %	18.7 %
H1+H2	398,802	32,623	8.18 %	6.2 %
H1+H2a	387,696	32,026	8.26 %	6.2 %
H1+H2b	456,063	35,138	7.70 %	6.5 %
H1+H2c	452,189	35,602	7.87 %	6.7 %
H1+HG$_{10}$	299,140	15,537	5.19 %	16.7 %
H1+HG$_{100}$	300,927	15,755	5.23 %	19.6 %
H1+HG$_{1000}$	301,775	16,434	5.45 %	20.2 %
H1+HG$_{10000}$	308,900	18,788	6.08 %	19.7 %
H1+HV	296,132	14,736	4.97 %	6.9 %

with more monitoring nodes could be able to associate IP addresses to transactions more precisely. Furthermore, the statistical analysis used here only reveals certain correlations between a cluster and a single IP address.

7.4 Discussion

In this chapter we performed address clustering in Bitcoin according to published heuristics, and compared the resulting clusters to IP address information obtained from observations in the Bitcoin P2P network. We showed that only a small share of clusters was conspicuously associated with a single IP address, and that only a small number of IP addresses showed a conspicuous association with a single cluster.

Our results indicate that for the vast majority of users network information cannot facilitate address clustering easily. However, a small number of participants exhibit correlations that might make them susceptible to network based deanonymization attacks. A more precise network observation or better clustering heuristics might reveal further correlations that could not be observed with our approach.

The goal of our analysis of network-based deanonymization was to provide a high level view on clusters and their correlation to network observations. An obvious next step could be to focus on single clusters and IP addresses and identify the anomalous behavior that caused the revealed correlations. Since this would require an in-depth analysis of single entities on the network, we decided not to carry out such an analysis without ensuring the user's privacy. We emphasize that for ethical reasons no further attempt at linking the conspicuous IP addresses or clusters to other available information was performed.

In order to perform an in-depth analysis of single clusters and peers, a privacy preserving method for the analysis of blockchain and network data should be developed.

The results of such an analysis could point to possible improvements in the P2P protocol or specific client implementations. Furthermore, the consideration of temporal changes such as IP address changes or changes in client implementations might further improve the quality of our analysis. Finally, the statistical analysis might benefit from more advanced methods to establish sharper bounds on possible correlations.

8
Conclusions and Outlook

Since the publication of the concept of Bitcoin in 2008, permissionless blockchains have received widespread attention from the public as well as from the scientific community. While most research focuses on the consensus layer and the application layer, the underlying network layer is often overlooked.

In this dissertation we address security and anonymity aspects of the network layer of permissionless blockchains by answering two main research questions. The first research question is motivated by the limited research of the network layer of permissionless blockchains:

How to research requirements and tradeoffs present in the network layer mechanisms of permissionless blockchains?

In this dissertation we show that such research can be conducted by a combination of (1) a systematization of known attacks and design aspects of the network layer of permissionless blockchains, (2) measurements and experiments performed in real-world P2P networks, and (3) simulation studies based on validated models.

So far, a number of attacks on the network layer of permissionless blockchains have been analyzed, however, no general notion of the desired security properties of the network layer of blockchains exists. Furthermore, the effect of design options on the fulfillment of performance and security properties is not sufficiently understood. We provide a **systematization of attacks** on the network layer of Bitcoin regarding the exploited network layer mechanisms and the effects of the attacks on higher layers of Bitcoin. We show that the goal of all studied attacks is either monetary benefit or the deanonymization of users by linking IP addresses to application data. Furthermore, all known attacks aiming at a monetary benefit of the adversary are based on DoS attacks. In contrast to DoS attacks on classical unstructured P2P networks (e.g., file sharing networks), DoS attacks on permissionless blockchains can enable attacks on

the consensus system, which can result in direct monetary loss for participants of the system. Based on the systematization, we derived the requirements performance, low cost of participation, anonymity, DoS resistance, and topology hiding for the network layer of permissionless blockchains.

Our **survey of the design space** of the network layer qualitatively shows the effect of design decisions on the fulfillment of the requirements and illustrates existing tradeoffs. Developers of network layers of permissionless blockchains may use this knowledge in order to adjust the properties of the developed system according to their requirements. Furthermore, each identified tradeoff can be seen as a starting point for new research.

The assessment of the security of distributed systems requires an understanding of the general (protocol) behavior and the characteristics of the specific system. While the behavior can often be deducted from specifications, the characteristics of a decentralized system are influenced by numerous factors (e.g., user behavior), which can strongly deviate between two instances of the same protocol. Observations of the real-world deployment of permissionless blockchains enable us to characterize these systems. In this dissertation we provide a **characterization of the Bitcoin P2P network**, based on measurements performed since 2015. The network characterization enables the parametrization and validation of our simulation models by directly providing some of the required parameters (e.g., number of reachable peers) and by serving as a basis for the estimation of other parameters (e.g., peer-to-peer latencies, number of unreachable nodes). Our measurements not only serve as a basis for simulations, but also enable the assessment of the reliability of experiments performed in the Bitcoin network. Finally, the network characterization provides insights into the behavior of peers and their operators. For instance, we show that Sybil events actually happened in the Bitcoin P2P network, and that the performance and anonymity of transaction and block propagation has been improved by implementation and protocol changes.

