Emancipation of the Bitcoin Outcasts

Addressing Overlooked Elements of the Bitcoin Network for Improving Security and Efficiency

Federico Franzoni

TESI DOCTORAL UPF / Year 2021

THESIS SUPERVISOR
Vanesa Daza
Department Information and Communication Technologies

To my family and friends.

Thanks

I would like to express my immense gratitude to Vanesa Daza for her valuable advise. Her constant support and optimism have been of great help for overcoming the difficult moments of my doctoral adventure. Special thanks also to Roberto Di Pietro and Flavio Lombardi for initiating me to research and giving me the opportunity of doing my PhD in Barcelona.

I would also like to thank Lydia, Montse, and all the people from the secretary for helping me fight the bureaucratic machine with professionalism and a touch of good mood.

A big thanks to all my friends and colleagues, who made this journey more pleasant and entertaining. They are the real achievement of my career. A special mention for Maria, Olga, and Cecilia who added a sparkle of joy and happiness to the working environment, helping make a community out of our department.

Special thanks to all the people of room 55.210, both past and new, who kept me good company during all these years. In particular, I want to thank Pablo, Geordie, Federico, Javi, Zaira, Rasoul, Xavi, Arantxa, Alex, Conor, Sergi and Marta, who have been more like a family than simple colleagues.

A loving thought for Zaira, who with her lively friendship and tender affection won over my hearth and ended up being my wonderful companion of life.

Lastly, I want to give thanks to my family, who made me what I am today, and in special way to my caring mother, who gave me all I needed and taught me the value of knowledge and critical thinking.

Abstract

During the last decade, cryptocurrencies have revolutionized the financial industry. In these systems, participants communicate by means of a peer-to-peer protocol. Today, many of such protocols take Bitcoin as a reference model, making its study particularly important.

This thesis explores some important aspects of the Bitcoin network, related to its security and efficiency, that received limited coverage in research. Firstly, properties of the Testnet network are explored, showing they can be exploited for malicious activities. Secondly, security aspects of an open network topology are studied, arguing against the current obfuscated approach, and designing a viable monitoring system. Then, unreachable nodes are considered, showing their relevance in the network, and proposing changes to the protocol that improve efficiency and security. Finally, a new transaction relay protocol is proposed, which improves anonymity.

The results obtained show that the aspects we analyze are not sufficiently covered in research and deserve more deep investigation.

Resum

En l'última decada, les criptomonedes han revolucionat el món de les finances globals. En aquestes xarxes, els participants comuniquen a través de un protocol peer-to-peer. Molts d'aquests protocols fan servir Bitcoin com punt de referencia, fent el seu estudi especialment important.

Aquesta tesi explora alguns aspectes rellevants per la seguretat i eficiencia de la xarxa Bitcoin que han estat poc endreçats a la recerca. En primer lloc, analitzem la xarxa Testnet, mostrant com les seves proprietats poden ser explotades per activitats malicioses. A continuació, estudiem la seguretat de la topologia de la xarxa Bitcoin, promovent la seva accessibilitat i dissenyant un sistema de monitoratge. Finalment, mostrem la importància dels nodes *unreachable* a la xarxa i dissenyem nous protocols de propagació per protegir l'anonimitat de les transaccions.

Els resultats obtinguts demonstren que els aspectes analitzats no reben suficient atenció a la recerca i mereixen ser investigats més a fons.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Over the last decade we entered the fourth industrial revolution (a.k.a. Industry 4.0), with major innovations like cloud computing, big data, machine learning, AI, 5G, IoT, and blockchain [1]. All these technologies are converging and mixing into a global, interconnected network. At the same time, the number of devices connected to the Internet is also growing exponentially [2]. Not surprisingly, academic and industrial research is playing an essential role in this new context [3], and is attracting a constantly increasing amount of resources [4].

While the potential benefits of Industry 4.0 are immense, so are the risks connected to it. As noted in [5], new technologies have to face both traditional issues and new, unique security and privacy challenges. Properly addressing these challenges is of paramount importance for achieving the true potential of this revolution.

This thesis answers to this tacit call to arms for cybersecurity research by focusing on one of the biggest innovations of this century: blockchain. Thanks to its capacity of decentralizing trust and improving asset management, blockchain is being applied to a variety of use cases, including cryptocurrency, healthcare, advertising, insurance, copyright protection, energy, and governance [6]. In an effort to respond to different functional requirements, numerous variations of blockchain emerged. Nevertheless, all implementations share a common element at their core: an interconnected network of devices. More specifically, most blockchains work on top a peer-to-peer (P2P) network.

P2P networks are not new, and have been extensively addressed in research [7, 8, 9]. However, the characteristics of blockchain networks are fundamentally different from classic P2P implementations [10]. As such, new, specialized research is needed to shed light on the fundamental properties these networks have, so as to enable the design of adequate security measures. With this thesis we give our contribution to the study of blockchain networks by addressing some of the topics, related the security and efficiency of the Bitcoin protocol, that received

little attention in the literature.

Published in 2008 under the pseudonym of Satoshi Nakamoto, Bitcoin [11] is a digital currency (or *cryptocurrency*) and the first example of blockchain network. In the original white paper, this cryptocurrency is described as a pure P2P system, which allows parties to send money each other directly. As the most innovative feature, Bitcoin solves the problem of double-spending (i.e., the use of the same coins for two different transactions) without the need of trusted third parties. In particular, the system leverages a distributed record of transactions (the ledger), stored as a chain of blocks (or *blockchain*), which can be extended, but not modified. Alterations to the blockchain are prevented by requiring a proof of work in each block. This system allows to secure the ledger as long as the majority of computing power is held by honest parties.

The first Bitcoin implementation [12] appeared in 2009, and rapidly gained worldwide popularity [13], attracting massive amounts of capital [14]. Since then, thousands of other cryptocurrencies, have appeared [15], sharing similar goals and characteristics. Many of these alternative systems are directly derived from Bitcoin by tweaking its source code to meet different features or criteria. However, in most cases, the underlying P2P protocol has been adopted with little or no modification [10]. In other words, a vast number of cryptocurrencies implement the same network layer as Bitcoin.

This fact is of utmost importance, as numerous studies have highlighted how issues in this layer can affect the security and efficiency of the whole system [16, 17, 18, 19]. In particular, several attacks are possible at this level, that ultimately allow adversaries to subvert the protocol by performing denial-of-service, double-spending or even to tamper with the blockchain. Furthermore, these attacks be used to detect the source of a transaction in the network, thus leading to user deanonymization.

While tackling these problems per se is not trivial, this task is made even harder by the lack of solid theoretical background. In fact, as previously mentioned, these networks are fundamentally different from other P2P networks, due to the differences in their goal and requirements. As a result, a twofold limitation is present, with respect to the related literature. Firstly, state-of-the-art research on P2P systems does not cover some topics that are relevant to cryptocurrency networks. Secondly, previous results for P2P networks might not be valid in this new context.

In this thesis, we try to narrow the gap by exploring aspects of the Bitcoin network that have little or no coverage in previous research. We particularly focus on security concerns at different levels of the protocol. Although specific to Bitcoin, most results obtained in this work can be generalized for all the blockchains that adopt a similar network protocol. Therefore, our results can serve as a basis for further discussion and, hopefully, lead to a general improvement of the efficiency

and security of blockchain networks.

1.1 Contributions

We tackle different concerns of the Bitcoin network layer, such as the Testnet network, the openness of the Bitcoin network topology, the role of unreachable nodes, and the anonymity of transaction propagation. In the following we review our contributions for each of these topics.

TESTNET The first topic addressed by our work is the Testnet network. Testnet is a network used by Bitcoin developers to test new features or software. Despite, being designed as a simple testing environment, Testnet runs the same protocol as the main Bitcoin network, with small, but relevant, differences that makes it a unique blockchain.

We show that Testnet can have its own, unique, and potentially malicious, use cases, by designing a communication protocol for a botnet. Our results demonstrate that Testnet deserves more careful attention by developers and researchers.

More specifically, we make the following contributions:

- We analyze the characteristics of Testnet as a blockchain, stressing its differences from Bitcoin Mainnet;
- We show how Testnet can be misused for botnet communications and what advantages it provides;
- We design and implement a practical botnet Command and Control protocol that allows the registration and management of infected devices to the botnet, and enables bidirectional, encrypted communication with high resiliency and virtually no cost.

The above contributions led to the following conference publication:

> F. Franzoni, I. Abellan, V. Daza. Leveraging Bitcoin Testnet for Bidirectional Botnet Command and Control Systems. In: *International Conference on Financial Cryptography and Data Security*, 2020 [20].

NETWORK TOPOLOGY The second topic we study in this thesis is the network topology. In particular we look at the security concerns related to its knowledge. In fact, the current Bitcoin protocol implements measures to hinder obtaining this information. However, no solid proof has been given to support this approach. Additionally, the obfuscation of the topology hinders accurate measurements of the network, thus preventing to detect structural problems that affect efficiency and security [21].

We argue that the risks of making the topology public are limited, while its knowledge has a potentially beneficial impact on the maintenance of the network. Although several inferring techniques have been proposed in the past, no reliable system has ever been proposed to obtain Bitcoin topology information in a reliable way. We then propose a viable system that allows a set of semi-trusted monitors to reliably infer the network topology and monitor changes over time.

More specifically, we make the following contributions:

- We empirically review the security issues, as found in research, that are related to the knowledge of network topology;
- We highlight the potential benefits of a publicly-discoverable topology and mention favorable opinions in the literature;
- We design AToM, a viable protocol for inferring and continuously monitoring the topology of the Bitcoin network;
- We theoretically analyze the proposed protocol, and evaluate its effectiveness through experiments.

The above contributions led to the following manuscript, currently submitted and under review:

> F. Franzoni, X. Salleras, V. Daza. AToM: Active Topology Monitoring for the Bitcoin P2P Network.

UNREACHABLE NODES Another subject we address in this thesis is that of unreachable nodes. These are nodes that cannot accept incoming connections, typically due to being behind NAT or firewall. Because of the inability of monitoring tools to reach all of them, the vast majority of research studies in the literature do not take them into account. However, it has been shown that they count up to 90% of the whole network [22].

We stress the relevance of unreachable nodes for the network, and propose targeted protocol changes to improve efficiency and security. Additionally, we design a transaction propagation protocol that explicitly leverage unreachable nodes to protect user anonymity.

In particular, we make the following contributions:

- We study the importance of unreachable nodes in the Bitcoin network;
- We show their role is underestimated and overlooked in research;
- We propose simple changes to the protocol that could improve the efficiency of the network;
- We highlight how unreachable nodes are inherently protected from certain attacks;
- We design and analyze ReAP, a transaction relay protocol that leverage this property to protect anonymity.

The above contributions led to the following conference publication:

> F. Franzoni, V. Daza. Improving Bitcoin Transaction Propagation by Leveraging Unreachable Nodes. In: *3rd IEEE International Conference on Blockchain*, 2020 [23].

TRANSACTION PROPAGATION The last topic we cover in this thesis is the anonymity of transaction propagation. Recent studies [24] showed that gossip propagation protocols leak information about the source of transactions. To solve this issue, a new protocol has been proposed [25] that breaks the symmetry of the initial broadcast to hinder deanonymization. To date, this is the only example of such an alternative protocol.

In this thesis, we show that alternative designs are possible. An example of this is the ReAP protocol, mentioned above. We then generalize and improve this protocol while formally and experimentally proving its effectiveness against deanonymization attacks.

In particular, we make the following contributions:

- We propose a new, improved version of the ReaP protocol, called Clover which simplifies the design while improving anonymity;
- We formally analyze the new protocol against an eavesdropper adversary;
- We provide a proof-of-concept implementation and evaluate its performance in a simulated environment;
- We compare it to the Diffusion protocol and show it reduces the deanonymization precision of the adversary up to ten times.

The above contributions led to the following manuscript, currently submitted and under review:

> F. Franzoni, V. Daza. Clover: an Anonymous Transaction Relay Protocol for the Bitcoin P2P Network.

1.2 Thesis Outline

In Chapter 2, we explain the building blocks of this work: the blockchain and the Bitcoin protocol. In Chapter 3, we highlight some relevant aspects on the efficiency of the Bitcoin network, and we review state-of-the-art research on Bitcoin network security.

Chapter 4 explores important features of Bitcoin Testnet and show how it can be used to implement a powerful and resilient communication channel for botnets. Chapter 5 tackles the problem of topology inference and proposes a monitoring

system for the Bitcoin network. Chapter 6 highlight the importance of unreachable nodes and proposes targeted changes to the network protocol aimed at improving efficiency and security. Chapter 7 describes and evaluate Clover, a new propagation protocol that protects transaction anonymity.

Finally, in Chapter 8, we indicate possible paths for future work and conclude.

Chapter 2

PRELIMINARIES

In this chapter, we give a comprehensive overview over Bitcoin and its main security concerns. We first provide a brief introduction to blockchain technologies. Then we explain the details of the Bitcoin protocol, with particular emphasis on the underlying P2P network and its communication protocol. Finally we review the most relevant security issues and known attacks.

2.1 Blockchain

A blockchain [26, 27] is a continuously-growing distributed database, or ledger, containing a list of transactions, and shared among nodes of a peer-to-peer network. Transactions and ledger are validated by all nodes independently, without using any central authority. In particular, nodes agree on the content of the ledger by running a consensus protocol, and use such content to validate new transactions.

Transactions represent changes in the global status of the system, like the change of ownership of an asset or a modification to some variable. When a new transaction is created, it is broadcast to all nodes in the network.

To be added to the ledger, transactions have to be validated and included in a new block to append to the blockchain. This operation is done by special nodes called *validators*, or *miners*. Similarly to transactions, when new blocks are created, they are broadcast to all nodes in the network.

Each node stores a local copy of the ledger, and updates it with new blocks received from the network, according to the protocol rules.

2.1.1 Data Structures

Although the content of transactions and blocks strongly depends on the protocol, some characteristics are common to all blockchains.

TRANSACTIONS Transactions are the basic data unit of a blockchain. A transaction usually contains a timestamp and the digital signature of the user that created it. This signature guarantees the authenticity of a transaction and allows other nodes to verify the validity of the operation. Moreover, it ensures non-repudiation, meaning that the creator cannot deny being the author of the transaction.

LEDGER Technically speaking, the blockchain is a linked list of blocks, each containing a list of transaction records. Each block also contains a timestamp and the hash of the previous block in the chain. This hash field represents the link of the block to its *parent*. Each block is linked to exactly one parent block. The first block in a blockchain, known as the *genesis block*, is created when bootstrapping the network, and has no parent block. A depiction of the blockchain data structure is shown in Figure 2.1.

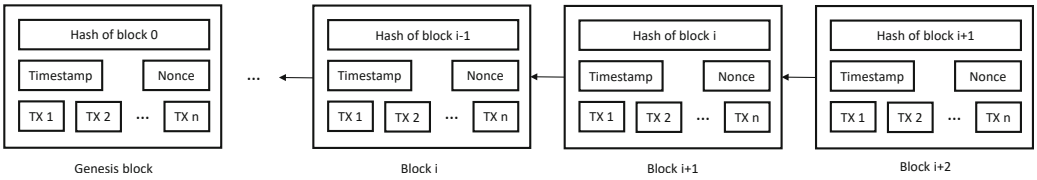

Figure 2.1: Blockchain Structure [28]

2.1.2 Consensus and Mining

In blockchain networks, blocks are distributed over the network on a best-effort basis [11]. This means that propagation delays and packet loss can occur [29]. In particular, each message can reach nodes at different times and in variable order [30]. Additionally, as in other distributed systems, network communications can be interrupted, and nodes can crash or behave maliciously.

Due to these facts, the local copies of the ledger, held by different nodes can be inconsistent with each other [16]. To solve this issue, a fault-tolerant *consensus protocol* [31] is used, which allows nodes to agree on a common, unique, version of the ledger. Specifically, nodes agree on which blocks are included in the ledger and in which order.

FORKS Although each block has only one parent, it can occur that multiple blocks are created, which are linked to the same parent. When this happens, parallel chain branches, known as *forks*, can grow.

Forks can be accidental or intentional. Accidental forks happen when two miners create a block approximately at the same time [32]. Since only one branch can be valid, this type of fork lasts for a limited period of time, and is resolved when the network eventually agrees on one of the two branches. This phenomenon is called *eventual consistency*. Blocks in the discarded branch are known as *orphan blocks*, and are not considered part of the ledger.

Intentional forks correspond to changes in the protocol and can be distinguished in *hard forks* and *soft forks* [33]. Hard forks occur when, after a protocol update, a part of the network follows a different set of rules than the rest of the network. In this case, the two chains can keep evolving independently, thus creating a new, separate, blockchain.

Soft forks occur when a protocol update is forward-compatible, meaning that blocks created under the new rules are also considered valid under the old rules. This term was introduced to indicate protocol changes that do not force obsolete nodes out of consensus. However, no real fork is generated in the blockchain.

2.1.3 Properties

Blockchain systems allow decentralized storage and verification of virtually any kind of transaction. Additionally, its design provides a number of useful properties, the most relevant of which are *immutability* and *auditability* [28].

Records in a blockchain are considered to be immutable, or more precisely, hard to modify [34]. In fact, since each block contains the hash of the previous one, it is not possible to alter or delete records in a block without having to change all subsequent blocks. In particular, the more the blocks added after some record, the harder this is to modify. This property is common to all blockchains, with very few exceptions [35], and it is considered to be a cornerstone characteristic of such systems.

Public verifiability, or auditability, is another fundamental property of blockchains. In fact, all users in a blockchain network have access to the ledger, and can verify the authenticity and validity of each record, since transactions are timestamped and digital signed. Nonetheless, the level of transparency and privacy can vary among different systems [36].

Depending on the protocol, properties like decentralization, openness, transparency, anonymity, and traceability, can be fine-tuned to adapt to specific use cases [37].

2.1.4 Types

At a macroscopic level, blockchains can be divided into three categories: *permissionless*, *permissioned*, and *private* blockchains [36].

Permissionless, or public, blockchains are completely open and distributed. Everyone is allowed to freely join and leave the network at any time. Similarly, any node can validate and maintain the ledger by running the consensus protocol. Being so, these systems operate under unknown and untrusted nodes, and need to be resistant to misbehaving or malicious nodes. Bitcoin and Ethereum [38] are the most known example of permissionless blockchain.

Permissioned blockchains are a hybrid between public and private. They allow to incorporate many parties, but the main nodes are selected at the beginning. This kind of blockchain is suitable for semi-closed systems, usually consisting of a consortium of enterprises. Joining the network is usually conditioned by access-control. Nonetheless, given their relatively openness, these systems normally have some degree of fault tolerance. Hyperledger Fabric [39] and Ripple [40] are examples of permissioned blockchains.

Private blockchains are closed, trusted environments, where nodes need to be authorized by a central authority in order to join the network. Similarly, the ledger is validated by a predefined group of nodes. These kind blockchains have very low decentralization and no transparency, but are generally more efficient and more secure.

2.1.5 Uses

Being originally developed for money transfer, blockchain is still widely used for cryptocurrencies, like Bitcoin [41], or Monero [42].

Nevertheless, over time, blockchain evolved into a general-purpose framework which can be applied to a variety of contexts [43], such as Smart Contracts and Decentralized Applications, asset trading, supply-chain management, healthcare, electronic voting, and countless more.

2.2 Bitcoin

Bitcoin is the first and most widespread cryptocurrency [44]. It is a permissionless blockchain, that is, users can freely join the network by using any compatible software running the Bitcoin protocol [45].

Although some documentation can be found online [46], the full Bitcoin protocol is only described by the source code of the reference client implementation, Bitcoin Core [47]. In particular, it is worth noting that many details of the proto-

col can only be deducted by studying its sources, greatly hindering its study and analysis.

In this section, we give an outline of the Bitcoin protocol, as described by both online sources and the reference implementation. Following the example in [48], we divide the protocol in three layers: application, consensus, and network. Note however that no such separation is defined in the protocol itself, and many of the aspects described can be transversal to these three layers.

2.2.1 Application Layer

Bitcoin allows users to send and receive coins, called *bitcoins* (BTC), directly to each other. Transfers are made by means of *transactions*, typically using a *wallet* software.

Ownership of bitcoins is established by means of *addresses* and *keys*. Each user can create one or more addresses, each controlled by a unique key, and use them to receive coins. When coins are sent to a specific address, they can only be spent by using the corresponding key. In particular, this key is used to digitally sign the transaction spending such coins.

To ease the management, users make use of *wallet* software, which store keys and automate the creation of transactions. In the following we explain in detail how this works.

2.2.1.1 Addresses and Keys

To create an address, a pair of asymmetric cryptographic keys is generated, which are stored by the wallet software. The private key is used to digitally sign transactions, while the public key is used to generate the corresponding address. More specifically, the address is created by passing the public key through a cryptographic hash function. Bitcoin addresses are strings of numbers and letters, starting with the digit "1", like $1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy$.

KEY GENERATION To create the cryptographic key pair, first the private part is generated at random. Then, the public part is calculated from the private key using elliptic curve scalar multiplication. Elliptic curve is an asymmetric cryptographic scheme based on the discrete logarithm problem, expressed by operations on the points of an elliptic curve. To obtain the public key, the private key is multiplied with a constant point known as *generator*, which is publicly known. The relationship between private key, public key, and Bitcoin address is shown in Figure 2.2. Note that both the elliptic curve scalar multiplication and the cryptographic hash are one-way functions, which means it is computationally infeasible to calculate

from the output the input that generated it. This makes it virtually impossible to find the private key from a Bitcoin address.

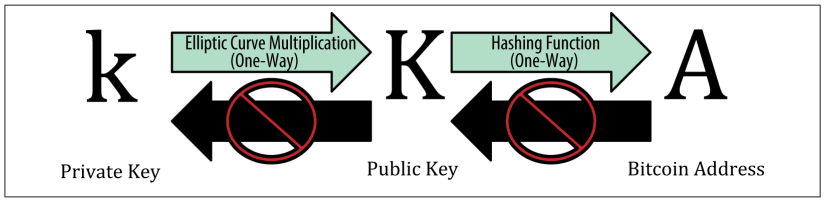

Figure 2.2: Relationship between private key, public key, and Bitcoin address [33]

The generation of the keys is completely independent from the Bitcoin protocol, and can be done by using any compatible tool, without any knowledge of the blockchain or existing addresses. Typically, the wallet software is used for this purpose.

PAY-TO-SCRIPT HASH (P2SH) In 2012, a new type of address, known as *pay-to-script hash* (P2SH), was introduced to expand the functionality of the address field. Differently from traditional Bitcoin addresses, P2SH addresses begin with the digit "3", and are generated from a transaction script, instead of a private key. Such a script defines who can spend the coins sent to the corresponding address. In contrast to P2SH, traditional addresses are also known as pay-to-public-key-hash (P2PKH).

The most common use of P2SH is the multi-signature address, which requires multiple digital signatures in the transaction in order to spend some coins. Specifically, the script is designed to require M signatures out of N keys (this scheme is known as *M-of-N multi-sig*).

2.2.1.2 Transactions

Transactions are the basic operation in Bitcoin, describing coin transfers among users.

LIFECYCLE When a user wants to make a transfer, it creates a transaction indicating what coins he wants to spend and the address of the recipient user. He then digitally sign the transaction (to prove he actually owns them) and sends it to its peers in the network. Each node validates and propagates the transaction until it reaches all nodes in the network. Eventually, it is verified by some validator node, known as *miner*, and included in a new block of the blockchain.

CONFIRMATIONS Once included in a block, a transaction is said to be *confirmed*, meaning that it has been considered valid by the network and added to the ledger. Each new block appended on top of such a block implicitly accept it as valid, thus adding an extra confirmation to the transaction. The higher the number of confirmations, the harder it is for a transaction to be reverted. In other words, once the transaction reaches an enough number of confirmations, it is considered to be permanently part of the ledger. In Bitcoin, 6 confirmations are typically considered the minimum number to safely consider a transaction as accepted.

SPENDING A TRANSACTION Technically speaking, a transaction does not send bitcoins. Instead, it locks them with the address (i.e., the public key) of the recipient, making them spendable only by using the corresponding private key.

As a consequence, to spend some coins, they first have to be unlocked. This is done by adding a digital signature, which proves the knowledge of the private key, and hence the right to spend these coins.

STRUCTURE The transaction data structure contains all the necessary data to unlock some funds and lock them with a new public key.

In particular, it lists one or more *inputs* and one or more *outputs*. Each input consists of a previous (unspent) transaction output and the digital signature to unlock it. On the contrary, each output consists of an amount value and a locking script the Bitcoin address of the recipient.

A depiction of the Bitcoin transaction data structure is shown in Figure 2.3. Transferred amounts are indicated in *satoshis*. The satoshi (sat) is the smallest fraction of a Bitcoin that can be spent, and corresponds to 0.00000001 BTC.

It is worth noting that transactions are uniquely identified by their hash.

VALIDATION In order to be valid, transactions must meet three conditions:

1. The unlocking script of each input must return true (i.e., the digital signature is valid);

2. The sum of the values of the outputs must be equal or smaller than the sum of the values of the inputs;

3. All inputs have to correspond to unspent outputs of previously accepted transactions, also known as Unspent Transaction Outputs (UTXOs).

Condition 3 allows to determine, in any moment, the set of all spendable outputs, which is known as *UTXO Set*.

Bitcoin nodes verify transaction inputs by combining their unlocking script with the locking script of the corresponding UTXO. The resulting script is then

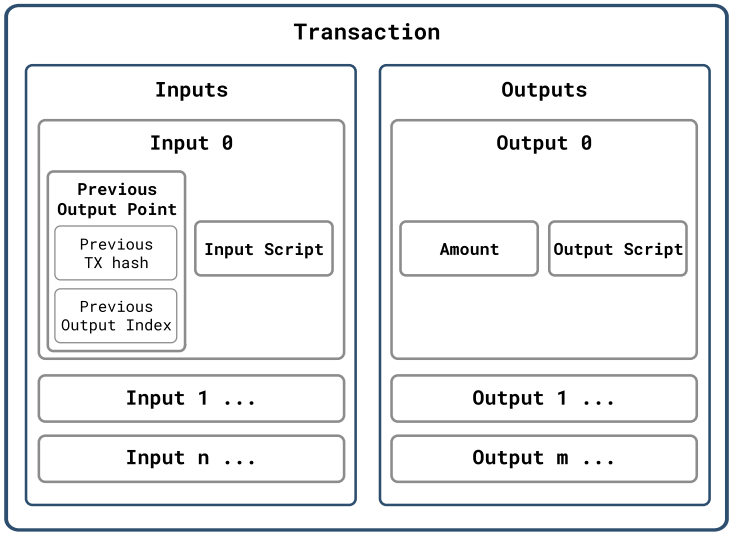

Figure 2.3: Bitcoin Transaction Structure [49]

executed during the verification process. If the execution raises errors or returns a non-true result, the transaction is considered invalid and rejected.

CHANGE OUTPUTS AND FEES Given the fact that UTXOs can only be spent once, they are an indivisible unit. In other words, UTXOs are like banknotes: they can only be spent entirely.

If only a fraction of an UTXO has to be transferred, a so-called *change* output can be included in the transaction, which transfers the remaining coins back to the sender. To that purpose, an existing address can be used or a new one can be created. For privacy reasons, Bitcoin wallets typically generate, if needed, a *change address* for each transaction.

Note that, given the validation rules, it is always possible to send an amount which is smaller than the sum of the inputs. However, the difference between the input amount and the output amount is considered to be a *fee* for the miner that includes the transaction to the blockchain (see Section 2.2.2.2). Consequently, if no change output is included, the whole remainder will be given to the miner. As such, a typical Bitcoin transaction will include a change output that correspond to the remainder of the transferred amount, minus the fee the user is willing to pay.

DUST OUTPUTS When including transactions to a new block, miners tend to give priority to those paying higher fees, as this means more profit for them. For

this reason, although fees are not mandatory by protocol, they are usually included in every transaction. In fact, most nodes will refuse to propagate transactions that do not include a high enough fee (or no fee at all).

In particular, a minimum fee is required by most nodes to accept and propagate a transaction, which is equal to 546 satoshis [50]. This amount is calculated as the fee required to spend a basic transaction with one input and one output.

As a consequence, outputs that are less than 546 are considered as uneconomic *dust*, because they are too small to be spent in a following transaction. Dust outputs are not considered as valid in Bitcoin and thus transactions including them are rejected by the network.

ORPHAN TRANSACTIONS As described above, each transaction input corresponds to an output in a previous transaction. In particular, each transaction is virtually linked to (at least) a parent transaction and a child transaction, thus forming a chain.

Nonetheless, given the asynchronous nature of the network, it is sometimes possible that a child transaction is received before its parent, thus impeding its validation. Such a transaction is known as *orphan transaction*.

Instead of being rejected, orphan transactions are collected by nodes and validated as soon as their parents are received.

OP_RETURN Since 2014, it is possible to embed a small amount of data inside a transaction, using the OP_RETURN opcode [51]. This possibility was introduced to discourage other wasteful methods of embedding data, such as using non-existing transaction output addresses. The new opcode allows adding a non-spendable output, which carries up to 80 bytes of arbitrary data. OP_RETURN is often used to implement asset exchange protocols on top of Bitcoin or to add valuable data in the blockchain [52].

2.2.2 Consensus Layer

Nodes in the Bitcoin network implement the *Proof-of-Work* (PoW) consensus protocol to agree on a common version of the blockain. In this section we first describe the block data structure used in Bitcoin and then explain PoW consensu.

2.2.2.1 Blockchain

The Bitcoin blockchain structure is not very different to the one described in Section 2.1, containing the creation timestamp and the hash of the parent block. Additionally, the Bitcoin block has two important field:

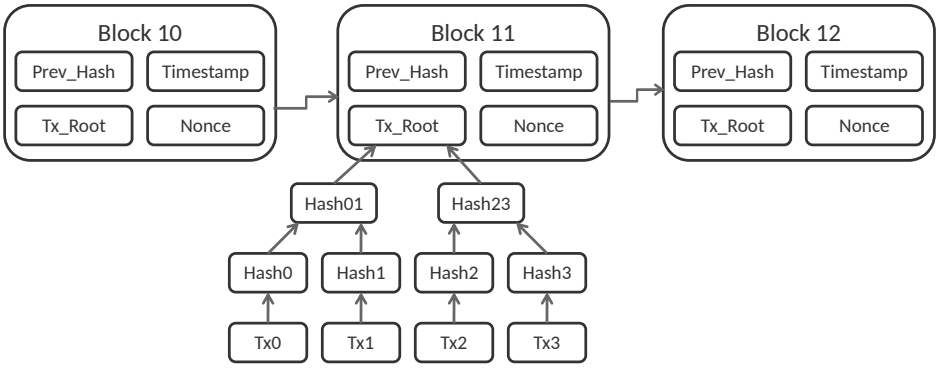

Figure 2.4: Bitcoin Block Structure [54]

- The *nonce*: this field contains the solution to the PoW problem and it shows the miner performed a certain amount of computation to create the block (see Section 2.2.2.2);
- The *Merkle root* [53]: this is the root of a binary hash tree having as leaves the transactions contained in the block, while inner nodes contain the hash of their children. As the root of the tree depends on all leaves, this field is used to efficiently summarize and verify all transactions in the block.

A depiction of the Bitcoin block structure, including the transactions Merkle tree, is shown in Figure 2.4.

Similarly to transactions, blocks are uniquely identified by their hash. In particular the block identifier is obtained by hashing the block header (which contains all data except transactions) twice using the SHA256 algorithm. Another way to identify a block is by its position in the blockchain (where the Genesis block has position 0). The Bitcoin blockchain is sometimes represented as a vertical stack, resulting in the use of the term *height* to indicate the block number. Similarly, the terms *tip* or *top* are used to indicate the most recently added block in the chain.

2.2.2.2 Mining and Consensus

In Bitcoin, the process of creating new blocks is known as *mining*. Mining has a twofold effect: first, it creates new coins; second, it secure the ledger against invalid transactions and double spending.

Bitcoin mining is based on *proof-of-work* (PoW): validator nodes, called *miners*, dedicate their processing power to create new blocks in exchange of coins. In turn, other nodes agree on the chain of blocks that required more computation to be created. In the following, we describe in detail how this works.

PROOF-OF-WORK In order to create a block, miners have to solve a cryptographic problem based on cryptographic hashing and whose solution depends on the block contents. Specifically, a value, called *nonce*, has to be found, which, hashed with the rest of the block, outputs a number smaller than a certain target.

Since a lower value implies a smaller number of possible solutions, the smaller the target the harder it is to find a suitable nonce. Additionally, as cryptographic hash functions are unpredictable (the only way to know the output for a given input is to execute the hash function), miners can only find a solution by brute-forcing. That is, all possible values have to be tried until the output is below the target. As a result, creating a block necessarily requires a significant computing effort. For this reason, the nonce value in a block is also called a *proof of work*.

DIFFICULTY Since the hash output has a fixed number of digits, the target value is usually expressed as the minimum number of trailing zeroes that the nonce must have. In fact, the more the number of trailing zeros, the lower the value; for instance, $0002f$ is smaller than $00a62$.

The target is automatically adjusted over time to create, on average, one block every 10 minutes: if, over a time slot, this average is below 10 minutes, the target the number of trailing zeroes is increased (i.e., mining is made harder); on the contrary, if the average is over 10 minutes, this number is decreased (i.e., mining is made easier).

COMPETITION AND REWARDS Miners compete against each other to create new blocks. The competition makes it unpredictable to determine in advance which miner will create the next block. This effectively randomizes block creation, securing the ledger from possible manipulations.

To incentivize competition, rewards are assigned to miners creating new blocks (if accepted to the main chain). Two types of rewards exist. In first place, miners are given the coins generated by the new block (this is known as *block reward*). The number of new coins per block started as 50 BTC and is halved every 210.000 blocks, corresponding to approximately 4 years. This number is currently equal to 6.25 BTC.

Besides the block reward, miners collect the fees of all transactions in the block. As described in Section 2.2.1.2, although fees are not strictly mandatory, a minimum amount is usually required to have transactions spread and mined. Additionally, as block rewards decrease, fees become increasingly important. In particular, when all coins will be issued (this is expected to happen after the year 2140), miners will only earn from transaction fees.

For this reason, when deciding which transactions to include in a block (this number is limited by the maximum size of the block, which is currently 1 MB),

miners usually give priority to those paying higher fees. As a consequence, transaction fees are constantly increasing over time.

CONSENSUS Since miners compete to create blocks, it is possible that two blocks of the same height (i.e., at the same position in the blockchain) are created. This happens when two blocks are mined approximately at the same time.

When this happens, two competing blocks will be spread through the network, generating a fork (see Section 2.1). To decide which chain to consider as the valid one, Bitcoin nodes (including miners) always choose the longest one known.

The intuition behind this behavior is that a longer chain requires a bigger amount of computation to be created. Assuming the majority of computing power is controlled by honest miners, choosing the longest chain ensures that dishonest miners cannot subvert the blockchain. In fact, to do so, a malicious actor should be able to produce blocks faster than the others. As long as this is not true, the honest chain will always be longer than the malicious one.

Note that, at the same time, each node in the network independently verifies all transactions and all blocks, thus guaranteeing that only valid block can be added to the blockchain.

ORPHAN BLOCKS Similarly to transactions, the asynchronous propagation of blocks in the network can make some block reach a node before its parent. These blocks, known as *orphan blocks*, are usually kept by nodes for some time, in case their parent is received and they get to be part of the main chain.

MINING POOLS Due to the competitive nature of mining, processors used for proof-of-work have become more and more specialized over time. As a result, not every node is able to participate to the competition. Instead, only those using specialized hardware can currently mine a block in Bitcoin.

Given the costs of the mining process and the low chances of winning the race of producing a new block, working alone is hardly profitable. For this reason, miners join their forces by creating *pools*.