While experiments in deployed networks can deliver substantial insights, their execution is not only costly but also commonly subject to ethical and legal concerns. Furthermore, the complexity of the analyzed networks often prevents a purely analytical approach. Therefore, simulation is a common method for the analysis of network based attacks as it allows capturing complex system behavior and network parameters. In this dissertation we present two approaches for modeling the behavior of clients of the Bitcoin P2P network as a **discrete-event simulation model**. Based on the conducted measurements we parametrize our simulation model, and validate the parametrization of our simulation models by comparing the simulated information propagation delay to the observed one. Both information propagation delays only show a minor deviation, indicating a high correspondence between the simulated network and the real network. The presented simulation models enable the simulation of the Bitcoin network with a precision sufficient for the analysis of attacks at the full scale of the Bitcoin network.

We show that research on the network layer of permissionless blockchains can be conducted using the proposed combination of systematizations, measurements, experiments, and simulations by analyzing the possibility of network-based topology

inference and deanonymization in Bitcoin using these methods.

Under which assumptions and to which degree are network-based topology inference and deanonymization (im-)possible in Bitcoin?

Our systematization of attacks shows that resistance against denial-of-service and deanonymization attacks is an important requirement of the network layer of permissionless blockchains. A key factor for the feasibility and costs of these attacks is knowledge of the P2P network topology. In this dissertation we give an in-depth analysis of the possibility and limitations of **topology inference** in Bitcoin. We propose four different methods for topology inference that exploit four different aspects of the network layer, i.e., double spends, timing, transaction accumulation, and peer discovery. We evaluate the methods using real-world experiments and simulations.

We can summarize the results of our experiments performed in the Bitcoin network and the Bitcoin testnet in the following statements:

- Assuming an active adversary that is able to connect to all peers of the network, inference of the neighbors of a peer is possible with high recall and precision (87 % recall, 71 % precision) at low costs (99 transaction fees per peer) by a coordinated creation of double spending transactions. Remark: connecting to *all* peers of the network is a non-trivial task.

- Assuming a passive adversary that is able to connect to all reachable peers, inference of the neighbors of a peer was possible in 2016 with a recall of 40 % at a precision of 40 % (only considering connections between reachable peers) by observing at least 8 transactions originating from that peer.

Furthermore, we can summarize the results of our simulation studies in the following statements:

- Assuming an active adversary that is able to connect to all peers of a simulated network consisting of 500 peers, a coordinated creation of transactions that exploits transaction accumulation can lead to the correct inference of 20 existing connections of the network and the incorrect inference of 2 non-existing connections at minimal costs. Remark: The adversary has no influence on which connections will be inferred.

- Certain parametrizations of transaction relay delays can actually improve topology inference, instead of defying it.

- A P2P network's peer discovery mechanism can be configured so that the reception of 40 address messages by the adversary results in a recall of less than 20 %, and so that a sufficient number of reachable IP addresses is provided to peers.

Overall, the results indicate that substantial effort and strong adversary models are required for a high topology inference quality. Therefore, we choose to model our adversary in the analysis of a **network based deanonymization** attack without

access to network topology information. We assess whether a correlation between the IP address of the peer announcing a transaction first and the suspected creator of the transaction exists. Our dataset is based on our network measurements and consists of almost 10 million transactions with associated IP addresses. We handle the inherent ground truth problem of the association of transactions to users by applying all previously published Bitcoin address clustering heuristics known to us. The results from our analysis can be summarized in the following statements:

- Transactions of 5 % to 8.3 % of users (as identified by the used clustering heuristics) were conspicuously often published by single peers, potentially making these users susceptible to network-based deanonymization attacks.
- No such association could be identified for the remaining users.

We emphasize that network-based deanonymization might be also possible for the latter group of users under the assumption of stronger adversary models.

Outlook

The results presented in this dissertation suggest several directions of future research. First, discrete-event simulations were extensively used in this dissertation, especially in the analysis and execution of several topology inference approaches. However, the creation of discrete-event simulation models is time consuming and requires ongoing effort, because client implementations as well as the constitution of deployed networks constantly change. Therefore, the automatic generation of simulation models based on source code fragments of client implementations, along with an ongoing adaption of the parametrization based on measurements could facilitate more research on the network layer of permissionless blockchains.

Secondly, our characterization of the Bitcoin P2P network only gives a view of the network from one vantage point. One limitation of this approach is that it is often not possible to definitely decide, whether an observed effect is caused by the observed network or by the measurement infrastructure itself. We believe that data aggregated from a geographical and organizational distributed measurement infrastructure could provide a better view on P2P networks.

In this dissertation, methods for topology inference have been extensively analyzed, however, topology inference has not been applied to real-world networks with the goal of actually learning the network topology of that network. The main reasons for not applying a topology inference approach depend on the analyzed method and are either the low expected inference quality, or the high costs, or assumptions that cannot be met in the real network with the available resources. In order to use topology inference for gaining insights on the network, the presented approaches might be further improved. However, we emphasize that hiding the network topology is an important security objective of the network layer of permissionless blockchains. Therefore, while inference of the network topology is of scientific interest, preventing such inference is also in the interest the users of the permissionless blockchain.

Finally, the adversary model considered in the analysis of network based deanonymization in this dissertation does not match the capabilities of organizations actually

interested in performing such an attack. However, our results regarding topology inference suggest that perfect knowledge of the network topology is also hardly achievable. Therefore, the analysis of network based deanonymization might consider adversary models with partial and imperfect topology knowledge.

A
Proofs of Topology Inference Methods

In this chapter we provide proofs for the correctness of the topology inference approaches exploiting transaction accumulation and double spends.

Let $G = (V, E)$ be the undirected graph modeling the peers (V) and connections (E) of the Bitcoin network. In our proofs we make the following assumptions:

- The adversarial monitor peer v_M is connected to all peers $v_i \in V$ of the network.
- The network topology remains static during the execution of the topology inference.
- All peers eventually forward valid and non-conflicting transactions to their neighbors.
- The adversary is able to send messages to all peers of the network, so that all messages arrive at the same time at all peers.

A.1 Exploiting Transaction Accumulation for Topology Inference

We furthermore assume that all peers use the transaction accumulation method described in Section 6.1, i.e., valid transactions are queued into outgoing queues per neighbor. Eventually, all elements of a queue are announced in one `INV` message to a peer's neighbor.

The adversary creates one transaction $t_i \in \tau$ for each connected peer v_i. All transactions are independent and not conflicting in any way (i.e., they are spending different

outputs). All transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the same time.

Let $v_A \in V$, $v_B \in V$, and $v_C \in V$ be peers on the network.

Theorem 1. *If the first* `INV` *message that peer v_A sends to v_M contains only t_B (i.e., the transaction sent to v_B) and no other transaction from the set of created transactions τ, then v_A and v_B are directly connected.*

Proof. Theorem 1 is equal to the statement *If v_A and v_C are **not** directly connected, then the first* `INV` *message that v_A sends to v_M will **not** contain only t_C and no other transaction from the set of created transactions τ.* We will now prove this statement. Assume that peers v_A and v_C are not directly connected.

If there is no path between v_A and v_C, transaction t_C cannot reach peer v_A, therefore, v_A cannot send an `INV` message containing t_C. If there is a path between v_A and v_C, then there is at least one additional peer, say, v_B on that path, because v_A and v_C are not directly connected. Let us for now assume that v_B is the only peer on the path between v_A and v_C. Because the adversary is connected to all peers, v_M is also connected to v_A, v_B, and v_C. Because incoming transactions are immediately written into outgoing queues at each peer, and because all transactions are sent by v_M so that they arrive at the same point in time, the outgoing queue from v_B to v_A contains t_B at that time, and the outgoing queue from v_C to v_B contains t_C at that time.

In order for peer v_A to send an `INV` message that contains only t_C and no other transaction from the set of created transactions τ, the outgoing queue of v_A to v_M has to contain t_C and not t_B when v_A sends its first `INV` message to v_M. As t_C can reach v_A only via v_B, and v_B's outgoing queues already contain t_B, and because all transactions in a queue are sent in one message, t_B will be sent by v_B no later than t_C. Therefore, the outgoing queue of v_A to v_M cannot contain t_C and not t_B at the time of sending the first `INV` message from v_A to v_M. By induction this also holds if v_B is not the only peer on the path between v_A and v_C, and if there is more than one path between v_A and v_C. □

Please note that the method does not guarantee any progress, i.e., we cannot exclude that the first `INV` message v_A sends *always* contains more than one transaction from the set of created transactions τ.

A.2 Exploiting Double Spends for Topology Inference

We furthermore assume that peers drop transactions that spent outputs that were already spent by a transaction that has previously been received by the peer (double spends). The adversary selects on target peer v_T, the connections of which shall be inferred. The adversary creates one transaction $t_i \in \tau$ for each peer $v_i \in V$, except for the target peer v_T. All transactions have the same input, but all transactions are unique. All transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the same time.

Theorem 2. *If the target peer v_T forwards transaction t_A to the monitor peer v_M, then v_A and v_T are directly connected.*

Proof. Theorem 2 is equal to the statement *If v_A and v_T are **not** directly connected, then the target peer v_T will **not** forward transaction t_A to the monitor peer v_M.* We will now prove this statement. Assume that peers v_A and v_T are not directly connected.

If there is no path between v_A and v_T, transaction t_A cannot reach peer v_T, therefore, v_T cannot forward transaction t_A to v_M. If there is a path between v_A and v_T, then there is at least one additional peer, say, v_B on that path, because v_A and v_T are not directly connected. Because the adversary is connected to all peers, v_M is also connected to v_A, v_B, and v_T. Because all transactions are sent by v_M so that they arrive at the same point in time, v_B has received t_B no later than v_A can send t_A to v_B As t_A is conflicting with the already received t_B, v_B will drop t_A upon reception. Therefore, t_A cannot reach v_T. The same argument applies for other paths and longer paths. □

Progress, i.e., that the target peer v_T actually forwards a transaction from the set τ to the monitor peer v_M, follows directly from the bounded interval between clients flushing the queues, assuming the target peer is connected to at least one other peer besides v_M. With variant *Suppress*, the repeated inference of the same connection can be prevented.

B
Approximative Propagation Delay Model

The proposed timing analysis method infers the number of hops a message traveled through the network by comparing the observed delay to the delay that is to be expected for several path lengths. Therefore, a model for the propagation delay depending on the path length between sender and receiver is required. Given two randomly chosen peers as the sender and the receiver and their (hop) distance in the network graph, we want to know the delay distribution between sending and receiving a message. The delay model presented in this chapter has been previously published in [NAH16].

B.1 Notation and Assumptions

We will now introduce the notation used in the propagation model. All random variables in this model are regarded as discrete random variables. The network is represented as a connected and undirected graph $G = (V, E)$, where V is the set of peers ($v \in V$) and E is the set of connections ($e \in E$) between peers. We assume a random, *Erdos-Renyi* graph network model [ER59], which means each possible edge exists independently with a certain probability p.

A simple path R between two peers is a sequence of edges $(e_1, ..., e_c)$ connecting the peers with a length of $|R| = l$.[1] We use simple paths to model the propagation of messages through the flooding network. As the graph is assumed to be connected, at least one path between any two vertices of the graph exists. Furthermore, we define C_{min} as the minimum path length between two peers.