In a mining pool, several miners work in parallel to solve the PoW for the next block. If any of them succeeds and the produced block is added to the main chain, the rewards are split among all miners in the pool. This mechanism allows to amortize the risks, thus making profits more stable over time. Currently, joining a pool is the only way for a miner to make its activity profitable.

2.2.3 Network Layer

The P2P network is the backbone of all communications in Bitcoin. Information is propagated by all nodes and distributed to (almost) the entire network. This includes transactions, blocks, and all auxiliary data used to build and maintain the network itself. For this reason, the network layer is a core component of the whole system.

2.2.3.1 Node Types

Nodes are usually distinguished between *full nodes* and *lightweight nodes*.

Full nodes store and maintain an up-to-date copy of the full blockchain. This allows them to independently verify all transactions and blocks. By replicating the whole ledger, these nodes contribute to the overall security of the network. Full nodes may or may not implement wallet functionalities, depending on the user needs.

Lightweight nodes implement wallet functionalities but do not maintain a full copy of the blockchain. Instead, they download all block headers but not the transactions included in each block. By not storing the full ledger, these nodes can be run by resource-constrained devices, such as mobile phones. However, lightweight nodes cannot independently verify transactions and rely on full nodes to receive relevant parts of the blockchain. Transactions are verified using a method called *Simple Payment Verification* (SPV). For this reason, they are also known as *SPV nodes*. SPV allows to verify the presence of a transaction in a block by checking the branch of the Merkle tree that contains it.

SPV nodes can thus verify a transaction of interest is being included in the blockchain, and how many times it has been confirmed. Nonetheless, they cannot ensure that a double-spending transaction of the same UTXO exists because they have no access to the full ledger. Despite this vulnerability, SPV nodes are secure enough for most practical purposes and provide the right balance between practicality and security.

2.2.3.2 Node Connectivity

Independently from their functionality (Full or SPV), nodes can also be categorized by their network-level characteristics. Currently, three types of network addresses are supported in Bitcoin: IPv4, IPv6, and OnionCat [55].

Unsurprisingly, IPv4 nodes are by far the most common in the network, accounting for roughly the 70% of the network [56]. In contrast, IPv6 nodes are the least common, representing only the 10% of the network. OnionCat addresses correspond to nodes using Tor [57] and account for the 20% of the network.

UNREACHABLE NODES A further categorization of Bitcoin nodes is based on their reachability, that is, the ability of other nodes to open an outgoing connection towards them. When a node is able to accept incoming connections it is said to be *reachable*. Otherwise, it is called *unreachable*.

Although nodes can explicitly reject incoming connections, the most common reason for the unreachability of a node is its position in the network. In particular, if the node resides behind a firewall or in a NAT network [58], it is not possible for other nodes to reach them directly.

For this reason, most unreachable nodes have IPv4 addresses. In fact, IPv6 is specifically designed to avoid the use of NATs. At the same time, OnionCat nodes run *Tor hidden services* [59], which makes them reachable even when behind NAT.

Differently from other nodes (see Section 2.3.1.1), unreachable nodes only establish outgoing connections. This has an influence on different aspects, such as network topology (Section 3.1) and data propagation.

Nevertheless, Bitcoin does not explicitly distinguish between reachable and unreachable nodes at the protocol level. In Chapter 6, we show how unreachable nodes impact the efficiency and security of the Bitcoin network, and, by explicitly differentiating their behavior, we propose targeted improvements to the protocol.

2.2.3.3 Testnet

Bitcoin *Testnet* is a network for testing purposes. It is completely separate from the main network (called *Mainnet*), thus making it an ideal environment for testing new features and running experiments.

Unlike Mainet, Testnet coins (tBTC) have no value in the real world. In fact, they can be obtained for free through online services, known as *faucets*.

Mining is also easier in Testnet, which makes the production of blocks much faster. This helps experimenting with the blockchain more efficiently. Transactions are also confirmed more quickly, making them quickly usable for new experiments.

From a more technical perspective, Testnet runs the same protocol as Mainnet. However, some restrictions are ignored to allow developers to test edge cases. In particular, differently from Mainnet, non-standard transactions are allowed, thus being relayed and mined by the network.

In Chapter 4, we will see these differences in detail and show their implications from a security perspective. Our findings show Testnet is not just a simple test network but a full-fledged blockchain with potential applications.

2.3 The Bitcoin P2P Network Protocol

In the previous section, we gave a general overview over the Bitcoin protocol. In this section, we describe the network protocol in detail, which will be the focus of this thesis. Given the complexity of the protocol, only the most relevant parts are described. While most information has been taken from [60] and [33], many implementation details are extracted from research papers and the source code of Bitcoin Core [47].

PROTOCOL MESSAGES Several network messages are used in the Bitcoin network protocol. We here give a reference list of those mentioned in this section. Their use will be further explained later.

- `version`: exchanged to establish a new connection, it contains information about the client, such as protocol version, services, IP addresses, and so on;
- `verack` (version Ack): sent in response to the `version`;
- `ping`/`pong`: exchanged to keep the connection alive;
- `getaddr` (get address): it requests a list of known active peers;
- `addr` (address): it contains a list of IP:port tuples (maximum 1000 addresses), corresponding to known nodes. It can be unsolicited or sent in response to `getaddr`;
- `inv` (inventory): announces known transactions or blocks, it contains a list of hashes (not the actual data);
- `getdata`: requests the data of a single block or transaction by hash;
- `tx` (transaction): sent in response to `getdata`, it contains a single transaction data;
- `block`: sent in response to `getdata`, it contains a single block data;

2.3.1 Peer Management

Being a decentralized network, peer management is a crucial aspect of the Bitcoin protocol.

2.3.1.1 Connection Management

The first thing a node has to do to join the network is finding other nodes. This phase is knonw as *bootstrapping* and is especially tricky when running a node for the first time, as no previous connections have been done.

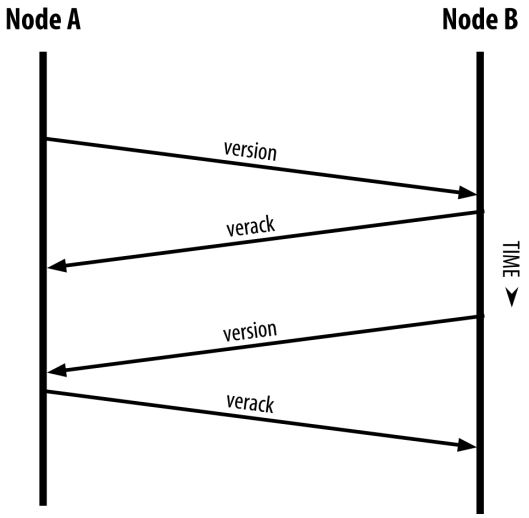

Figure 2.5: Connection establishment [33]

PEER DISCOVERY Upon the first execution, the Bitcoin client tries different methods to locate nodes in the network.

First, *DNS seeds* are used. These are standard DNS servers run by known members of the Bitcoin community. When queried, these servers return a list of IP addresses where a node is currently running. The servers usually run crawlers to discover new nodes and maintain their lists updated. In case the DNS method fails, the client program has an hard-coded list of addresses pointing to known stable nodes.

When connecting to a node, more addresses are requested using a `getaddr` message. When the client is closed, the list of known nodes is stored to a file. Following executions will load addresses from this file to join the network.

ESTABLISHING CONNECTIONS Once the list of potential nodes has been loaded, the client select 8 nodes at random and establishes a connection to each of them.

To establish a connection, a three-way handshake is done. Let us assume node A wants to connect to node B. The following sequence of messages is exchanged:

1. Node A sends `version` to Node B;

2. Node B sends `verack` and `version` to Node B;

3. Node A sends `verack` to Node B.

The initial handshake is shown in Figure 2.5.

INBOUND AND OUTBOUND Although all connections are bidirectional, Bitcoin distinguishes between *outbound* connections, which are established by a node towards its peers and *inbound* connections, which are established by peers towards the node. Similarly, peers called inbound and outbound, depending on the type of connection.

Each node connects to 8 outbound peers and accepts up to 117 inbound connections, for a total maximum of 125 peers. Note that while outbound connections are always established (i.e., all nodes establish 8 outgoing connections), the number of inbound connections depend on the reachability of the node (see Section 2.2.3.2) and its stability over time.

CONNECTION MANAGEMENT After establishing the connection, the nodes send a `ping`/`pong` messages every 30 minutes to check if the peer is still active. If no message is received over 90 minutes, the peer is considered inactive, and the connection is closed.

Additionally, as an anti-DoS protection, nodes implement a reputation system for their peers. Specifically, for each peer a penalty score is maintained. When a malformed message is received from a peer, its reputation is decreased. If the score reaches the value of 100, the peer is marked as *misbehaving* and banned for 24 hours.

Whenever an outbound connection is closed, a new one is established. The candidate node is chosen among entries in the local database of known addresses (see Section 2.3.1.2). Peers are selected using a pseudorandom procedure to give the network more dynamism and a more randomized structure. In this procedure, fresh peers (i.e., recently discovered addresses) are preferred over old ones. Generally speaking, the selection of new connection depends on how addresses are stored and managed. In the next section, we explore this aspect in more detail.

2.3.1.2 Address Management

As previously mentioned, connection management and, hence, the network connectivity are heavily dependent on how addresses are managed. In the following, we describe how addresses are advertised through the network and how they are managed by nodes.

ADDRESS PROPAGATION AND DISCOVERY When connecting to a peer for the first time, most nodes advertise their own address with an `addr` message. At the same time a list of known peer addresses is asked to the new peer, by sending a `getaddr` message. The list is then received via `addr` messages and stored into an address database (see Section 2.3.1.2). This database is used both to answer

getaddr requests and to establish new outbound connections (e.g., because a peer disconnected).

When replying to a `getaddr` message, up to the 23% of the addresses stored in the database is sent, but no more than 2500 addresses. On the other hand, when an `addr` message is received, the node decides for each address whether to forward it to its peers or not. This decision depends on the number of addresses received and the their timestamp (addresses that are older than 10 minutes are not forwarded). When forwarding, reachable addresses are forwarded to two neighbors, while unreachable ones are forwarded to one neighbor only. In this context, the reachability of an address is simply determined by its network family (see Section 2.2.3.2). Specifically, a node considers an address reachable if they both belong to the same family, and unreachable otherwise.

For each peer, the node remembers which addresses have been forwarded, so as to avoid repeating the transmission. This history is kept for each connection (i.e., per session, not per IP) and it is cleared every 24 hours.

THE ADDRESS DATABASE: TABLES AND BUCKETS The address database store IP:port tuples together with a timestamp that helps to evaluate the freshness of an address.

The database is divided into two tables: the `tried` table, which stores addresses to which the node has previously connected to, and the `new` table, which contains addresses advertised by other peers (i.e., received via `addr` messages) or received from DNS seeds.

Tables are further organized in groups, called *buckets*. The `tried` table consists of 64 buckets, each containing up to 64 addresses. Each address is mapped to a bucket by hashing it with a random number, and its subnet, or *group*, that is, the /16 prefix for IPv4, /32 prefix for IPv6, or the first 4 bits for OnionCat addresses. In particular, every address is mapped to a single bucket, and each group maps to up to four buckets.

When a node connects to a new peer, its address is added to the `tried` table. If the designated bucket is full, *eviction* is used: four random addresses from the bucket are selected, and the oldest one is replaced with the new peer's address. The replaced address is moved to the `new` table.

The `new` table consists of 256 buckets, each containing up to 64 addresses. The assignment of the bucket for each address depends on the both its group, as defined for the `tried` table, and its *source group*, that is, the group of the IP address of the peer or DNS seed from which the node learned the new address. Each (*group*, *source group*) pair maps to a single bucket, while each group maps to up to 32 buckets. If a bucket is full, then a *terrible* address is selected to be replaced. An address is called terrible if: (1) its timestamp is more than 30 days

old or more than 10 minutes in the future, (2) has 3 consecutive failed connection attempts. If no terrible addresses are in the bucket, then eviction is used, this time discarding the replaced address. Note that the same address can be mapped to different buckets is advertised by different peers.

The total number of addresses a Bitcoin peer can store is limited by 20480.

SELECTING NEW PEERS When a new outbound connections has to be established, an address is selected from the database according to a probability function that depends on the the number of current number of outbound peers and the ratio between the size of the `new` table and the `tried` table. In particular, the new peer is more likely to be selected from `tried` if there are few outgoing connections and the `tried` table is large. Moreover, fresh addresses have more probability to be selected.

2.3.2 Data Propagation

In Section 2.2.3, we mentioned how the distribution of information in the Bitcoin network is essential for a correct functioning of the protocol. In this section, we describe in detail how core data (transaction and blocks) is propagated through the network.

Bitcoin uses a gossip-like protocol [61] where new data is spread through the network by flooding nodes. In other words, as soon as a node receives a new transaction or block, it transmits it to all its neighbors.

3-STEP TRANSMISSION To avoid transmitting the same data to a node that already knows it, nodes use a 3-step process. First, the transaction or block is announced by sending an `inv` message containing its hash. The receiver checks whether the hash is known, and, if not, it requests the corresponding data with a `getdada` message. Finally, the transaction or block data is transmitted using a `tx` or `block` message, respectively. This process (for a transaction) is shown in Figure 2.6.

TRANSACTIONS PROPAGATION PROTOCOL Earlier, we said that data is propagated as soon as it is received. However, this is not the case for transactions, whose actual transmission is randomized for security reasons (see Section 3.3.1.2). In particular, nodes maintain a queue for each peer, containing the transactions to be forwarded. A loop is then executed to periodically flush these queues (i.e., to send all transactions that are in the queue). This step follows a specific protocol, as described in the following.

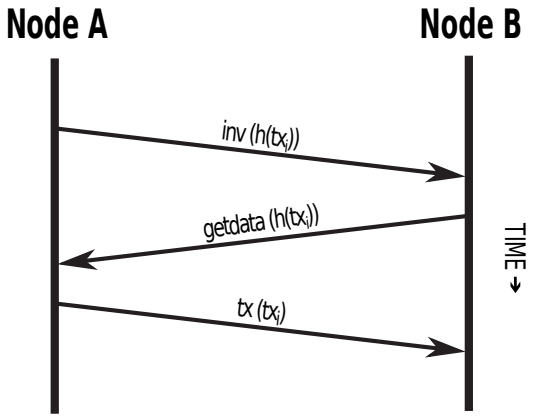

Figure 2.6: 3-step transaction transmission [33]

Until 2015, the *Trickle* protocol was used. In this protocol, at each loop execution a random node is selected, called *trickle node* and its queue is flushed.

Starting from 2015, Bitcoin switched to the *Diffusion* protocol. In this protocol, the queue for each node is flushed with an independent, exponential delay based on a Poisson distribution [62].

Note that for similar security concerns, the same protocol is used for propagating nodes addresses (see Section 2.3.1.2).

BLOCKCHAIN SYNCHRONIZATION When a node joins the network for the first time or it rejoins it after leaving for some time, it will have its ledger out of sync. To synchronize the ledger among peers, when a new connection is established, the currently known tip of the blockchain is exchanged. The one with the lower tip then sends a `getblocks` message, requesting all missing blocks, to which the peer replies with an `inv` message containing the corresponding block hashes. Finally, the node requests blocks data, one by one, using `getdata` messages.

This procedure is done at every connection so as to have all nodes synchronized. Once synced, nodes will request and forward new blocks as they are announced by its peers.

Chapter 3

EFFICIENCY AND SECURITY OF THE BITCOIN NETWORK

In this thesis, we aim at improving security and efficiency aspects of the Bitcoin P2P network. In particular, we do it by taking a different look at some of the components that have been poorly covered by state-of-the-art research. In this chapter, we give review the most relevant studies concerning the Bitcoin network, mentioning some of the shortcomings we address in our work.

3.1 Network Structure

The Bitcoin network is one of the largest in terms of number of nodes [63], counting for a daily average of approximately 11.000 reachable nodes [56]. Its topology has been extensively studied in research [16, 64, 65, 21]. These studies also show the presence of a massive amount of unreachable nodes (see Section 2.2.3.2) in the network, whose number is still unknown. The above-mentioned analyses hardly give a precise account, since they do not distinguish between offline nodes and nodes that are actually out of reach. However, results show that the number of unreachable nodes is much greater than reachable ones. In particular, estimates go from 10 [66] to 30 times more [22]. This means unreachable nodes count for approximately 90% of the whole network.

Technically speaking, Bitcoin is an *unstructured P2P network* [67] designed to have a random topology, so as to maximize decentralization. Each node establish the same number of outgoing connections (see Section 2.3), selecting at random among known peers. However, this number is not enforced by any means, which means that nodes can potentially have any number of connections. In fact, it has been shown that supernodes actually maintain a number of connections way beyond the limit [21, 66]. Similarly, measuring tools are able to connect to all

reachable nodes of the network.

Furthermore, given the high number of unreachable nodes, which cannot accept incoming connections, the topology of the Bitcoin network is relatively centralized in its reachable portion, with 10% of the nodes accounting for the totality of incoming connections. Due to this fact, the reachable part of the network is central to communications in Bitcoin and is generally considered more relevant for data propagation.

As a side effect, despite representing the 90% of the network, unreachable nodes are left at the margins of research studies ([23]). In Chapter 6, we show how these nodes can and should be a relevant part of the network, explicitly levraging them involving to improve the efficiency and security of transaction propagation.

3.2 Data Propagation

Data propagation is of utmost importance in a distributed network like Bitcoin, and its efficiency can be decisive for the security of the system. While not the main focus of this thesis, network efficiency will be a relevant part of our discussion. We here give a brief resume of important aspects regarding the network.

3.2.1 The Scalability Problem

One of the most relevant issues related to data propagation is the so-called *scalability problem* [68]. Specifically, this issue is related to the time needed for a block to propagate to all nodes in the network. This time determines the window of opportunity for forks to occur. In fact, the probability of forking directly depends on the ration between block propagation time and block production rate. In particular, the higher the propagation time the higher the probability of forking.

Currently, Bitcoin supports around 3 transactions per second (TPS), which is far inferior to centralized currency transaction rates [69]. To increase the TPS value two strategies are possible: (1) increase the block production rate, and (2) increase the block size to store more transactions. However, in the first case, more blocks are produced during the propagation window, leading to a higher number of forks. In the other case, an increase in the block size means a longer propagation time (because more data has to be verified and transmitted), leading again to more forks.

For instance, increasing the block size by 100 to obtain 300 TPS would likely make the propagation time longer than the interval between blocks. In such a situation, forks would occur at a higher rate than blocks themselves, resulting in the consensus to break.

Several mitigations have been proposed to the scalability problem [70], among which: sharding [71], relay networks [72], and payment channels [73]. However, none of these seem to definitely solve the problem [74]. In this perspective, it should be clear that data propagation plays a key role for the scalability of the network.

RELAY NETWORKS Relay networks are among the first attempts to mitigate the scalability problem. The basic idea is to improve block propagation among miners by tightly connecting them to each other in fast networks.

The first proposal has been the Bitcoin Relay Network [75] which consisted of 9 nodes strategically distributed around the globe. Miners could connect to the geographically-closest relay nodes to improve transmission and reception of blocks. The main downside of this network was his high centralization, since it was controlled by a single person.

The Bitcoin Relay Network was replaced by the more decentralized FIBRE (Fast Internet Bitcoin Relay Engine) [76], which redesigned the relay protocol to reduce the impact of packet loss. Statistics show that FIBRE transmission rates are nearly as fast as the speed of light.

Falcon [77] is another alternative developed at Cornell University. Its design speeds up block transmission by relay the IP packets it is composed of before verifying the validity of the whole block. In other words, instead of waiting for the whole block to be received, verified and retransmitted at each hop, Falcon nodes forward packages as soon as they are received. This leaves space for dishonest miners to take advantage of the missing verification step to inject invalid packages only to waste competitor resources. To mitigate this risk, gatekeepers are used, thus introducing a point of centralization.

3.3 Security of the Bitcoin Network

All attacks in Bitcoin are aimed at one of the following objectives:

- Denial of Service (DoS): the goal is to disrupt the ability of using Bitcoin; it can be aimed at a single node, a subset, or the whole network;
- Double spending: the goal is to use the same coins more than once, which means either creating money out of thin air, or deceiving a victim into accepting, in exchange of goods, money that he will never actually receive; this is one of the core issues that Bitcoin, as a digital currency, is designed to solve;
- Deanonymization: the goal is to determine which person or entity owns a specific amount of coins or performed a specific transaction; although

Bitcoin uses random pseudonyms, methods exist to trace back money to real identities;
- Profit: this goal is specific to mining attacks, where a miner tries to cheat in order to gain more than the other miners; this kind of attack harms concurrency, which is a vital feature of PoW systems.

In this section, we divide Bitcoin attacks into two categories: *blockchain-level* and *network-level*. We give a general overview of blockchain-level attacks, while describing network-level attacks in greater detail, as this will be the main focus of this thesis.

3.3.1 Blockchain-level Attacks

These attacks can be grouped into two types: consensus attacks, aimed at cheating the consensus protocol, and deanonymization attacks, aimed at unveiling real users identity.

3.3.1.1 Consensus attacks

These attacks work by deviating from the PoW protocol with the goal of gaining control over the blockchain or increasing profit.

SYBIL ATTACKS In a general distributed setting, the Sybil attack [78] consists in forging multiple identities so as to act like different identities.

In the context of Bitcoin, identities are represented by addresses. Since each user can create addresses at will, with no cost, it is actually trivial to perform a Sybil attack. Nonetheless, Bitcoin avoid risks by means of PoW: since the creation of blocks is linked to computational power, the attacker has no gain in forging multiple identities.

However, at the network level, as we will see in Section 3.3.2, other, more effective forms of Sybil attacks are possible.

51% ATTACK This is the best known and feared attack against PoW consensus, since it breaks the original assumption of an honest majority of computing power [11].

In other words, the 51% attacks simply consists in controlling the majority of mining capacity. In fact, with such a power, an attacker could produce blocks at a faster rate than the rest of the network, thus being able to create a longer blockchain (which would be accepted by nodes as the right one).

This way, it would be possible for the attacker to modify a past block, thus breaking the immutability property. In particular, this ability could be used to

exclude a transaction from the ledger (DoS), or to double spend by reverting a previous transaction.

Such an attack is also possible with less than the 50% of computing power, but the success rate will be below 100%. For instance, with 40% of the mining capacity it is possible to overcome a 6-deep confirmed transaction with a 50% success rate [79].

Note however that even controlling a majority of computing power it is not possible to produce invalid blocks (e.g. containing a double-spending transaction), since nodes would not accept them in their ledger. This means that, to double spend, the attacker have to first spend a transaction, then modify the blockchain to revert it, and then spend it again.

Bitcoin (partially) protects from 51% attacks by using *checkpoints* [80], that is, blocks that are hardcoded into the reference client as safe points in the blockchain. In particular, nodes will not accept blocks at a lower height than the last checkpoint, even if a whole longer chain is received.

Given the difficulty and cost (mining consumes power energy), and the relatively small benefits, 51% attacks are considered to be not very profitable and hence unlikely to happen [81].

SELFISH MINING First reported by Eyal and Sirer [82], selfish mining is a dishonest strategy where one or more colluding miners deviate from the protocol to increase their profit.

The strategy works as follows: when the selfish miner finds a new block, it keeps it private instead of publishing it, thus effectively creating a secret fork. The miner then keeps working on its private branch while honest miners work on the public chain. If the private fork becomes longer than the public chain, the miner keeps it secret. Conversely, when the length of the public chain is approaching the private one, the selfish miner reveals it to the network. Since the newly published fork is longer than the main chain, nodes will immediately switch to it, this giving all the relative rewards to the selfish miner.

This has a twofold effect. First, it gives selfish miners a share of revenues that exceeds their proportion of mining power. Second, it wastes all the computational effort made by honest miners.

This attack proved that Bitcoin is vulnerable to attacks by a miner (or pool) controlling only 1/3 of the total mining power, a portion substantially lower than the theoretical 50% assumption.

The selfish mining strategy can be generalized and combined with network-level attacks to further increase gains [83]. Furthermore, the competitiveness of a selfish pool can lure honest miners to join it, thus leading to a possible 51% attack.

3.3.1.2 Anonymity attacks

The goal of these attacks is to link one or more transactions in the blockchain to real-world identities. In fact, despite common belief, Bitcoin is not anonymous. A certain degree of privacy is provided by the pseudonymity of Bitcoin identities, which can be created at will and even changed at each transaction. However, every transaction in the blockchain is publicly accessible and traceable. As such, identifying the owner of one address is sufficient to reveal the whole transaction history of an user.

TRANSACTION GRAPH ANALYSIS The most typical approach to deanonymizing user identities is to analyze the blockchain content, clustering addresses according to their relation to each other. In fact, as explained in Section 2.2.1.2, all transaction inputs correspond to outputs in previous transactions, thus forming a unique chain.

Based on this property, it is possible to build a *transaction graph* [84], where all transactions in the blockchain are linked to each other by analyzing inputs and outputs. Then, different heuristics can be used to cluster addresses together [85]. For instance, all inputs in a transaction are considered to belong to the same user [86, 87, 88]. Another simple heuristic is to considered a previously-unused address in the output as the change address (see Section 2.2.1.2) [87, 89].

Such clusters can be joined with external information on how certain addresses were used (e.g. to buy goods) to extract the whole transaction history of a given user [89, 86].

MIXING SERVICES After the first deanonymization techniques were disclosed, online anonymization services emerged, called *mixing services* or *tumblers* [90].

Using these services, Bitcoin users can mix, shuffle, and redistribute their coins so as to break their traceability. In other words, mixers break the link between some coins and the users who purchased them. Numerous mixing services are currently available in the wild [91].

3.3.2 Network-level Attacks

In this section, we describe in detail the most relevant and well-known attacks targeting the Bitcoin network protocol.

3.3.2.1 Sybil Attacks

In P2P networks, a sybil attack consists in the use of multiple nodes (or more specifically, IP:port tuples) by the same adversary [91]. This is actually a specific

case of Byzantine nodes [92] cooperating against a common target.

Bitcoin, as a permissionless blockchain network, is especially vulnerable to this kind of attack [18]. In particular, an attacker can (1) deploy an arbitrary number of nodes, and (2) open multiple connections from a single node towards the same node. Sybil attacks are functional to other network-level attacks, such as deanonymization (see Section 3.3.2.6), partitioning (see Section 3.3.2.3), and eclipse (see Section 3.3.2.5).

In most cases, the practical objective of a sybil attack is to control the connections of one or more target peer. In this respect, there is an important difference between incoming and outgoing connections. In fact, in a P2P network the attacker can easily open an arbitrary number of connections towards a known, reachable target. However, she cannot control which nodes the target connects to. To increase the chances of a target opening a connection towards her, the attacker needs to deploy several different nodes [59].

For this reason, incoming connections are less trustworthy than outgoing connections [18], and are thus treated differently in Bitcoin. Nonetheless, apart from this differentiation, very few countermeasures are implemented in Bitcoin, to mitigate the risk of sybil attacks. One of these countermeasures is to limit the number of connections towards addresses in the same IP range [18]. The reason for this approach lies in the assumption that a single adversary is unlikely to control IPs from many different ranges. Based on the same assumption, bucketing (see Section 2.3.1.2) is used to divide the addresses used to establish new outbound connections.

3.3.2.2 Double Spending

As explained at the beginning of this section, double spending is one of the possible targets of any attack in Bitcoin.

At the network level, this can be obtained by sending a transaction to a target node while broadcasting another one, spending the same coins, to the rest of the network. In fact, a node receiving two transaction spending the same output, will accept the first one to be received, and reject the other one. The attacker succeeds if she manages to make the victim accept a transaction in exchange for goods, while having the rest of the network accept (and mine) the double-spending transaction.

A typical setting for this kind of attack is that of fast-payments scenarios, where the time between the acceptance of a transaction and the transfer of goods is short (e.g. in a shop). Note that in this scenario, the victim needs accept zero-confirmation transactions (i.e. a transaction without any confirmation), because confirmation times in Bitcoin are too long (10 minutes, on average for the first one). In other words, the victim only relies on just receiving the transaction from

the network, which typically occurs within few seconds from it was sent. This event is supposed to prove that the transaction has been broadcast to the network and thus considered sufficient by the victim to believe the transaction has been made.

Let us denote the victim's transaction with tx^V and the double-spending one with tx^A. Two conditions have to be met for the attack to succeed: (1) the target node has to receive tx^V before tx^A, (2) tx^A is accepted into the blockchain. It is easy to see that a requirement for the attacker is to know the IP (or at least the network) of the victim's node.

The first example of this attack was showed by Karame et al. [93]. There, the attacker connects directly to the (reachable) victim and use helper nodes to broadcast the double-spending transaction. She then sends tx^V to the victim and, after a short time, tx^A to the helper nodes. To ensure the victim receives tx^A after tx^V, helper nodes does not connect to the victim. Furthermore, to help tx^V spread faster (which, in turn, increase the probability of having it added to the blockchain), the attacker can increase the number of helper nodes or the number of peers connected to them. Their results show that the probability of success decreases as the time between the spread of the two transactions increase. This is due to the fact that delaying the broadcast of tx^A allows tx^V to reach more nodes, thus increasing its probability of being accepted to the blockchain. In particular, if this time is 1 second and 2 helpers are used, the attack is almost guaranteed to succeed. The victim can protect himself by connecting to more nodes, and adopting a listening period to check whether a double-spending transaction is received. However, the authors claim these countermeasures can be circumvented and that additional measures are needed, such as having nodes alerting peers about double-spending transactions.

More effective countermeasures are proposed by Bamert et al. [94], such as having vendors not accept incoming connections or not relay transactions, that reduce the success rate of the previous attack to just the 0.09%.

Among other factors that influence the success rate of double-spending fast payments, information propagation speed is among the most relevant [16, 18]. On the other side, double-spending attacks can be made more effective by leveraging other attacks, such as partitioning (see Section 3.3.2.3) and eclipse (see Section 3.3.2.5), which enable the attacker to hide information from the victim node.

Generally speaking, the only secure way to avoid double spending is to wait for a transaction to be confirmed [91].

3.3.2.3 Partitioning Attacks

A partitioning attack consists in trying to split the network so as to prevent communications between two isolated group of nodes.

This attack on the Bitcoin network was first studied in [95]. The authors consider an adversary controlling a powerful botnet with Distributed DoS (DDoS) capabilities. Their attack consists of two phases. In the first phase, the botnet nodes join the network, behaving conforming to the protocol. Bots only advertise addresses belonging to the botnet, allowing to reduce the number of connections between honest nodes, which makes the attack easier. In the second phase, the attacker stops forwarding transactions and blocks, and, at the same time, performs a DDoS attack against the nodes in the *minimum vertex cut*, that is, the smallest set of nodes whose removal causes the network to split. To do so, the attacker needs knowledge of the network topology (see Section 3.3.2.7). Their analysis showed that the Bitcoin network can resists an attack lasting several hours against an adversary controlling a botnet as large as the network itself. Nonetheless, more powerful adversaries should be considered.

Partitioning attacks can be used as for DoS or simply to create distrust in the system (e.g., for speculation on the bitcoin value). Furthermore, it can be used to split miners, thus generating forks in the blockchain. In turn, this can be used to perform double spending or selfish mining. Note, however, that isolating miners is made harder by the existence of relay networks (see Section 3.2.1), which are separate from the P2P network.

3.3.2.4 Routing Attacks

While most network attacks consider adversaries at the node level, more powerful adversaries can exist at the AS-level. Several factors make Bitcoin particularly vulnerable to such an adversary. First, most nodes and miners are concentrated in just a few ASs [96, 97]. Second, the BGP routing protocol [98] can be easily hijacked [99] (that is, network traffic can be rerouted through a different AS). Finally, as communications in Bitcoin are not encrypted, an AS-level adversary can perform any MitM attacks (eavesdrop, drop, modify, inject, or delay messages).

Apostolaki et al. [96] are the first to study an AS-level adversary. In particular, they show how such an adversary can partition the network by diverting the traffic via BGP hijacking. She then uses MitM inspection to drop packets coming from and going to the target partition. Their attack does not require knowledge of the topology except the IP addresses of the target nodes.

In [100], a stealthier version of the attack is presented, which shows analogous capabilities without making any route manipulation. The attack targets reachable nodes and is particularly dangerous since it is hardly detected and leaves no traces.

As shown in [96] and later confirmed by Saad et al. [97], BGP routing attacks can not only be used for double spending but also to perform selfish mining and generate forks in the blockchain. To avoid such a risk, Apostolaki et al. [101] proposed SABRE, a secure and scalable relay network (see Section 3.2.1) that is resilient to routing attacks.

3.3.2.5 Eclipse Attacks

An eclipse attack [102] is a special case of partitioning attack in which a single node is targeted. The adversary aims at controlling, or colluding with, all the victim's peers so as to control all communication from and towards the node. In particular, information from the network can be completely hidden to the target (hence the name "eclipse").

Similarly to partitioning attacks, the eclipse attack can be used for double spending, selfish mining, or even to make 51% attacks easier [103]. Furthermore, it has been shown how combining selfish mining with an eclipse attack can further increase gains for the dishonest miner [83].

In [103], Heilman et al. show a practical attack against the Bitcoin reference client that allows monopolizing all the connections of a reachable node. To do so, they connect to the victim, fill its `new` table with bogus addresses and its `tried` table with addresses controlled by the adversary. Upon restart, the victim will then establish all 8 outbound connections (see Section 2.3.1.1) with high probability. At this point, the adversary occupies all inbound connections to completely isolate the target.

Their attack exploits the eviction mechanism of the address database (see Section 2.3.1.2) and the fact that nodes accept unsolicited `addr` messages. To perform this attack, the adversary needs a large number of IP addresses, which can be obtained by using a botnet. Their results showed it was possible to eclipse a node with at least 85% probability using a small botnet of 4600 bots and that the attack often succeeded even with just few hundred adversarial nodes. Nevertheless, if the victim stores enough legitimate addresses, it cannot be eclipsed regardless of the number of IPs controlled by the adversary.

3.3.2.6 Deanonymization Attacks

As mentioned in Section 3.3.1.2, Bitcoin transactions can be linked to real world identities by inspecting the blockchain and crossing it with publicly-available information. Similarly, at the network level, it is possible to deanonymize a transaction by detecting the node of the network that generated it. In fact, knowing the IP address of the device where a transaction was created often means identifying who did it. While this is nearly impossible with past transactions, an adversary

can perform real-time monitoring to determine the source of a specific transaction message.

The attack can be directed against a single node or it can be generalized. In the first case, the adversary aims at determining which transactions have been created by the chosen target. This can be done trivially (with respect to the deanonymization part) by eclipsing the node and intercepting all the transactions it sends.

Conversely, network-wide deanonymization, which aims at determining the source of all new transactions, requires more complex strategies. The first such strategy was proposed by Dan Kaminsky [104] and it is based on the observation that the node that creates a transaction will be the first one to announce it. Based on this fact, an adversary that connects to all nodes can simply monitor all transaction announcements (i.e., `inv` messages) and then associate each transaction to the first node to announce it. By implementing this approach, Koshy et al. [105] were able to deanonymize several hundred transactions. Nevertheless, in most cases, this was only possible thanks to anomalies in the relay pattern, while normally-relayed transactions proved to be hard to deanonymize. Neudecker et al. [106] also combine observations of the message propagation with address clustering techniques (see Section 3.3.1.2). However, their results show that, for the vast majority of users, this information does not facilitate deanonymization.

This basic approach was generalized and expanded in [24]. In this work, the *eavesdropper* adversary is defined, which not only connects to all reachable nodes, but also maintains multiple connections for each node, which increases the probability of receiving a transaction from its source. To determine the origin of each transaction, an *estimator* functions are used. The first one, called *first-timestamp* or *first-spy* follows the strategy, described above, of linking each transaction to the first node that relays it. The second one, called *maximum-likelihood* estimator generalizes the previous strategy based on the so-called *rumor-centrality* [107, 108] of gossip networks: as transactions are spread symmetrically (broadcast) by each node to its peers, nodes that are close to the source of a transaction will relay it before those that are far from it.