In this model, we assume that the message delay (i.e., the time difference between sending by one peer and reception by another peer) only depends on the length of

[1] Each vertex may only be contained once in a simple path.

the path the message traversed. We define $P(D_l = t)$ as the probability for a message delay of t for a message transmitted via a path of length l. We assume that these probability distributions are known to the adversary.

B.2 Probability for Shortest Path Length

We will now show how to approximate $P(C_{min} = l)$, i.e., the probability that the shortest path between two randomly chosen peers of an ER graph equals l. The probability can be calculated based on the *maximum possible number* Z_l of simple paths with length l between two peers. Z_l can be calculated as

$$Z_l = \binom{|V| - 2}{l - 1} \cdot (l - 1)!. \tag{B.1}$$

Proof. Let s and t denote the first and last node of the path. A simple path of length l traverses the nodes s, t, and $l - 1$ additional nodes, which can be freely chosen from all nodes of the network, except for s and t. Hence, there are $\binom{|V|-2}{l-1}$ possible sets of nodes for a simple path of length l. The graph can traverse all nodes except for s and t in any order, i.e., there are $(l - 1)!$ permutations of the path. □

The probability of a shortest path length of l between two randomly chosen peers s and t is calculated for $C_{min} < 3$ as

$$P(C_{min} = l) = (1 - (1 - p_l)^{Z_l}) \cdot \prod_{i<l} (1 - p_i)^{Z_i}. \tag{B.2}$$

Proof. Case $C_{min} = 1$: Z_l equals 1, hence the term evaluates to p_1, i.e., the probability that the shortest path length between two peers is 1 equals the probability that an edge between those peers exists. Case $C_{min} = 2$: There are Z_2 possible paths of length 2 between s and t, each path exists independently with probability p_l. Hence, $1 - (1 - p_l)^{Z_l}$ gives the probability that at least one such path exists. The product $\prod_{i<l}(1 - p_i)^{Z_i}$ is calculated only for $i = 1$, i.e., it gives the probability that no path of length 1 exists. □

The given formula is only exact for $C_{min} < 3$, because longer paths can overlap (i.e., they are not vertex independent between s and t) and their probability of existence is not independent anymore. For $C_{min} \geq 3$ the given formula only approximates the real probability.[2]

B.3 Delay Distribution Depending on Path Length

For the presented timing analysis approach we need to calculate $P(D = t|C_{min} = l)$, i.e., the probability of observing a message delay t, given a shortest path length l. Let $P(\hat{D}_k = t)$ be the probability of a delay t of a message sent over any existing path of length k. Then, we can calculate $P(D = t|C_{min} = l)$ as

[2] A recursive approach might result in better approximations [KNbA+15].

$$P(D = t|C_{min} = l) = \sum_k (P(\hat{D}_k = t|C_{min} = l) \cdot P(\hat{D}_{k' \neq k} \geq t|C_{min} = l)). \tag{B.3}$$

Each summand gives the probability for a delay t over a path of length k, and no shorter delay over any path of a different length.[3] Iterating over all considered path lengths k results in the overall delay distribution.

We will now show how to calculate the probability $P(\hat{D}_k = t|C_{min} = l)$ of a delay t of a message sent over any existing path of length k. Let $P(z_k = n)$ be the probability that *exactly* n paths of length k exist. Then, $P(\hat{D}_k = t|C_{min} = l)$ can be calculated as

$$P(\hat{D}_k = t|C_{min} = l) = \sum_{n=1}^{Z_k} (P(z_k = n|C_{min} = l) \cdot P(\hat{D}_k = t|z_k = n)). \tag{B.4}$$

We iterate over all possible number of paths of length k (i.e., n goes from 1 to Z_l) and sum the joint probability that exactly n paths of length k exists *and* that this number of paths results in a delay of t. Therefore, we have to be able to calculate the probability $P(z_k = n)$ (i.e., exactly n paths of length k exists) and have to be able to calculate the probability $P(\hat{D}_k = t|z_k = n)$ (i.e., the probability of a delay t of a message sent over any existing path of length k assuming there are n paths of length k).

$P(z_k = n)$ can be approximated as the binomial distribution

$$P(z_k = n) \approx \binom{Z_k}{n} \cdot p_k^n \cdot (1 - p_k)^{Z_k - n}. \tag{B.5}$$

For the same reason as Equation B.2 is only an approximation for path length of 3 or more, the given formula is also an approximation for path lengths of 3 or more. Obviously, $P(z_k = n|C_{min} = l)$ equals zero for $k < l$ and $n > 0$, i.e., if the shortest path length is l, the probability for any positive number of shorter paths is zero.

Finally, the probability $P(\hat{D}_k = t|z_k = n)$ of a delay t of a message sent over any existing path of length k assuming there are n paths of length k can be calculated as

$$P(\hat{D}_k = t|z_k = n) = n \cdot P(D_k = t) \cdot P(D_k \geq t)^{n-1}. \tag{B.6}$$

We defined $P(D_k = t)$ as the probability of a message delay of t, when the message traverses a single, isolated path of length k, and assume that this probability distribution is known to the adversary. As there are n paths over which the message can be transmitted first, the delay distribution of each path has to be considered ($P(D_k = t) \cdot n$), however, no other path may result in a shorter delay ($P(D_k \geq t)^{n-1}$).

Summarizing, we broke down the overall probability $P(D = t|C_{min} = l)$ first by path length (Equation B.3) and then by the number of possible paths (Equation B.4) for that length.

Validation and Limitations

The presented propagation model is only an approximation for path length of 3 and more. However, as only short path lengths are of interest for the proposed timing

[3] $P(\hat{D}_{k' \neq k} > t) = \prod_{k' \neq k} P(\hat{D}_{k'} > t)$, i.e., the probability that the delay of all paths with a length different than k result in a delay larger than t.