As such, if the structure of the network graph (i.e., the topology) is known, it is possible to determine the source of a transaction by observing the order in which nodes announce it. Based on this information, the maximum-likelihood estimator determines which node is the most likely to be the source for a given transaction.

Using this adversarial model, the authors analyze the security of the Trickle and Diffusion protocols (see Section 2.3.2), showing that both provide poor anonymity guarantees. The reason for this lies in the symmetry of the transaction relay pattern.To solve this issue, a new protocol, called Dandelion [25, 109] has been proposed, which, by breaking this symmetry, offers near-optimal anonymity guarantees. In their protocol, new transactions are relayed over a single path of nodes for few hops, until they get broadcast. To increase anonymity, the relay path

is built among all nodes as a network-wide boolean circuit.

Note that the works discussed so far only focus on reachable nodes. This is because unreachable nodes, by definition, are safe against adversaries that need to connect to all target nodes. In fact, the only known deanonymization attack that targets unreachable nodes is the one by Biryukov et al. [110]. In this attack, the eavesdropper adversary is able to distinguish unreachable nodes with the same IP address (i.e., nodes behind the same NAT) by the set of their (reachable) peers. In particular, this is obtained using a fingerprinting technique that exploits the rumor centrality of the address propagation (see Section 2.3.1.2). Since this attack only works during a single session, Mastan et al. [111] propose a method that exploits block requests patterns to identify unreachable nodes over consecutive sessions. However, their technique can only be used in conjunction with other deanonymization techniques.

In Chapter 6 we propose a countermeasure that prevents the fingerprinting of unreachable nodes based on address propagation, and leverage their safer position in the network to design a new transaction relay protocol, that, like Dandelion, breaks the symmetry of propagation to improve anonymity. Furthermore, in Chapter 7, we generalize this approach to design a simpler but safer protocol.

TOR NODES ANONYMITY While Tor is specifically designed to protect the anonymity of its users, a work by Biryukov et al. [59] shows that this is not the case for Bitcoin. On the contrary, combining Bitcoin and Tor creates a vector for a man-in-the-middle attack. In turn, such an attack allows for easy deanonymization, and enables capabilities analogous to partitioning and eclipse attacks. Additionally, the attacker can fingerprint Tor nodes so as to recognize them when they decide to connect to the Bitcoin network directly.

3.3.2.7 Topology Inference Attacks

Differently from classic P2P networks, knowledge of topology in Bitcoin is particularly important for its security implications [21, 18]. In fact, many of the attacks described in this section are somehow related to the connectivity of nodes.

Dishonest miners can use topology information to gain advantage in the propagation of blocks and perform selfish mining [82]. Double-spending attacks can be facilitated by knowing the neighbors of the target node [112]. Partitioning attacks that target the minimum vertex cut would be made easier with a complete view of the network graph structure. In an eclipse attack, if the attacker knew the peers of a target node, she could try to disrupt their connections instead of waiting for the target to restart. Even deanonymization, in the case of rumor-centrality-based heuristics are enabled by the knowledge of the topology.

For the above reasons, although topology hiding is not a design goal, the Bitcoin protocol explicitly hinders the ability to discover connections among nodes. Nonetheless, several methods have been found over the years that allow to infer connections by leveraging information leakage in the protocol.

The first such method was proposed in [110] and allowed to determine the outbound connections of a node based on `addr` messages: as nodes advertise their own address when connecting to a new peer, and such messages are forwarded to other peers, it was possible to detect the target's peers by connecting to all nodes and analyzing received addresses. In [21], a network-wide technique was proposed, called AddressProbe, that allowed to infer all connections among reachable nodes. Their technique exploited the timestamp attached to each entry of `addr` messages: since the address of each outbound peer was updated every time a message was received, it was possible to determine which entries in the `addr` message corresponded to outbound peers by looking at their timestamp. Both techniques were made ineffective by a targeted change[1] in the reference client [113].

A more generic technique was proposed by Neudecker et al. [114], based on the rumor-centrality of gossip propagation. Assuming the source of a specific message is known and that the adversary is connected to all nodes, it is possible to infer connection by observing the time at which nodes propagate the information. Their technique was made ineffective by the switch from Trickle to Diffusion (see Section 2.3.2).

In [115], Grundmann et al. propose two different methods. The first one exploits the fact that nodes accumulate transactions before announcing them to their peers. In particular, an `inv` message contains all transactions received since the last `inv` message was sent. Based on this fact, the adversary creates marker transactions for all peers and observe `inv` messages to infer links. This technique shows high precision (more than 90%) but has very low recall (10%), and is hardly practical in real life. The second targets a single node, and exploits the fact that nodes do not relay double-spending transaction. To infer the target's peers, the adversary sends a different double-spending transaction to each node, except the target. Then observes which transaction the target relays and deduce a link with the node to which that transaction was sent. Although this technique has very high precision (97%) and good recall (60%), no countermeasures were introduced to date.

The most recently disclosed technique for topology inferring is TxProbe [66]. Similarly to the previous example, this technique leverages marker messages, this time based on orphan transactions (see Section 2.2.1.2). Despite its high precision

[1] Reduce fingerprinting through timestamps in 'addr' messages - https://github.com/bitcoin/bitcoin/pull/5860

and recall (more than 90%), this method is rather invasive and can interfere with the ordinary transaction propagation. Again, the technique was invalidated by a recent update in the reference client [2].

The number of techniques disclosed (and fixed) over time poses the question of whether topology-hiding is a valid approach for protecting the Bitcoin network, given the fact that knowing the topology is not a threat per-se and it hinders measurements and analysis that could be essential to improve the network efficiency and security. In Chapter 5, we advocate for an open topology and propose a viable protocol to discover connections among reachable nodes.

[2] Select orphan transaction uniformly for eviction - https://github.com/bitcoin/bitcoin/pull/14626

Chapter 4

TESTNET: MORE THAN A TEST NETWORK

Despite being designed as a simple testing environment for Bitcoin, Testnet (see Section 2.2.3.3) has its own, unique characteristics that makes it different from the main network. At the same time, Testnet is a full-fledged blockchain network, with potential applications in the real world. In this chapter, we show how its peculiarities, if properly exploited, could serve as a basis for large-scale malicious activities. We observe that such features are the result of a careless design, which stemmed from a superficial consideration of Testnet as a real blockchain.

4.1 Leveraging Testnet for Bidirectional Botnet Command and Control Systems

With the growth of the Internet of Things, millions of insecure systems have been deployed worldwide [116, 117]. Botnets benefited from this rise to increase their size and the magnitude of their attacks [118]. Recent attacks from the infamous Mirai botnet [119] showed the potential of this threat, with DDoS attacks of up to 1.1 Tbps [120]. A lot of research has been done to help detect and disrupt botnets [121], most of which focus on what is notoriously their weak spot: the Command & Control (C&C) communication channel. In fact, C&C systems are either based on centralized services or require a complex infrastructure to avoid disruption [122].

Blockchain technologies may give botnets a powerful tool to increase their resilience. Recent research showed how blockchains can be leveraged to implement the command and control (C&C) system of a botnet [123, 124]. In fact, using public blockchains like Bitcoin as the communication channel has several advantages for a botnet. First of all, as other P2P networks, they provide robust-

ness and efficiency. Secondly, they are not regulated by any authority, making them censorship-resistant, that is, no specific content or user can be banned. Furthermore, they privilege privacy, by making use of pseudonyms and hindering the association between a transaction and the device that generated it. As such, although possible (see Section 3.3.2.6), it is not trivial to detect nodes participating in a botnet. More importantly, it is hard to identify the botmaster. All these properties are ideal for botnets [125], as they allow to operate over a long period of time, with virtually no risk of having communications disrupted.

State-of-the-art solution on Bitcoin make use of transactions as the main C&C vector, following different strategies to embed messages. However, these proposals have important limitations. First of all, transactions have a cost, as they have to spend a minimum amount of coins and are required to include a fee (see Section 2.2.1.2). As such, sending messages can be expensive, and the actual cost can fluctuate according to the current coin value. Secondly, transaction-based communications are only used to send commands from the botmaster, delegating replies to external, typically server-based, channels. Finally, messages are very limited in size and are sent in clear, as cryptography is only implemented on the external channel. All these limitations make this approach rather impractical for a real-world botnet implementation.

In this chapter, we show that, using Testnet, it is possible to overcome all these limitations and implement a cost-free, bidirectional, encrypted botnet C&C system. We propose a communication protocol and analyze its viability in real life. Our results show that this approach would enable a botmaster to build a robust and hard-to-disrupt C&C system that is both practical and economical. We believe such a system represents a realistic threat, for which countermeasures should be devised.

4.2 Background

4.2.1 Botnets and C&C

A botnet is a network of infected devices, called *bots*, collectively controlled by a single actor, called the *botmaster*. Botnets have been a major threat on the Internet for a long time [126], as they are used for a variety of malicious activities, like spamming, credentials stealing, and Distributed Denial of Service (DDoS) attacks [127].

C&C COMMUNICATION In order for a botnet to operate, a communication channel is needed between botmaster and bots. The infrastructure used for that purpose is known as the Command & Control (C&C) system. This is a crucial

component for a botnet, as it is the only means to keep control over the bots. As such, it is has to be designed carefully, in order to avoid being disrupted. In other words, the C&C system should allow controlling the botnet as long as possible, providing stealthy and efficient communication between botmaster and bots.

Strategies to implement C&C changed over the years, following the evolution of available technologies and the ability of authorities to counter existing approaches [122]. First-generation botnets leverage hardcoded Internet Relay Chat (`IRC`) channels, where bots connect to receive instructions from the botmaster. This system is simple and cheap but is also easy to detect and take down [128, 129]. Second-generation botnets make use of `HTTP`, with hardcoded web domains, periodically contacted by the bots to download instructions. This approach allows to effectively blend messages into legitimate Internet traffic. Nonetheless, effective techniques exist to detect botnet communications [130, 131], allowing to quickly shutdown malicious domains [132].

Early botnets relied on a client-server model, thus having a central point of failure, which can always be detected and shut down by the authorities. Last generation botnets overcome this issue by adopting a P2P model. Bots and C&C server connect as peers to the same network, making it difficult to distinguish the source of the commands [133]. This architecture makes the botnet much more robust and hard to shut down. Nonetheless, it is still possible to detect P2P-botnet traffic using advanced techniques [134, 135]. Moreover, to join the network, bots need hardcoded addresses, which can be easily blocked by authorities if detected. Modern botnets tend to use a mix of techniques, such as P2P network with HTTP C&C server, or leverage cloud-based services and social media as rendezvous points [122]. Although these services are easy to setup and access, providers can promptly block any detected malicious account.

4.2.2 Testnet

As explained in Section 2.2.3.3, Bitcoin Testnet has some important differences from Mainnet. In particular:

- Testnet coins have no value in real life. For this reason, they can be easily obtained for free through online services called *faucets* [136].
- Mining is much easier, since the PoW difficulty is set to a lower value. As a consequence, unlike Mainnet, it is feasible to run a solo miner [137] to earn coins.
- The Testnet network and blockchain are about ten times smaller than Mainnet [66]. This makes clients synchronize faster and consume less resources.
- Non-standard transactions are validated and relayed by the network. This feature enables the following characteristics:

- OP_RETURN can be bigger than 80 bytes. In fact, there is no explicit limit to the amount of data that can be actually embedded;
- Transactions can have multiple outputs with the same address as well as multiple OP_RETURNs;
- Transaction outputs can be below the dust limit (see Section 2.2.1.2);
- Transaction size can be greater than the maximum (which is around 100kB).

All these properties give numerous benefits for the implementation of a botnet. First of all, the botmaster can easily obtain the necessary amount of coins to run its botnet, either by using faucets or running a miner. Secondly, the reduced size of Testnet blockchain and network make bots less resource-demanding, allowing them to hinder detection and even to run on low-resource devices. Finally, non-standard transactions give the ability to send bigger and more complex messages.

These features allow overcoming all the main drawbacks of previous Bitcoin-based proposals: botnet communications have no cost thanks to the fact that Testnet coins have no real value; bidirectional communication can be implemented thanks to the great number of coins that can be obtained for free; encryption can be implemented thanks to the larger amount of data that can be embedded in each transaction.

4.3 Related Work

ZombieCoin [123] was the first paper to propose Bitcoin as a means for C&C communications. Bots embed the botmaster public key and decode transactions coming from the corresponding address. To embed commands, the OP_RETURN opcode is used, which allows to carry up to 80 bytes of data. In [138] the same authors propose enhancements such as *transaction-chaining* to embed longer messages and external upstream communication by means of periodical *rendezvous-point* announcements. The main limitations of this proposal are the server-based upstream communication and the cost of messages sent on the blockchain. The authors claim that it would be impractical and economically prohibitive to implement upstream communication on top of the blockchain. We show that this is not true when leveraging Testnet.

ChainChannels [124] proposes a more generic approach, which can be used on different blockchains as it does not leverage Bitcoin-specific features. The authors describe a method to insert hidden data into transaction signature, which can be later decoded with the private key used for the signature. For this purpose, the authors propose a key-leakage scheme that allows bots to decipher messages at a later time. This is a very portable approach, since virtually all blockchains em-

ploy digitally-signed transactions with a compatible signature scheme. Nonetheless, this approach suffers from the same limitations as ZombieCoin: messages are costly and limited in size; communication is unidirectional and unencrypted. Furthermore, bots can only decrypt messages in a second moment, assuming they execute commands altogether after these have been issued, something that might not be realistic.

Differently from other blockchain-based solutions, DUSTBot [139] have nodes connect and communicate directly to each other, like in a classic P2P botnet. Special bots, called sensors, are delegated to collect responses and send them to the botmaster via OP_RETURN transactions, for which a number of UTXOs are available on the blockchain (see Section 2.2.1.2). To reduce costs, bots communicate via Testnet. However, only standard transactions are used, thus limiting the communication capacity of the botnet. Additionally, nodes have to connect to multiple networks and suffer the burdens of standard P2P botnets (bootstrapping, peer list exchange, message propagation, and so on), which other blockchain-based solution avoid.

In [140], an hybrid botnet is proposed, called LNBot, which leverages the Lightning Network (LN), a popular off-chain payment channel network for Bitcoin. In LNBot, the botmaster sends commands to a set of C&C servers via using payments over LN. In particular, covert channels are used to embed data into transactions. Each server controls a separate mini-botnet, each using its own architecture and communication channel (IRC, P2P, and so on). This hybrid architecture allows a good scalability but adds the extra complexity of managing multiple botnets. Moreover, each C&C server constitutes a single point of failure for the corresponding botnet. Finally, upstream communication is not considered. Our solution is purely blockchain-based and allows bidirectional communication.

Yin et al. [141] propose to use blockchain explorers and url shorteners to have bots retrieve commands. In particular, a URL generation algorithm is used to know the next link where to retrieve the transaction containing the command. The main drawbacks of this solution is that it is centralized (block explorers and URL shortener services are both server-based) and that new URLs can be predicted and blocked in advance, if a bot is compromised. Our solution does not use any centralized service nor generation algorithms to be stopped.

In [142], the authors propose an approach based on *Whisper*, a communication protocol that runs on top of the Ethereum network. This approach does not use transactions and thus has no cost. It also provides a good level of privacy and allows for two-way communication. Moreover, as messages are not in transactions, they are not added to the blockchain, making their backward analysis harder. However, Whisper, which is still in a PoC stage, it is not enabled by default on the standard Ethereum client (geth) and there are no known statistics about how many nodes currently run the protocol. Consequently, its reliability is

unknown, making it unlikely to be actually used by a botnet as of today.

Similarly to the previous solution, Oliveira et al. [143] makes use of Ethereum for C&C communication. In their system, smart contracts are used. In particular the botmaster deploys a smart contract containing the command instruction, while the bots can run a function on the contract to retrieve such instruction. The botmaster can also run a function to update the instruction. The authors make use of a private implementation of the Ethereum network, where only bots participate. However, such a network allows to easily detect all participating bots and can be easily disrupted by authorities. On the other hand, using the public blockchain makes running the botnet costly. Like other solutions, this proposal do not take upstream communication into account.

For what concerns detection, [144] shows that it is indeed possible to distinguish bot-produced transactions from legitimate ones. Nonetheless, blockchain systems are very unlikely to adopt any censorship measure to block such transactions, since this would contradict one the main goals of blockchains. Instead, preventive measures should be taken to hinder botmasters from taking advantage of blockchain networks.

4.4 Our C&C Protocol

We propose a viable communication protocol for Testnet, based on non-standard transactions, that provides a bidirectional and encrypted C&C channel at zero cost.

As in previous works, we assume there exists an infection mechanism that takes control of devices and downloads the bot client. The botnet is composed by a C&C server node, directly controlled by the botmaster, and a number of bot nodes. We assume the C&C server is not resource-constrained and runs a full node. On the other side, bots run an SPV node to consume less resources and hinder detection.

In the rest of this section, we explain how the communication works (transactions, fees and encryption) and describe the different phases of the protocol (registration, commands and responses).

4.4.1 Communication

All communications between the botmaster and the bots happen through transactions. Note that, by using this mechanism, bots do not need to connect to each other, nor to deal with the propagation of messages. Instead, being embedded into transactions, messages are relayed by all nodes in the network, thus reaching all bots.

DATA EMBEDDING AND FEES We use OP_RETURN outputs to embed messages inside transactions. As previously mentioned, this operator has no explicit limits of size on Testnet. As such, the amount of data that can be embedded is only limited by the maximum size of a transaction, which, again, is not explicitly limited on Testnet. This makes the theoretical size limit bound by the size of a block (around 1 MB). However, a practical limit to this amount is given by the minimum fee needed to have the transaction relayed by other peers. This value is known as the *minimum relay fee* (MRF). MRF does not differ between Mainnet and Testnet and is proportional to the size of the transaction itself. This means that, although sending very large messages is possible, this can be excessively expensive in terms of fees. We will see more details about MRF later in Sections 4.6 and 4.5.

In our protocol, all transactions spend a fee equivalent to the corresponding MRF. To this respect, it is important to notice that using low fees might make the transaction mined later. However, from the botnet perspective, it is not important if and when messages are added to the blockchain, but only if they travel across the network and reach the C&C server.

ENCRYPTION AND AUTHENTICATION In order to protect communications, we use encryption in both directions. To obtain the best compromise between security and efficiency, we make use of an hybrid approach.

We assume the botmaster creates an asymmetric key pair, called *botmaster keys* before the creation of the botnet and hardcode bots with the public key. This key pair is completely unrelated to the address used to send commands, which in fact, can change at every message. Additionally, a symmetric key is also embedded in the bots, called *botnet key*.

For the sake of clarity, we distinguish between *downlink* encryption, used from the botmaster to the bots, and *uplink* encryption, used by the bots to communicate with the botmaster.

Downlink encryption works this way: when the botmaster wants to send a command, it encrypts it with the botnet key and signs it with its private key; when bots receive a transaction with an OP_RETURN, they check the signature using the botmaster public key. If the signature is valid, they decrypt the message with the botnet key and execute the command. This scheme allows the bots to recognize transactions from the botmaster even without knowing its address. Moreover, thanks to the signature, bots are assured about the authenticity of the source.

For uplink encryption, each bot creates a private symmetric key, called the *bot key*, which is sent to the botmaster at the time of registration, encrypted with the botmaster public key. When sending messages, bots encrypt data with their bot key. Furthermore, bots use a new address for each message, which corresponds to

the change address of the previous transaction. In order to recognize and decrypt bots messages, the botmaster keeps track of the current address of each bot and the corresponding encryption key.

TRANSACTIONS We have the following types of transactions: *quotas*, *registrations*, *fundings*, and *messages*. Quotas have one input and several outputs (the quotas), which are used as input for the registration transactions. Registration transactions have one input (a quota) and one OP_RETURN output. The quota equals the MRF for the registration message, so no change output is required. Funding transactions have one input and one output, which equals the input value minus the MRF. Messages (commands and responses) always have two outputs, one with the OP_RETURN carrying the message and the other sending the change (minus the MRF) to another address belonging to the sender (i.e. the *change address*).

4.4.2 Bot Registration

When a new bot joins the network, the first thing it needs is to get some funds to send transactions. As the bot cannot obtain funds autonomously (like the botmaster does), it needs to ask the botmaster to provide some. However, at the same time, the botmaster needs to know the address of the bot in order to send such funds.

We solve this problem by having all bots sharing a common private key, that gives access to all transactions of an address called the *shared account*. The botmaster periodically puts funds on the account, while new bots use such funds to register to the botnet. They do so by sending a *registration message* which contains their own address and encryption key. Since SPV clients do not store the UTXO set (the set of unspent transactions), they ask their peers about any available fund on the account. The botmaster monitors transactions sent from the shared account and when it detects one, it stores the information about the new bot and sends it some funds. After the registration, bots will only receive funds directly from the botmaster.

If more bots try to register at the same time, there might be a conflict between their transactions (i.e. a double spend). In order to minimize this risk, the botmaster puts on the account several transactions, called *quotas*, containing just the right amount of coins needed to send the registration message. Furthermore, to reduce concurrency, it always sends multiple quotas at the same time. When a new bot wants to register, it picks a random quota and tries to send the message. It then sets a timeout for receiving the funding from the botmaster. If the timeout expires, the bot picks another quota and repeats the process. The same happens if its transaction gets rejected by peers or if another transaction spending the same

quota is detected. At any time, the botmaster makes sure there are enough quotas on the shared account, according to the rate at which new bots are joining.

Since the registration transaction comes from a shared account and only has an OP_RETURN output, neither the botmaster address nor the bot one are revealed.

It is worth noting that creating quotas would not be possible on Mainnet, as they would be considered as dust outputs and rejected by the network.

4.4.3 Commands and Responses

We distinguish between commands, that are messages sent by the botmaster, and responses, that bots send after executing a command. Bots can execute three types of commands: *hardcoded*, *shell* and *script*.

Hardcoded commands are functions that are already implemented by the bot code. They can be executed once or repeated over a period of time. Examples of hardcoded commands include a DoS function to attack a target or a keylogger to steal credentials. The botmaster can send parameters such as interval and number of iterations, or make the function run indefinitely until it sends a stop command.

Shell commands are command-line instructions that the bot directly execute on the infected machine. When the bot receives such command, it runs it and converts the output into a hexadecimal string to be sent as a response.

Script commands work similarly, but they use code stored on the blockchain. In particular, the code to execute is embedded by the botmaster in a previous transaction, called *script transaction*, and encrypted with a symmetric key, which is unknown to the bots. The command includes the transaction ID of the script transaction and the key to decrypt. When bots receive these commands, they retrieve the data, decrypt the payload and execute the code. They then convert the output into a hexadecimal string and send it the botmaster. In order to ensure all bots can send their response, the botmaster checks current funds of each bot before sending the command. If any bot does not have sufficient funds, the botmaster sends them more coins.

This approach takes advantage of the larger storage capacity of transactions on Testnet, which allow storing kilobytes of code on the blockchain. Additionally, this technique enables the botmaster to reuse the same code several times, saving coins and reducing its traffic. By using shell and script commands, bots are not limited to the functions their code implements, but are able to perform a variety of attacks, making it harder to estimate their real capacity.

4.5 Analysis

In this section, we analyze the sustainability of our protocol in terms of the amount of coins needed to run a botnet, as well as the robustness of its architecture and the security of its design.

4.5.1 Costs

FUNDING THE BOTNET At the time of writing, we were able to find six active faucets on the web. The amount of coins obtained per request varies from 0.0001 to 0.089 tBTC, with an average of 0.05 tBTC per request. By making a single request per faucet, we obtained approximately 0.12 tBTC. Requests are usually limited by faucets to one per day, for each given IP address. However, it is not hard to bypass the limit by using VPNs or proxy services. Furthermore, as previously discussed, a botmaster could run a miner to obtain a much greater amount of coins, without any restriction.

As such, we consider the estimate of 0.1 tBTC per day as a conservative lower bound of the funds that a botmaster can obtain to operate its botnet. In a real-life context, it is likely feasible to obtain ten to hundred times more than such an amount.

PROTOCOL MESSAGES COST As discussed in Section 4.4, all messages sent by the botnet spend the minimum relay fee (MRF), which is directly proportional to the size of the message and calculated as 1 satoshi per virtual byte.

In our protocol, transactions can have a fixed size, like quotas, registrations, and fundings, or variable size, like commands, responses, and scripts. Table 4.1 shows the MRFs for all transactions used in our protocol. For a quota batch transaction, which has 11 outputs, a MRF of 454 sats is needed. Registration transactions have a payload of 256-byte long, corresponding to a MRF of 373 sats. Fundings, which have 2 outputs, can be sent with 166 sats. Commands payload size is the smallest multiple of the AES block size (16 bytes), plus the IV (16 bytes). To simplify things, we assume hardcoded commands are short enough to fit into 2 blocks (32 bytes), which adds up to 48 bytes, with the addition of the IV. To send such a transaction, a fee of 161 sats is needed. We also assume shell commands are smaller than 100 bytes, with bigger instruction sent as scripts. Since the minimum size is 17 bytes (1-byte command plus the IV), the MRF varies from 133 to 230 sats. Script commands have a 3-byte command plus a 32-byte transaction ID, a 32-byte script encryption key and the IV. This sums up to 83 bytes, requiring a MRF of 197 sat. We assume the maximum size of script transactions and responses is 50kB. For what concerns our non-hardcoded commands, we have the

Message	OP_RETURN (Bytes)	Fee (Satoshis)
Quotas Batch	N/A	454
Registration (quota)	256	373
Funding	N/A	166
Hardcoded Command	48	161
Shell Command	17 - 116	133 - 230
Script Command	83	197
Script (Transaction)	117 - 51200	231 - 51349
Response	17 - 51200	133 - 51349

Table 4.1: Minimum relay fees for our protocol transactions.

following values. The encrypted screenshot script, along with the IV, is 128-byte long, corresponding to a MRF of 242 sats. To send the response (50kB), 51349 sats were needed. To send the output of `lswh` (12kB), 12860 sats were needed.

RUNNING THE BOTNET To have 1000 bots registered, 100 quota batches are needed, corresponding to 373000 sats. Considering the fees for the batch transactions (45400 sats), this sums up to 418400 sats (0.004184 tBTC), which is then the amount required to register 1000 bots. To fund the same number of bots, assuming an initial funding of 0.0001 tBTC each, and considering fees for the funding transactions (166000 sats), we have a total of 0.10166 tBTC. This means that 0.1 tBTC (our estimated lower bound) are enough to register and fund 1000 bots per day.

For what concerns daily operations, assuming a specific behavior is hard, as C&C communications for real botnets can be very diverse. As such we will focus on the number of bytes that can be sent per day by a 1000-bot botnet, assuming it is funded with 0.1 tBTC per day. To simplify things, we assume 1 sat is needed to send 1 byte of data. This way, 0.1 tBTC is enough to send around 10MB per day, which translates to 10kB per bot in our example, which is likely to be insufficient for a modern botnet, according to available statistics [145].

However, by analyzing the Testnet blockchain, it is easy to see that a solo miner could obtain an average budget of as much as 4 tBTC per day, which would allow the botmaster to run, for instance, a spamming botnet, or use this channel as a component of a larger hybrid botnet.

4.5.2 Architecture

TESTNET Despite being a testing network, Testnet is a very solid blockchain, as it constitutes a fundamental component of the Bitcoin ecosystem. In fact, it allows

developers to test changes to the protocol and new applications without wasting money or messing the real chain. Specifically, being released in 2012, the current version of the network (Testnet3) is one of the longest-running blockchains in the wild. Although a new version might be introduced, this would affect a lot of ongoing projects and protocol improvements development, making it unlikely to happen soon. As such, Testnet is a very stable backbone for a botnet C&C system.

A possible drawback of leveraging Testnet for a botnet might be its reduced network size, as fewer nodes might ease detection. However, the botmaster could mitigate this by deploying more nodes.

FAUCETS Faucets are a vital service for Testnet, as they allow developers to easily obtain the coins they need tu run their tests. In their absence, developers would need to run a miner, making their job both harder and more expensive. As such, it is unlikely that such services will cease to work.

BANDWIDTH Despite the use of non-standard transactions in our protocol allows transmitting bigger amounts of data, message size is still limited compared to the traditional client-server model. However, this system gains in terms of robustness, as communications are very hard to disrupt.

Given the above, it is possible that a real botnet would adopt a hybrid approach, with commands and responses happening on the blockchain, and larger data transmission being sent to a server, whose address changes periodically and gets updated via transactions.

4.5.3 Security

STEALTHINESS As communications happen via transactions, botnet messages will be permanently stored on the blockchain, creating an accessible evidence of past botnet activities and facilitating their analysis. Furthermore, the use of non-standard transactions makes it easier to recognize botnet messages. To mitigate this risk, the botmaster can limit their usage to only a part of the communications, trying to make other messages more similar to regular transactions.

ENCRYPTION All communications in our protocol are encrypted. However, if a bot is compromised, the adversary can learn both the botmaster public key and the botnet key, enabling the monitoring of all the messages coming from the botmaster. While this can help fighting the botnet activities, it does not prevent other bots from receiving and executing commands, thus being irrelevant to their operation.

To prevent this risk, the botmaster could encrypt and send messages individually for each bot. This would make the protocol more expensive and less scalable but it might still be feasible if the botmaster were able to obtain coins at a fast rate.

SHARED ACCOUNT In case a bot is compromised an adversary can also learn the private key of the shared account and try to drain all the funds, preventing new bots from registering.

A possible solution for the botmaster would be to employ a backup registration system, such as an external channel where new bots can post their encrypted registration message. To avoid disruption, the botmaster can regularly change it and communicate the updated info via transaction[1].

Another way the adversary can steal funds is to register fake bots to get the corresponding coins sent by the botmaster. This would increase the cost of the botnet and possibly make it infeasible to sustain. The botmaster, however, can monitor and test bots to detect and ban misbehaving ones. As an additional precaution, the botmaster could initially send a smaller amount of coins, and only send more if the bot behaves as expected.

Another issue, related to the shared account, is that it allows to compute the size of the botnet in terms of spent quotas. To mitigate this risk, the botmaster could periodically spend quotas at a random rate. Although this would make the system slightly more expensive, it would effectively conceal the real number of bot registrations.

COUNTERMEASURES As previously mentioned, the non-standard nature of the transactions used in our protocol allows to detect many of the botnet messages. Additionally, if a bot is compromised, it is possible to monitor and decrypt all messages from the botmaster. Furthermore, new bots can be prevented (or at least hindered) from registering.

Nonetheless, blocking botnet communications is hard as they are embedded into valid transactions. If a botnet is detected, messages coming from the botmaster could be prevented from spreading. However, this would be in sheer contrast with the anti-censorship principle at the base of the Bitcoin blockchain.

The most effective way to limit botnet communications would be to disallow non-standard transactions. However, it is unclear how this would affect the regular operations of Bitcoin developers.

[1]Note that bots are able to receive messages from the botmaster regardless of their registration status.

4.6 Experimental Results

We created a PoC botnet that implements our protocol, and then, we simulated its basic activities. In particular, we verified the ability to send, receive, execute and reply to commands. We then calculated the necessary amounts of coins needed for each type of transaction we use. Our results show that the proposed protocol is both viable and sustainable.

4.6.1 Non-standard transactions and fees

As a preliminary step, we verified the ability to send non-standard transactions on the network. We also tested the limits we could reach while still having transactions relayed.

As stated in Section 4.2.2, non-standard transactions allow us to do the following:

- send OP_RETURN outputs that are larger than 80 bytes,
- send repeated outputs, both OP_RETURN and addresses,
- send dust outputs,
- send transactions larger than 100 kB.

We used Bitcoin Core v0.18.0 to perform our tests, patching its code to allow creating transactions with repeated outputs (OP_RETURN or address). All other tests were possible without any modification.

For what concerns OP_RETURN size, we successfully sent transactions carrying as much as 50 kB of data. All transactions got immediately relayed and, after some time, mined. Although theoretically possible to send more, we were not able to send transactions carrying more data due to a limitation on the size of the argument that can be passed through the Linux command line[2]. As such we were not able to verify the ability to send transactions bigger than 100kB. However, we are confident this is actually possible, as this limit is not enforced for non-standard transactions. Transactions with repeated outputs, both addresses and OP_RETURN , were also accepted and relayed by all peers. As for dust outputs, we successfully sent transactions with as little as 0 satoshis, having them relayed and mined.

4.6.2 Proof of Concept

We implemented the C&C server with our patched version of Bitcoin Core, while bots run an SPV node using `bitcoinj`, which did not need any modification to

[2]This is a known limitation of the Linux kernel; the actual argument size limit depends on the stack size of the system [146].

use our protocol. Both bots and the C&C server run on a Linux operating system.

ENCRYPTION For asymmetric encryption and digital signature, we use RSA with a 2048-bit key and OAEP padding, which generates outputs of 256 bytes. This allows bots to send up to 214 bytes of encrypted data to the botmaster.

For symmetric encryption we use AES with 256-bit keys, using CRC block mode and PKCS5 padding. This encryption mode requires a random 128-bit IV (Initialization Vector), which is also needed for decryption. As the IV does not need to be secret, we send it in clear along with the cyphertext.

FEES The default MRF value on Bitcoin Core clients is set to a value of 1000 satoshis (sats) per kB. However, with the introduction of the so-called Segregated Witness (BIP141), transaction fees became dependent on what is known as virtual size, which is a function of the actual transaction size [3]. More specifically, the current MRF is calculated as 1 sat/vB, where vB stands for *virtual Byte*.

In our implementation, we make use of the embedded functions of the clients to calculate this value for each transaction.

TRANSACTIONS As stated in Section 4.4, we have the following types of transaction: quotas, registration, fundings, commands and responses. All transactions in our protocol have only one input.

Quotas transactions have 11 outputs, corresponding to batches of 10 quotas plus the change address. Each quota corresponds to the MRF of a registration message.

Registration messages have a quota as the input and 1 OP_RETURN output containing the payload. The payload contains a 36-byte-long Testnet address and a 32-byte-long AES key, encrypted with the public RSA key of the botmaster, which generates an output of 256 bytes.

Fundings contain two outputs: the bot address, receiving the funds, and the change address of the botmaster.

Commands and responses have 1 OP_RETURN output, plus the change address of the sender. Hardcoded commands have 3 bytes for the command plus the arguments (e.g. a target). The payload is encrypted with AES, so their output size corresponds to the size of the payload, padded to fit the block size (16 bytes), plus the IV (16 bytes). So, for example, an instruction like `dos www.domain.com`, which is 19-byte long, will have a data output of 32 bytes. Adding the IV we have 48 bytes. The script command has the following format: `scr TXID KEY`,

[3] The virtual size v is computed as $v = (w + 3 \cdot s)/4$, where w is the size of the transaction and s is the size of the corresponding base transaction (without the witness). In case of non-SegWit transaction, the virtual size is the actual size.

where TXID is a 32-byte-long transaction ID and the key is a 32-byte AES key. The corresponding IV is stored alongside the script itself.

COMMANDS We implemented the following commands: *dos* and *stop* as hard-coded commands, *lshw* as shell command, and one script command called *screenshot*. After executing shell and script commands, bots convert the output to a hex string and send it as a response message. To convert outputs into hex they use the following command: `$(CMD) | tr -d '\n' | xxd -r -p`, where `CMD` stands for the command they are executing. The `dos` command makes the bot attack a specific target, which is sent as a parameter. The DoS attack is performed using `hping3` and can be interrupted by a `stop` command. This command has no output. The `lshw` shell instruction makes the bot gather information about the hardware of the infected machine. On our bot machine, this command generates approximately 12 kB of data. The *screenshot* script is shown in Listing 4.1.

```
import -window root screenshot.png
convert -quality 5 screenshot.png screenshot.jpg
cat screenshot.jpg
```

Listing 4.1: The *screenshot* script

This script takes a screenshot in PNG format, which is around 500 kB, then compresses it to JPEG format, reducing its quality to fit into 50 kB of data. The `cat` command dumps the content of the file to produce the output to send as a response.