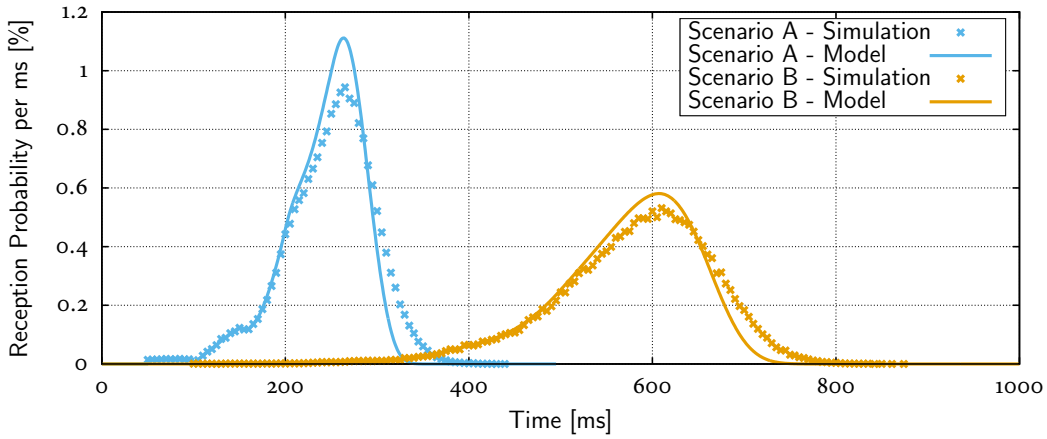

Figure B.1: Propagation Delay: Comparison of Model to Simulation [NAH16].

analysis technique, we can accept errors for longer path lengths. We will now validate the model using simulations in order to assess the quality of the approximation. We consider two exemplary scenarios: *Scenario A* comprises a network with 1,000 peers, an average of 8 connections per peer and a uniformly distributed latency (D_1) between 50 and 100 ms. *Scenario B* consists of 6,000 peers, each with 16 connections on average and a uniform latency distribution (D_1) between 100 and 300 ms. The delay distribution for multi-hop transmission ($D_2, D_3, \ldots$) was calculated by convolution from D_1 in both scenarios. Figure B.1 shows the resulting delays for these two scenarios.

It can be seen that during the beginning of the propagation our model perfectly matches the simulation outcome, whereas a deviation becomes visible afterward. This deviation is caused by the already mentioned abstraction made in the model: we treat two paths between sender and receiver as independent although the paths may share common hops (i.e., they may not be vertex-independent). This causes a wrong estimation on the number of paths. Additionally, when a peer receives a message over more than one path (i.e., the peer is a common hop to two or more paths), it will forward it *once*, effectively joining both paths from this hop on. Therefore, the resulting propagation delay differs to a scenario with two independent paths.

Obviously, this situation can only occur for paths with a length of three or more, as one hop must be the joining hop and one hop must differ in order to result in different paths. Additionally, the likeliness of paths not being vertex-independent increases with the paths' length and the number of possible paths, which also increases with the paths' length. We will use the model to distinguish between a shortest path length of one and any longer path length. Therefore, a precise modeling of the propagation's beginning is required, whereas the further propagation is not crucial.

Another, more practical, limitation of the presented model can be imposed by the possible extremely large number of possible paths Z_c, which can reach values of more than 10^{100} pushing the following calculation of $P(z_c = k)$ against common numerical limits. However, the use of arithmetic libraries as well as limiting the path length bypasses these issues. Lastly, the assumption of an ER-graph does not hold in most

real-world networks. We argue, however, that for some network models an adaption is possible analytically, whereas for others the use of simulation and fitting techniques allows the development of practically usable models, which is done in the real-world experiments presented in Subsection 6.3.2. The model derived here can be especially useful for theoretically assessing the presented timing analysis method.

Bibliography

[AB02] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[AEH75] Eralp A Akkoyunlu, Kattamuri Ekanadham, and RV Huber. Some constraints and tradeoffs in the design of network communications. In *ACM SIGOPS Operating Systems Review*, volume 9, pages 67–74. ACM, 1975.

[AJB00] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[AKR+13] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. Evaluating user privacy in Bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 34–51. Springer, 2013.

[Alm17] Karl-Johan Alm. Rate limiting via peer specified challenges (bip 154). https://github.com/bitcoin/bips/blob/master/bip-0154.mediawiki, 2017.

[Asp] James Aspnes. Notes on theory of distributed systems cpsc 465/565: Fall 2017. http://www.cs.yale.edu/homes/aspnes/classes/465/notes.pdf.

[AZV17] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. Hijacking Bitcoin: Routing attacks on cryptocurrencies. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 375–392. IEEE, 2017.

[B+09] Albert-László Barabási et al. Scale-free networks: a decade and beyond. *science*, 325(5939):412, 2009.

[BDE+13] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, and Samuel Welten. Have a snack, pay with Bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013.

[BDOZ12] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, and Aviv Zohar. On Bitcoin and red balloons. In *Proceedings of the 13th ACM conference on electronic commerce*, pages 56–73. ACM, 2012.

[BKP14] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[Blo70] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[BMC+15] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A Kroll, and Edward W Felten. Sok: Research perspectives and challenges for Bitcoin and cryptocurrencies. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 104–121. IEEE, 2015.

[BMS01] Adam Back, Ulf Möller, and Anton Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding*, pages 245–257. Springer, 2001.

[BP14] Alex Biryukov and Ivan Pustogarov. Bitcoin over Tor isn't a good idea. *arXiv preprint arXiv:1410.6079*, 2014.