Chapter 5

NETWORK TOPOLOGY: OPPORTUNITIES BEYOND THE THREAT

In Section 3.3.2.7, we showed the arms race between the Bitcoin community, trying to conceal the topology of the network for security purposes, and researchers revealing methods to infer this information from side channels. In this chapter, we try to answer two questions:

1. What is the actual risk of revealing the network topology?

2. Can we make this information available in a reliable and trustworthy way?

We address the first question by empirically studying all the known attacks mentioned in research as aided by topology inference. As for the second question, we propose a viable protocol for inferring and monitoring the topology of the reachable network.

5.1 Active Topology Monitoring for the Bitcoin Peer-to-Peer Network

In Section 3.3.2, we described a variety of attacks on the Bitcoin P2P network, such as partitioning [95], eclipse [103], and deanonymization [110, 105]. These attacks are often mentioned as the main reasons that led developers to actively conceal information on the network topology. More specifically, while nodes in the network are publicly known [56], connections among them are are kept hidden.

This security-by-obscurity-like approach is meant to hinder adversaries from performing the above-mentioned attacks. However, as showed in Section 3.3.2.7, several techniques have been devised over the years that allow inferring the topology from side channels [21, 114], or by exploiting peculiarities of the protocol [66]. Although Bitcoin developers promptly react by fixing the protocol to make such techniques ineffective, it is hard to prevent attackers from using undisclosed methods or devising new ones. On the other hand, a side effect of this approach is that it impedes measurements and analyses of the network [21, 66], as well as an accurate definition of network models [95, 147]. In turn, these limitations hinder the improvement of the efficiency and security of the network. In contrast, having a reliable source of topology information could enable the design of a safer and more performing network.

In this chapter, we empirically show the limited effectiveness of topology concealment as a protection from known network-level attacks. We argue that the benefits of an open topology potentially outweigh its risks, and propose a protocol to reliably infer and monitor links among reachable nodes of the network.

We analyze the potential impact of our protocol on the Bitcoin network and experimentally evaluate its performance in a simulated environment, as well as its resilience to malicious nodes. Results show that our system has little impact on the network and can reach over 90% accuracy while being resilient up to a 30% of malicious nodes. Furthermore, although designed for Bitcoin, our solution can be implemented on any P2P network.

The contributions of this chapter include:

- we study the potential benefits of an open network topology for the Bitcoin P2P network;
- we show that most attacks mentioned in literature as related to topology information are actually independent from such knowledge;
- we design a simple protocol to prove the connection between two peers, and based on that, we propose a system to compute and monitor the complete network topology;
- we implement a proof of concept and evaluate the accuracy of our solution through simulations.

5.2 On the Open Topology

In this section we investigate on the threats and benefits of an open Bitcoin topology. In particular, we review state-of-the-art research to analyze the actual risks related to the knowledge of the topology and to highlight the advantages of making this information public.

5.2.1 Threats of Open Topology

As described in Section 3.3.2.7, the main reason to keep the topology hidden is to avoid the risk of network-level attacks and deanonymization [21, 148, 66]. We here analyze known threats and evaluate their relation to topology knowledge.

5.2.1.1 Network-level attacks

Network attacks commonly related to topology information include double spending, selfish mining, partitioning, and eclipse attacks.

DOUBLE SPENDING ATTACKS Double spending in fast payment transactions (see Section 3.3.2.2) was one of the first attacks correlated with topology. In particular, the work by Karame et al. [93] is often cited as a reference. However, the attack described does not make use of any topological information beyond the IP address of the victim, which is publicly available. On the other side, the probability of success decreases exponentially in the number of connections of the victim. As such, by monitoring the network topology, we could make nodes cooperate to ensure all peers have a sufficient number of connections.

Knowledge of the topology might indeed ease the attack when the victim is only connected to few, reachable, peers [112]. However, this situation is typical of unreachable nodes, which are out of the scope of our protocol. Furthermore, the time frame in which this attack can succeed is so short that the victim only needs to wait a couple of seconds to be safe. Additionally, the improvements made in the network propagation delays [149] likely made this frame even shorter.

MINING ATTACKS Mining attacks (see Section 3.3.1.1) could also benefit from the knowledge of the topology. In particular, miners could take advantage of network information to improve the propagation of their own blocks or to slow down competing ones [21, 83]. However, this is true if only a fraction of miners have such knowledge. Instead, if all miners had access to topology information, they could all leverage highly connected nodes to speed up block propagation, with no advantage of one over the other. Additionally, nowadays, miners often connect to each other through high-speed relay networks (see Section 3.2.1), which are separate from the main network.

PARTITIONING ATTACKS Partitioning attacks (see Section 3.3.2.3) are also a commonly cited as concern. In fact, knowing the topology, an attacker could easily identify target cuts in the network. However, the reference study on the topic [95] shows how the network can resist attacks lasting several hours even

against a powerful adversary controlling a botnet as large as the Bitcoin network itself.

Either way, since our protocol only focuses on reachable nodes, the attacker could not calculate a full cut of the network. In fact, all unreachable nodes would be missing, which, given their large number, would likely keep any two partitions connected. For instance, let us consider a public topology containing a cut C between portions $P1$ and $P2$ of the network. By looking at it the attacker might think that taking down C, $P1$ and $P2$ would be disconnected (i.e., partitioned); however, the attacker does not see any of the unreachable nodes that might be connected to both $P1$ and $P2$.

Furthermore, active monitoring the network graph would also have a twofold benefit. On the one hand, it would allow to detect these attacks in real time, and promptly react. On the other hand, using an adaptive, topology-aware protocol, it would be possible to maximize the number of nodes in the minimum vertex cut to increase the resilience of the network.

ECLIPSE ATTACKS Another attack typically mentioned as related to network topology is the eclipse attack (see Section 3.3.2.5). However, despite being cited by virtually all state-of-the-art papers as one of the attacks that justifies hiding the topology, the eclipse attack does not make use of any topology information. Instead, the only information needed to perform the attack is the IP address of the victim. In fact, this attack is mostly related to address management and the mechanism used to establish new connections. Furthermore, a number of defensive mechanisms have been introduced in the Bitcoin reference client to avoid such attacks[1]. Additionally, miners further protect themselves by running highly-connected gateway nodes [21] and propagating blocks via relay networks.

5.2.1.2 Deanonymization

The second major concern commonly linked to the knowledge of the network topology is transaction anonymity. However, as showed in Section 3.3.2.6, although it might improve the heuristics used to link transactions to their source node, topology information is hardly a requirement to deanonymize a transaction. The only exception to this is [110], where unreachable nodes are deanonymized by identifying the set of their peers. Note however that such a technique does not work in the current protocol. Either way, our protocol does not reveal the connections of unreachable nodes, and is thus irrelevant in this case.

A part from the above-mentioned workd, none of the state-of-the-art techniques use topology information, proving how these attacks are loosely related to

[1]For instance, see https://github.com/bitcoin/bitcoin/blob/v0.10.1/doc/release-notes.md .

it. Instead, as pointed out by Fanti et al. [150], the problem with anonymity lies in the transaction propagation algorithm, and should be addressed by adopting alternative approaches, like [25, 109] (we also contribute in this respect by proposing alternative propagation protocols in Chapter 6 and Chapter 7).

5.2.2 Benefits of Open Topology

The relevance of the topology for blockchain networks has been clearly shown by state-of-the-art research.

Kiayais et al. [151] show how the efficiency of information propagation is directly influenced by the network topology. Propagation delay can also be greatly affected by the number of connections that nodes have [152]. Furthermore, all unstructured P2P networks, like blockchain ones, are known to suffer from the so-called *topology mismatch* problem (i.e., the incoherence between logical and physical links), which causes inefficiency in the transmission of data [153, 154]. As for security, it has been shown how the Bitcoin network topology is far from being a random graph as designed (see Section 3.1), presenting nodes communities [66] and high levels of centralization [21]. Knowledge of the network topology can also improve the propagation of transactions and blocks, which in turn reduces the ability of performing double spending and selfish mining [155] (see Section 3.3). All these aspects could be addressed in real time if nodes had access to topology information.

On the other hand, as stated by Delgado et al. [66], hiding the topology prevents network analysis and measurement, further complicating the solution of existing issues. Similarly, Miller et al. [21] point out that understanding topology allows identifying structural faults in the network that might hinder broadcast optimization, and support the idea that monitoring the network can help quickly detect and react to attacks and mistakes. Authors of [156] also state it is crucial to acquire the knowledge of topology to accurately manage the network, optimize its performances, and ensure that it works well.

Monitoring the network topology could allow the introduction of mechanisms to avoid centralization and increase connectivity among nodes, as well as to detect weak spots that can be exploited to perform network-level attacks. With this work, we aim at taking a first step towards this direction. We do this by showing that an open topology should not be considered a security concern, and by proposing an effective protocol to reliably monitor the state of the network. In particular, we focus on reachable nodes, as they represent the core of the whole network.

5.3 Related Work

To the best of our knowledge, there are no known algorithms for discovering the topology of a P2P network. While this might seem surprising, a possible explanation resides in the fact that topology information is not as relevant in other P2P networks as it is for cryptocurrencies, where network attacks might lead to monetary profit. In fact, it has been shown that cryptocurrencies face different security issues compared to other P2P networks [10].

The closest-related works are on location-aware topology studies [157, 158] and routing protocols [159, 160]. The first group aims at improving propagation efficiency by leveraging geographical information. In particular, they have nodes choose their peers based on geographical proximity. The second group deals with search and retrieval of information within the network (as unlike blockchains, information is not duplicated by all nodes). However, none ot these works provide means for obtaining information on the global topology. Most notably, the only known topology-inferring techniques are those aimed at the Bitcoin network, which we described in Section 3.3.2.7.

As common denominator, all such techniques focus on the reachable part, require the adversary to connect to the whole network, and infer link by observing data propagation either in a passive way or by actively introducing marker messages. In the next section, we will leverage these concepts to design our topology-inferring protocol.

5.4 The AToM Protocol

We propose a dynamic topology monitoring algorithm for the Bitcoin P2P network called AToM (Active Topology Monitor). In this section, we give an overview of our topology-computation protocol and explain its operating principles. We then go through the design choices that led to our dynamic and secure monitor.

5.4.1 Overview

Our view of the Bitcoin network is depicted in Figure 5.1: *monitor* nodes connect to all reachable nodes in the network, and run the protocol to compute and maintain a continuously up-to-date state of the topology. We assume monitors know all public nodes by running a crawler, or, alternatively, by having nodes register themselves when joining the network.

SCOPE Monitors only watch over the reachable part of the network. Several reasons motivate this choice. First of all, as shown in Section 3.1, reachable nodes

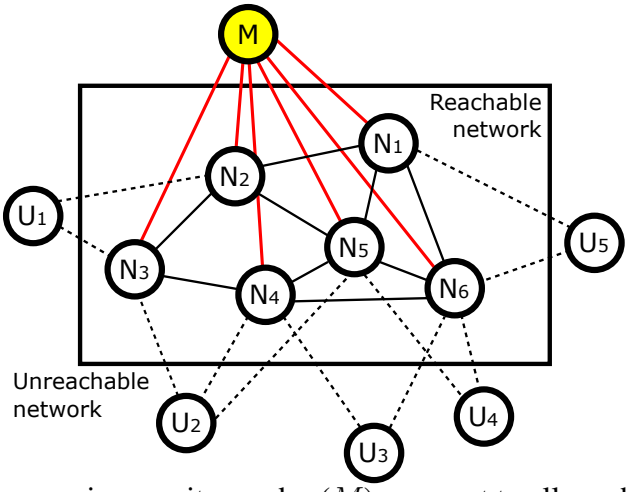

Figure 5.1: Our scenario: monitor nodes (M), connect to all reachable nodes (N), excluding unreachable ones (U).

constitutes the backbone of the network, since they maintain the greatest majority of connections and are involved in most data propagation. This makes the reachable part of the network arguably the most important to protect and optimize.

From a practical point of view, this part is also more convenient to monitor, as it is much smaller than the unreachable one [22], and its nodes show more stability in terms of number and connections [56, 161]. On the other hand, it is also harder to include unreachable nodes in the protocol. In fact, since such nodes do not have a public address, monitors cannot connect to them and, additionally, are not be able to uniquely identify them, thus making it hard to ensure they do not misbehave. Additionally, as unreachable nodes only connect to reachable nodes, they could compute their own topology by retrieving the public part from a monitor and adding their connections. This way, they would be able to autonomously decide the best peers to connect to.

Given the above, this restriction should be considered a feature of the protocol, rather than a limitation. In the rest of the paper, when talking about nodes, we will always refer to reachable ones.

APPROACH Similarly to some topology-inferring techniques described in Section 5.3, monitors in our solution connect to all nodes and leverage *markers* to prove connections. In order to minimize the overhead, we focus on outbound connections. It is easy to see that verifying outbound connections for every node means verifying all connections in the network: inbound connections are just the symmetric view of outbound connections. Furthermore, this approach allows us to limit our view to reachable nodes without being affected by the presence of unreachable ones in the network. In fact, by probing reachable nodes only and

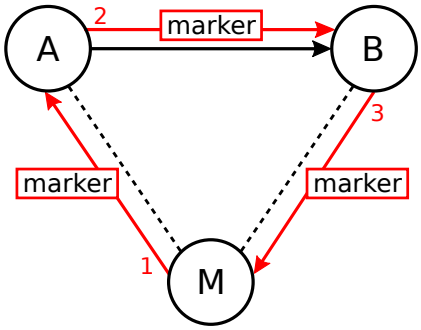

Figure 5.2: PoC overview: a monitor M verifies $A \rightarrow B$ by sending a $marker$ message to A and receiving it from B. Red arrows show the route of the marker.

monitoring their outbound connections, it is easy to see that no unreachable node can possibly be involved in the inferred topology.

To verify a connection, monitors have a unique marker message go through it, so as to prove the two nodes actually communicate. By making the marker random and unpredictable, monitors ensure the only way a node can know it, is to have received it from the peer to which it was originally sent. Differently from inferring techniques, which typically make use of side channels, we have node actively participate in the protocol. This makes the result more reliable, but presents a downside: if nodes misbehave, because faulty or malicious, it is hard to prove or disprove a connection without making use of trusted solutions like certificates or trusted execution environments. We address this by assuming nodes have a list of semi-trusted monitors, which are well-known (and thus partially trusted) nodes in the network that can potentially misbehave. This means not any node of the network can act as a monitor. Instead, this should be agreed on by the Bitcoin community. At a practical level, this can be obtained by leveraging existing semi-trusted entities of the Bitcoin network, such as the hardcoded list of peers in the reference client [12], or the DNS servers used for node bootstrapping.

5.4.2 Design

At a high level, the monitor computes the topology of the network by running, for each node, a protocol called *Proof of Connection* (PoC).

The PoC protocol for a *target node* N and a monitor M, and works this way: (1) M sends N a *marker* message, containing a random value r; (2) N forwards the marker to its outbound peers $P_1, P_2, ...$; (3) each peer P_i forwards the marker back to M; (5) when M receives a marker from P_i, it checks r and, if correct, adds the connection $N \rightarrow P$ to the topology.

We call a single execution of this protocol a *PoC round*. Figure 5.2 depicts a PoC round verifying the connection of a connection $A \rightarrow B$.

HANDLING NETWORK CHANGES By running a PoC round for every node we obtain a snapshot of the full network topology. However, since changes in the network can occur at any time, this snapshot can contain errors, making it an approximation of the actual topology. The relative stability of the reachable network makes the number of errors in a single snapshot very limited. In fact, all known topology-inferring techniques implicitly make the same kind of approximation. Nonetheless, to monitor the network, we need to keep this snapshot up to date over time.

The easiest way to do it is to repeat the network scan at a certain frequency so as to always have an updated snapshot. However, there are a few limitations with this approach. Firstly, short-lived connections established within two consecutive scans would remain undetected. This can be exploited by an adversary to completely hide a node from the monitor. In the second place, setting a scan frequency for the whole network will hardly adapt to all nodes, as each one will experience changes at different rates and times. This can affect both efficiency and accuracy.

Therefore, instead of repeatedly scan the network, we adopt a continuous-monitoring approach, where each node is scanned individually at a certain frequency, which is dynamically adjusted according to the stability of its connections (i.e., the number of changes per unit of time). This way, highly dynamic nodes are scanned at a higher rate than stable ones. This allows us to improve accuracy, as the scanning process adapts to the variability of single nodes, and reduce the impact on the network, as protocol messages are not exchanged all at once, but spread over time.

HANDLING MISBEHAVING NODES *Misbehaving nodes* can deviate from protocol, either accidentally (if faulty) or intentionally (if malicious), and produce errors in the result. While faulty nodes do not behave consistently, adversary-controlled ones, can act to deceive a monitor. As such, we will focus on malicious nodes, although the proposed solutions work for all misbehaving nodes.

We consider an adversary that control an arbitrary number of nodes in the network. With respect to our protocol, this adversary can have three possible goals: to hide a node, to hide a connection, or to fake one. In order to hide a node, this should refuse all connections from the monitors. However, monitors can be easily bypass this limitation, by connecting from a different address. Consequently, the only way for a node to hide is to reject all inbound connections. Nonetheless, this behavior would make the node unreachable, thus falling out of the scope of our protocol.

As for the other two objectives, the adversary can use one or multiple nodes. Note that a pair of cooperating nodes can easily hide a connection among each other, or fake one by using external channels or other malicious nodes. This is virtually impossible to avoid, as we cannot prevent malicious nodes from cooperating. Nonetheless, the usefulness of these attacks for the adversary appears to be limited. We then focus on the cases where at least one node is honest.

When deviating from the protocol, malicious nodes can: (1) forward a marker to an inbound peer; (2) forward a marker to a node that is not a direct peer, through another malicious node; (3) resend a valid marker used in a previous PoC round (replay attack); (4) tamper with a marker; (5) drop a marker received from the monitor (for one or more peers); (6) drop a marker received from a peer.