[BRE+15] Alexander Beifus, Daniel Raumer, Paul Emmerich, Torsten M Runge, Florian Wohlfart, Bernd E Wolfinger, and Georg Carle. A study of networking software induced latency. In *Networked Systems (NetSys), 2015 International Conference and Workshops on*, pages 1–8. IEEE, 2015.

[BVFV17] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 1(1):22, 2017.

[CdLSJ+17] Daniel Conte de Leon, Antonius Q Stalick, Ananth A Jillepalli, Michael A Haney, and Frederick T Sheldon. Blockchain: properties and misconceptions. *Asia Pacific Journal of Innovation and Entrepreneurship*, 11(3):286–300, 2017.

[CKLR18] Mauro Conti, Sandeep Kumar, Chhagan Lal, and Sushmita Ruj. A survey on security and privacy issues of Bitcoin. *IEEE Communications Surveys & Tutorials*, 2018.

[CKS+06] Tyson Condie, Varun Kacholia, Sriram Sank, Joseph M Hellerstein, and Petros Maniatis. Induced churn as shelter from routing-table poisoning. In *NDSS*, 2006.

[CL+99] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[CLA16] Fabio Caccioli, Giacomo Livan, and Tomaso Aste. Scalability and egalitarianism in peer-to-peer networks. In *Banking Beyond Banks and Money*, pages 197–212. Springer, 2016.

[CM01] B. Carpenter and K. Moore. Connection of IPv6 Domains via IPv4 Clouds. RFC 3056 (Proposed Standard), February 2001.

[Cor75] Lee C. Cora. Stone money of yap: A numismatic survey. *Smithsonian Studies in History and Technology*, (23):1–75, 1975.

[Cor16] Matt Corallo. Compact block relay (bip 152). https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki, 2016.

[CT15] Matt Corallo and Peter Todd. NODE_BLOOM service bit (bip 111). https://github.com/bitcoin/bips/blob/master/bip-0111.mediawiki, 2015.

[Daf15] Suhas Daftuar. Sendheaders message (bip 130). https://github.com/bitcoin/bips/blob/master/bip-0130.mediawiki, 2015.

[Dai04] L. Daigle. WHOIS Protocol Specification. RFC 3912 (Draft Standard), September 2004.

[DMMK17] Debajyoti Das, Sebastian Meiser, Esfandiar Mohammadi, and Aniket Kate. Anonymity trilemma: Strong anonymity, low bandwidth overhead, low latency - choose two. *IACR Cryptology ePrint Archive*, 2017:954, 2017.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[Dou02] John R Douceur. The sybil attack. In *International Workshop on Peer-to-Peer Systems*, pages 251–260. Springer, 2002.

[DPSHJ14] Joan Antoni Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. The Bitcoin P2P network. In *International Conference on Financial Cryptography and Data Security*, pages 87–102. Springer, 2014.

[DS15] Jules DuPont and Anna Cinzia Squicciarini. Toward de-anonymizing Bitcoin by mapping users location. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pages 139–141. ACM, 2015.

[DSPSHJ+18] Sergi Delgado-Segura, Cristina Pérez-Solà, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joan Borrell. Cryptocurrency networks: A new p2p paradigm. *Mobile Information Systems*, 2018, 2018.

[DW09] Jochen Dinger and Oliver Waldhorst. Decentralized bootstrapping of P2P systems: A practical view. *NETWORKING 2009*, pages 703–715, 2009.

[DW13] Christian Decker and Roger Wattenhofer. Information propagation in the Bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[DW15] Christian Decker and Roger Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems*, pages 3–18. Springer, 2015.

[Dwo08] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[ER59] Paul Erdős and Alfréd Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[EREL17] Oguzhan Ersoy, Zhijie Ren, Zekeriya Erkin, and Reginald L Lagendijk. Information propagation on permissionless blockchains. *arXiv preprint arXiv:1712.07564*, 2017.

[ES14] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[FB99] Armando Fox and Eric A Brewer. Harvest, yield, and scalable tolerant systems. In *Hot Topics in Operating Systems, 1999. Proceedings of the Seventh Workshop on*, pages 174–178. IEEE, 1999.

[Fin] Hal Finney. The finney attack. `https://bitcointalk.org/index.php?topic=3441.msg48384#msg48384`.

[FKO+17] Giulia Fanti, Peter Kairouz, Sewoong Oh, Kannan Ramchandran, and Pramod Viswanath. Hiding the rumor source. *IEEE Transactions on Information Theory*, 2017.

[FLP85] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[FOA] Muntadher Fadhil, Gareth Owenson, and Mo Adda. Locality based approach to improve propagation delay on the Bitcoin peer-to-peer network.

[FPCS06] Michal Feldman, Christos Papadimitriou, John Chuang, and Ion Stoica. Free-riding and whitewashing in peer-to-peer systems. *IEEE Journal on Selected Areas in Communications*, 24(5):1010–1019, 2006.

[FV17] Giulia Fanti and Pramod Viswanath. Anonymity properties of the Bitcoin P2P network. *arXiv preprint arXiv:1703.08761*, 2017.

[FVB+18] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(2):29, 2018.

[GCKG14] Arthur Gervais, Srdjan Capkun, Ghassan O Karame, and Damian Gruber. On the privacy provisions of bloom filters in lightweight Bitcoin clients. In *Proceedings of the 30th Annual Computer Security Applications Conference*, pages 326–335. ACM, 2014.

[GKCC14] Arthur Gervais, Ghassan Karame, Srdjan Capkun, and Vedran Capkun. Is bitcoin a decentralized currency? *IEEE security & privacy*, 12(3):54–60, 2014.

[GKL15] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[GKRN17] Steven Goldfeder, Harry Kalodner, Dillon Reisman, and Arvind Narayanan. When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. *arXiv preprint arXiv:1708.04748*, 2017.

[GKW+16] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 3–16. ACM, 2016.

[GL12] Seth Gilbert and Nancy Lynch. Perspectives on the cap theorem. *Computer*, 45(2):30–36, 2012.

[GNH18] Matthias Grundmann, Till Neudecker, and Hannes Hartenstein. Exploiting transaction accumulation and double spends for topology inference in Bitcoin. In *5th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 2018*, 2018.

[Gra78] James N Gray. Notes on data base operating systems. In *Operating Systems*, pages 393–481. Springer, 1978.

[GRKC15] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. Tampering with the delivery of blocks and transactions in Bitcoin.

In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 692–705. ACM, 2015.

[GRPS03] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[GSY18] Diksha Gupta, Jared Saia, and Maxwell Young. Proof of work without all the work. In *Proceedings of the 19th International Conference on Distributed Computing and Networking*, page 6. ACM, 2018.

[HB96] J. Hawkinson and T. Bates. Guidelines for creation, selection, and registration of an Autonomous System (AS). RFC 1930 (Best Current Practice), March 1996. Updated by RFCs 6996, 7300.

[HC12] Mike Hearn and Matt Corallo. Connection bloom filtering (bip 37). https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki, 2012.

[HF16] Martin Harrigan and Christoph Fretter. The unreasonable effectiveness of address clustering. *arXiv preprint arXiv:1605.06369*, 2016.

[HJPS16] Jordi Herrera-Joancomartí and Cristina Pérez-Solà. Privacy in bitcoin transactions: new challenges from blockchain scalability solutions. In *Modeling Decisions for Artificial Intelligence*, pages 26–44. Springer, 2016.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[Hui06] C. Huitema. Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs). RFC 4380 (Proposed Standard), February 2006. Updated by RFCs 5991, 6081.

[IBW17] Mohsen Imani, Armon Barton, and Matthew Wright. Forming guard sets using as relationships. *arXiv preprint arXiv:1706.05592*, 2017.

[JC10] Xing Jin and S-H Gary Chan. Unstructured peer-to-peer network architectures. *Handbook of Peer-to-Peer Networking*, pages 117–142, 2010.

[JH11] Rob Jansen and Nicholas Hooper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. Technical report, DTIC Document, 2011.

[Jün15] Konrad Jünemann. *Confidential Data-Outsourcing and Self-Optimizing P2P-Networks: Coping with the Challenges of Multi-Party Systems*. KIT Scientific Publishing, 2015.

[JW18] Marco Alberto Javarone and Craig Steven Wright. From Bitcoin to Bitcoin Cash: a network analysis. *arXiv preprint arXiv:1804.02350*, 2018.

[KAC12] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[Kam11] D Kaminsky. "black ops of tcp/ip. *Black Hat USA*, 2011.

[KDF13] Joshua A Kroll, Ian C Davey, and Edward W Felten. The economics of Bitcoin mining, or Bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, 2013.

[Kin09] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.

[KKM14] Philip Koshy, Diana Koshy, and Patrick McDaniel. An analysis of anonymity in Bitcoin using P2P network traffic. In *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 469–485. Springer Berlin Heidelberg, 2014.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[KL18] Merve Can Kus Khalilov and Albert Levi. A survey on anonymity and privacy in bitcoin-like digital cash systems. *IEEE Communications Surveys & Tutorials*, 2018.

[KNbA+15] Eytan Katzav, Mor Nitzan, Daniel ben Avraham, PL Krapivsky, Reimer Kühn, Nathan Ross, and Ofer Biham. Analytical results for the distribution of shortest path lengths in random networks. *EPL (Europhysics Letters)*, 111(2):26006, 2015.

[KTO17] Kota Kanemura, Kanemura Toyoda, and Tomoaki Ohtsuki. Design of privacy-preserving mobile bitcoin client based on γ-deniability enabled bloom filter. In *Personal, Indoor and Mobile Radio Communications(PIMRC), 2017. IEEE 18th International Symposium on*. IEEE, 2017.

[Lat08] Chris Lattner. Llvm and clang: Next generation compiler technology. In *The BSD Conference*, pages 1–2, 2008.

[Law14] Averill M. Law. *Simulation Modeling and Analysis (Int'l Ed)*. McGraw Hill Higher Education, 2014.

[LCP+05] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, and Steven Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, 7(2):72–93, 2005.

[LLJS13] Mingfeng Lin, Henry C Lucas Jr, and Galit Shmueli. Research commentary-too big to fail: large samples and the p-value problem. *Information Systems Research*, 24(4):906–917, 2013.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[LW16] Eric Lobrozo and Pieter Wuille. Segregated witness (peer services). https://github.com/bitcoin/bips/blob/master/bip-0144.mediawiki, 2016.

[Max13] Gregory Maxwell. Coinjoin: Bitcoin privacy for the real world. `https://bitcointalk.org/index.php?topic=279249`, 2013. Accessed: 27.09.2016.

[MBB12] William Mendenhall, Robert J Beaver, and Barbara M Beaver. *Introduction to probability and statistics*. Cengage Learning, 2012.

[MBK+17] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, Christopher Cordi, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning, 2017.

[MD05] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005.

[Mer80] Ralph C Merkle. Protocols for public key cryptosystems. In *Security and Privacy, 1980 IEEE Symposium on*, pages 122–122. IEEE, 1980.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[MHG18] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. `http://www.cs.bu.edu/~goldbe/projects/eclipseEth.pdf`, 2018.

[Mil16] Andrew Miller. *Provable Security for Cryptocurrencies*. PhD thesis, 2016.

[MIP+06] Harsha V. Madhyastha, Tomas Isdal, Michael Piatek, Colin Dixon, Thomas Anderson, Arvind Krishnamurthy, and Arun Venkataramani. iPlane: An Information Plane for Distributed Services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 367–380. USENIX Association, 2006.

[MJ15] Andrew Miller and Rob Jansen. Shadow-bitcoin: Scalable simulation via direct execution of multi-threaded applications. *IACR Cryptology ePrint Archive*, 2015:469, 2015.

[MLJ14] Andrew Miller and Joseph J LaViola Jr. Anonymous byzantine consensus from moderately-hard puzzles: A model for bitcoin. 2014.

[MLP+15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Discovering Bitcoin's public topology and influential nodes. `https://cs.umd.edu/projects/coinscope/coinscope.pdf`, 2015.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[MPJ+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 *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[NAH15] T. Neudecker, P. Andelfinger, and H. Hartenstein. A simulation model for analysis of attacks on the Bitcoin peer-to-peer network. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1327–1332, May 2015.

[NAH16] T. Neudecker, P. Andelfinger, and H. Hartenstein. Timing analysis for inferring the topology of the Bitcoin peer-to-peer network. In *2016 Intl IEEE Conference on Advanced and Trusted Computing (ATC)*, pages 358–367, July 2016.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[NC17] Arvind Narayanan and Jeremy Clark. Bitcoin's academic pedigree. *Communications of the ACM*, 60(12):36–45, 2017.

[Neu18] Till Neudecker. Bitcoin cash (bch) sybil nodes on the bitcoin peer-to-peer network. Technical Report 4, 2018.

[New10] Mark Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[NG16] Christopher Natoli and Vincent Gramoli. The balance attack against proof-of-work blockchains: The r3 testbed as an example. *arXiv preprint arXiv:1612.09426*, 2016.

[NH17] T. Neudecker and H. Hartenstein. Could network information facilitate address clustering in Bitcoin? In *4th Workshop on Bitcoin and Blockchain Research, Financial Cryptography and Data Security 2017*, 2017.

[NH18] T. Neudecker and H. Hartenstein. Network layer aspects of permissionless blockchains. *Accepted in: IEEE Communications Surveys & Tutorials*, 2018.

[Nic15] Jonas David Nick. Data-driven de-anonymization in Bitcoin. Master's thesis, ETH-Zürich, 2015.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 305–320. IEEE, 2016.

[OAB+16] A Pinar Ozisik, Gavin Andresen, GD Bissias, Amir Houmansadr, and Brian N Levine. A secure, efficient, and transparent network architecture for Bitcoin. Technical report, UMass Amherst, Tech. Rep. UM-CS-2016-006, 2016, 2016.

[PD16] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments. *draft version 0.5*, 9:14, 2016.

[PGES05] Johan Pouwelse, PaweThl Garbacki, Dick Epema, and Henk Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *International Workshop on Peer-to-Peer Systems*, pages 205–216. Springer, 2005.

[PSCVMV15] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3):925, 2015.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 643–673. Springer, 2017.

[Res10] Certicom Research. Sec 2: Recommended elliptic curve domain parameters. `http://www.secg.org/SEC2-Ver-1.0.pdf`, 2010.

[RH10] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[RH13] Fergal Reid and Martin Harrigan. An analysis of anonymity in the Bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[Rip01] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[Ros60] Jo Bo Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the society for industrial and applied mathematics*, 8(1):181–217, 1960.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full Bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[Sch16a] Jonas Schnelli. Peer authentication (bip 150). https://github.com/bitcoin/bips/blob/master/bip-0150.mediawiki, 2016.

[Sch16b] Jonas Schnelli. Peer-to-peer communication encryption (bip 151). https://github.com/bitcoin/bips/blob/master/bip-0151.mediawiki, 2016.

[SDHD13] Matthew Sottile, Amruth Dakshinamurthy, Gilbert Hendry, and Damian Dechev. Semi-automatic extraction of software skeletons for benchmarking large-scale parallel applications. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, pages 1–10. ACM, 2013.

[SW05] Ralf Steinmetz and Klaus Wehrle. 2. what is this "peer-to-peer" about? In *Peer-to-peer systems and applications*, pages 9–16. Springer, 2005.

[SZ16] Yonatan Sompolinsky and Aviv Zohar. Bitcoin's security model revisited. *arXiv preprint arXiv:1605.09193*, 2016.

[Sza97] Nick Szabo. The idea of smart contracts. *Nick Szabo's Papers and Concise Tutorials*, 6, 1997.

[TIDH17] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, 2017(4):404–426, 2017.

[TS16] Florian Tschorsch and Björn Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials*, 18(3):2084–2123, 2016.

[TVS07] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[Wat16] Roger Wattenhofer. *The Science of the Blockchain*. CreateSpace Independent Publishing Platform, 2016.

[WG16] Karl Wüst and Arthur Gervais. Ethereum eclipse attacks. Technical report, ETH Zurich, 2016.

[WG17] Karl Wüst and Arthur Gervais. Do you need a blockchain? *IACR Cryptology ePrint Archive*, 2017:375, 2017.

[Woo14] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.

[WP17] Liang Wang and Ivan Pustogarov. Towards better understanding of Bitcoin unreachable peers. *arXiv preprint arXiv:1709.06837*, 2017.

[YMM96] Daniel Yates, David Moore, and George McCabe. *The Practice of Statistics*. WH Freeman and Company, New York, NY, 1996.