Case (1) is easily solved by having honest nodes accept markers from inbound peers only. Case (2) would make an honest node send a marker to the monitor despite not being a peer of the target, resulting in the monitor inferring a wrong connection. We address this by including in the marker the identity of the target (i.e. its address) and by making honest nodes accept markers only when they come from the expected source. Replay attacks (case 3) can be used to fake a previously-existing connection. We exclude this possibility by having monitors accept only markers of the current PoC round, for which the random value r acts as identifier. In case (4), any modification to the marker would result in the monitor deeming the marker invalid. As such, this case is equivalent to dropping the message.

In cases (5) and (6), the attacker can successfully hide a connection from the monitor. As we cannot prevent a malicious node from dropping a message, we introduce a peer reputation system. Specifically, we make monitors send to each node the list of its verified peers at the end of each PoC round. Honest nodes use such lists to feed a reputation system for their peers. If the reputation for a peer goes beyond a certain threshold, it gets disconnected.

This mechanism has a twofold effect. On one side, it discourages malicious nodes with the risk of being disconnected. On the other, it allows us to fix the error in the snapshot computed by the affected monitor. In fact, while the snapshot is initially incorrect, for excluding an existing connection, it becomes consistent when the connection ceases to exist. We refer to this phenomenon as *eventual consistency*.

HANDLING MAN-IN-THE-MIDDLE ATTACKS As connections in Bitcoin are unencrypted, it is possible for an adversary to hijack them. This adversary can drop, modify, and forge messages. We eliminate the risk of forgery and modification by including in the marker the digital signature of the monitor. With this change, all the attacker can do is to cause a connection to be dropped (by dropping or tampering with markers), i.e. a DoS attack. Although this can be a threat, it

is worth noting that it is not desirable for the attacker to lose the connection he controls. At the same time, it is desirable for the victim to drop a connection that has been hijacked. As such, we believe the risk of such an attack is low.

5.4.2.1 Protocol Procedures

In this section, we detail the PoC protocol and use it to build our *Active Topology Monitoring* (AToM) algorithm. We describe our solution in terms of the procedures executed by nodes and monitors.

NOTATION We denote a generic node by $N{=}addr_N$, where $addr_N$ is the IP address accepting incoming connections. We indicate a peer as $P{=}(addr_P, out)$, where out is $true$ if P is an *outbound peer*, and $false$ if it is an *inbound peer*. We denote an outbound connection from N to P with $N{\rightarrow}P$.

We represent the network as a directed graph $G{=}(V, E)$, where $V{=}\{N_1, ..., N_l\}$ and $E{=}\{(N, P) : N{\rightarrow}P\}$. Finally, we use M to denote the monitor running the protocol, and $G_M{=}(V_M, E_M)$ to indicate its local topology snapshot.

PROTOCOL MESSAGES We introduce the following new messages:

- `marker = (target,monitor,value)`: sent by a monitor M to a node N, with $target = N$, $monitor = M$ and a random $value$;
- `verified = (`L_n`)`: sent by a monitor M to a node N, where L_N is the list of currently verified peers of N.

THE PoC PROTOCOL To run the PoC protocol, monitors execute the PoC procedure, shown in Algorithm 1. This procedure generates a random number `r` and sends N a `marker` message with the following payload:

```
marker := (target:N,monitor:M,value:r)
```

. A timeout `t` is set to wait for incoming markers from other nodes. This is needed to avoid indefinite waits and to mark the end of the PoC round. When `marker` is received back from P, it is checked against the one sent to N. If it matches, P is added to the list of verified peers of N, L_N. When the timeout `t` expires, the PoC round ends, and the procedure outputs L_N.

Upom receiving a `marker` message, nodes execute the $HandleMarker$ procedure, shown in Algorithm 2. This procedure first checks the source of the message ($pfrom$): if $pfrom$ is the monitor M, `marker` is forwarded to all outbound peers; if $pfrom$ is an inbound peer and corresponds to the target N, `marker` is sent back to M.

Algorithm 1 PoC

$PoC(N)$:
1: $\texttt{r} \leftarrow rand()$
2: $\texttt{marker} = (N, M, \texttt{r})$
3: $send(N,\ \texttt{marker})$
4: $\texttt{t} \leftarrow getTimeout(N)$
5: **while** $(now < \texttt{t})$ **do**:
6: $receive(P,\ \texttt{marker})$
7: **if** $(\texttt{marker} == (N, M, \texttt{r}))$:
8: $L_N := L_N \cup \{P\}$
9: **output** L_N

Algorithm 2 HandleMarker

Env: $L = \{P : P.out = true\}$

$HandleMarker(pfrom, \texttt{marker})$:
1: $(N, M, \texttt{r}) = \texttt{marker}$
2: **if** $(pfrom == M)$:
3: **for each** P in L **do**:
4: $send(P,\ \texttt{marker})$
5: **if** $(pfrom == N)$:
6: **if** $(pfrom.out == false)$:
7: $send(M,\ \texttt{marker})$

THE ATOM PROCEDURES To build and maintain a snapshot of the topology (G_M), monitors run the AToM procedure, shown in Algorithm 3. This executes a loop for each node N. At each iteration, the PoC procedure is run, and the output L_N is used to update the snapshot G_M ($updateTopology()$). The `verified` message is then sent to N with the list of the verified peers, that is L_N plus all known inbound peers verified by other PoC procedures ($getPeers()$). Note that peer lists do not change between PoC rounds, which means that a connection stays in G_M until a PoC execution fails to verify it.

Before waiting for the next PoC round, the scan frequency is adjusted by $adjustFrequency()$. This function maintains a counter which is increased for every round with changes and decreased for every round without changes. If the counter goes over or below a certain threshold, the scan frequency is increased or decreased, respectively. This mechanism allows our system to be more resilient to sudden and isolated pikes in the peer set. Once the frequency is adjusted, the loop waits until the next PoC round.

If N disconnects, it is removed from V_M, and all its connections are removed from E_M ($removeNode()$). Again, we assume new nodes are automatically connected to M and added to V_M. When this happens, the corresponding PoC round loop is executed.

THE PEER REPUTATION SYSTEM When nodes receive the `verified` message, they process it by running the $HandleVerified$ procedure, shown in Algorithm 4.

For every peer, a per-monitor reputation is maintained, called ***M-reputation*** ($rep[M]$). The M-reputation ranges from 0 to a value Max, which is set when the node starts. The total peer reputation rep_P is obtained by summing all M-reputation values. If rep_P goes below a threshold Thr, P is disconnected. At the beginning of a connection, the M-reputation is set to Max for all monitors, which means the initial reputation of a peer is $Max \cdot Num_M$, where Num_M is the

Algorithm 3 AToM procedure

Env: $G_M = (V_M, E_M)$

$AToM()$:
1: **for each** N in V_M:
2: **while** ($isOnline(N)$) **do**:
3: $L_N \leftarrow PoC(N)$
4: $updateTopology(E_M, L_N)$
5: $\texttt{verified} = getPeers(N, E_M)$
6: $send(N,\ \texttt{verified})$
7: $adjustFrequency(N, L_N)$
8: $waitNext(N.freq)$
9: **done**
10: $remove(N)$

Algorithm 4 HandleVerified

Env: L, $Max<Thr<=Max{\cdot}Num_M$

$HandleVerified(M, \texttt{verified})$:
1: $L_N = \texttt{verified}$
2: **for each** P in L **do**:
3: **if** P in L_N:
4: $P.rep[M] = Max$
5: **else**:
6: $P.rep[M]{-}{-}$
7: $rep_P \leftarrow sum(P.rep[])$
8: **if** ($rep_P < Thr$)
9: $disconnect(P)$
10: **done**

number of monitors. When $HandleVerified$ is executed, each peer is checked against the list L_N contained in `verified`. If a peer P is not in L_N, its M-reputation is decreased by 1. Conversely, if P is verified, its M-reputation is restored to Max.

The value of Thr must be between Max and $Max{\cdot}Num_M$. This means P has to be verified by at least one monitor to be considered honest. However, it is possible to adjust the system to a target level of security and resiliency. For instance, if $Thr{=}Max{\cdot}Num_M$, P is disconnected even if one monitor does not confirm it. However, by setting $Thr{=}Max{\cdot}Num_M/2$, we trust a peer if confirmed by half of the monitors. Similarly, we can set Max to allow a target number of failed rounds. For instance, with $Max = 2$ we allow peers to temporarily disconnect from all monitors during one round or, equivalently, to disconnect from half monitors during two rounds without being disconnected. This can be used to handle disconnections and delays.

Finally, we consider as a full, trusted snapshot of the topology, the union of the current snapshots of all monitors. Formally, we consider

$$G_{AToM} = \bigcup_{i=1}^{Num_M} G_M.$$

5.5 Analysis

In this section, we study the correctness and accuracy of our protocol, both in an honest setting and in the presence of misbehaving nodes. Finally, we analyze the overhead for participating nodes in terms of the number of messages exchanged.

5.5.1 Correctness

It is relatively straightforward to show the correctness of the PoC and AToM algorithms in a trusted environment. In particular, we can easily prove that if a connection $N \rightarrow P$ exists, than it is added to the topology snapshot E_M within a finite amount of time.

We consider a network $G{=}(V, E)$, and a monitor M. Without loss of generality, we assume $M{\in}V$ and $(M, N){\in}E$ for every N in V, but do not show them in G. As M is not included in G_M, this does not affect our analysis. We analyze PoC and AToM by means of Algorithm 1 and 3, respectively.

PoC We prove that if a connection $N \rightarrow P$ exists, then P will be in the peers list of N (L_N) at the end of the procedure.

We give a formal proof for a network $G{=}(V, E)$, and two nodes N, P in V.

Theorem 1. *If $(N, P){\in}E$, then $PoC(N)$ outputs L_N, such that $P{\in}L_N$.*

Proof. Let us assume $(N, P){\in}E$, M executes $PoC(N)$, and N and P execute $HandleMarker()$.

The following sequence of events happens:
(1) M runs $PoC(N)$ and creates `marker`$=(N, M, r)$; (2) M sends `marker` to N; (2) N receives `marker` from M; (3) as `pfrom`$= M$, N sends `marker` to every outbound peer, which by assumption, includes P; (4) P receives `marker` from N; (5) as `pfrom`$= N$, P sends `marker` to M; (6) M receives `marker` from P; (7) as `marker`$=(N, M, r)$, M adds P to L_N. □

AToM We prove that if a connection $N \rightarrow P$ exists in G, than it is added to the snapshot G_M by the procedure.

Theorem 2. *If $(N, P){\in}E$ then $AToM(V)$ will add (N, P) to E_M.*

Proof. The proof is trivial: since $PoC()$ is executed for all nodes in V, this includes $PoC(N)$. As $(N, P){\in}E$, by Theorem 1, $PoC(N)$ outputs L_N, such that $P{\in}L_N$. Consequently, $updateTopology(E_M, L_N)$ will add (N, P) to E_M.
□

5.5.2 Security

In this section, we show how the snapshot G_{AToM} computed by a monitor is guaranteed to be correct even in the presence of misbehaving nodes. We refer to the malicious behaviors listed in Section 5.4.2.

We consider a malicious node N and an honest peer P. For cases (1) to (3), we show that $(N, P){\in}E_M$ only if $(N, P){\in}E$. For cases (4) to (6), we show that a

mismatch between E and E_{AToM} can only occur for a limited time frame (eventual consistency).

Case (1): let us assume N sends a `marker` message to an inbound peer P; since `pfrom.out` is $true$, P does not send `marker` to M, which then does not add (N, P) to E_M.

Case (2): let us assume N sends, through other nodes, the `marker` to non-peer node P; as `pfrom` is different from N, P does not send `marker` to M, which then does not add (N, P) to E_M.

Case (3): let us assume N sends a `marker` received in a previous round; as `value` changes at every round, M does not consider it valid for this round and then does not add (N, P) to E_M.

Cases (4),(5) and (6): let us assume N modifies or drops a `marker` message for a connection (N, P) (the case in which P is malicious is symmetric); in this case, M does not add (N, P) to E_M. Consequently, N is not included in the `verified` message sent to P at the end of the PoC round, making P decreasing rep_N. Now, two cases are possible: either N lets at least one monitor verify (N, P), or it hides it from all monitors. In the first case, there is at least one M for which $(N, P) \in E_M$, which implies $(N, P) \in E_{AToM}$. In the latter case, the following happens: as rep_N is initially set to $Max \cdot Num_M$, and $Thr < Max \cdot Num_M$, after a finite number of PoC rounds, rep_N will be less than Thr, and then N will be disconnected from P. When this happens, $(N, P) \notin E$ and $(N, P) \notin E_{AToM}$. In both cases, the snapshot G_{AToM} becomes correct after a finite number of rounds, either by verifying (N, P) and adding it to E_{AToM}, or by having (N, P) removed from G.

5.5.3 Accuracy

In this section, we discuss the accuracy of AToM in a dynamic network.

As previously discussed, changes in the network can produce a mismatch between the local snapshot G_M and the real topology G. In particular, at any time, one of the following changes can occur: (1) a new node N joins the network ($V := V \cup \{N\}$), (2) a node N leaves the network ($V := V - \{N\}$), (3) a new connection $N \rightarrow P$ is formed ($E := E \cup \{(N, P)\}$), (4) a connection $N \rightarrow P$ is closed ($E := E - \{(N, P)\}$).

Note that case (1) implies case (3), since, by definition, a node with no connections is not part of the network. Similarly, case (2) implies case (4) for all the connections of the disconnected node. As such, without loss of generality, we can focus on cases (3) and (4).

For both cases, if the change involves an outbound connection of N and it occurs before executing $PoC(N)$, it is trivial to show that such change will be reflected in the result. Similarly, if a connection $N \rightarrow P$ is closed during the ex-

ecution of $PoC(N)$, M will not receive `marker`, and correctly exclude (N, P) from E_M. Given the above, two cases produce a mismatch: (1) when a connection $N \rightarrow P$ is dropped after $PoC(N)$ completes: in this case, we have a false positive ($(N, P) \in E_M$ but $(N, P) \notin E$); (2) when a connection $N \rightarrow P$ is established after $PoC(N)$ has started: in this case we have false negative ($(N, P) \in E$ but $(N, P) \notin E_M$). As both mismatches are solved at the next execution of $PoC(N)$, we can say that errors in E_M for each connection (N, P) in E are limited to the time frame between two consecutive executions of $PoC(N)$, which depends on the scan frequency for N. As such, it is possible to reduce errors for a node by reducing its scan frequency. In that respect, the adaptiveness of AToM (set via the variable $adjustFrequency$) is meant to balance the error rate with the scalability of the protocol by reducing the mentioned time frame for nodes showing faster changes in their topology.

Note that the variability of the Bitcoin network with regard to reachable nodes is relatively low, with an average of 3 changes per minute[2]. This allows a monitor to obtain highly accurate snapshots even at low monitoring frequencies.

5.5.4 Overhead

In this section, we analyze the impact of AToM in terms of number of messages. In particular, we calculate the average number of messages exchanged by a single node in a *complete PoC round*, by which we mean the execution of PoC for all its peers, both inbound and outbound.

During a complete round, nodes receive, from each monitor, a `marker`, which is sent to all outbound peers, plus a `verified` message. Additionally, they receive a `marker` from each inbound peer, which is sent to the monitor. As such, for a complete round, $Msg_M = (P_o + 2 \cdot P_i + 1)$ messages per monitor are exchanged, where P_o and P_i are the number of outbound and inbound peers, respectively.

As, on average, Bitcoin nodes experience around one change per hour in their outbound connections [161], monitors will need to run, for each node, approximately one PoC round per hour. Hence, each node will exchange, for each monitor, around Msg_M messages per hour. As mentioned in Section 2.3.1.1, connections for a node are usually limited 8 outbound and 117 inbound. Although, this value can actually be diverse in real life, with most nodes never reaching the limit [16, 66] and few others exceeding this number [21], we can set $P_o = 8$ and $P_i = 117$ as average values. As such, a node would exchange around $P_o + 2 \cdot P_i = 8 + 2 \cdot 117 + 1 = 243$ messages per hour, for each monitor.

If we run, for instance, 10 monitors, each node would exchange around 2430

[2]This value has been obtained from data available on Bitnodes [56]

extra messages per hour, that is, less than 1 message per second. Following the same reasoning, we could run as many as 50 monitors with only 3 extra messages per second for each node. Considering a Bitcoin node currently exchanges around 50 messages per second [162], we can say AToM has relatively little impact on nodes. Moreover, this can be further adjusted to reach the best compromise between accuracy and overhead.

5.6 Experimental Results

To evaluate our solution, we implemented a proof of concept and performed experiments in a simulated environment. We here give details and show the results of our experiments.

5.6.1 Proof of Concept

We implemented AToM using Bitcoin Core. Since inbound peers are assigned a random port, it is impossible to distinguish two different nodes connecting from the same IP. As such, we identify nodes by their IP address only, limiting our compatibility to 1 node per IP address. Considering that virtually all public nodes run from a different IP, we consider this a minor issue.

The average PoC round frequency (f) is adjusted as follows: after 3 consecutive PoC rounds with changes, f is increased by 1 second; after 3 consecutive PoC rounds without changes, f is decreased by 1 second. The PoC round scheduling is randomized following a Poisson distribution over f. This further spread messages over time and also makes it harder for an adversary to predict PoC round timings, thus reducing its ability to hide a connection. The PoC round timeout was set at the beginning of the next round, making a marker valid until a new one is created. As for the reputation system, Max was set to 10, and Thr to $Max \cdot Num_M/2$.

We also implemented malicious nodes that deviate from the protocol to hide existing connections and fake non-existing ones. In particular, all markers received from honest peers are dropped, thus concealing the connection from the monitor. On the other hand, when a marker is received from the monitor it is only forwarded to other malicious nodes, which, in turn, forward it to the monitor, thus making it infer a wrong connection. Note that such a behavior corresponds to the worst-case scenario for our protocol.

5.6.2 Evaluation

We simulated a small-scale Bitcoin network with 4 monitors and 50 nodes, randomly connected to each other. Each node establishes at least 3 outbound con-

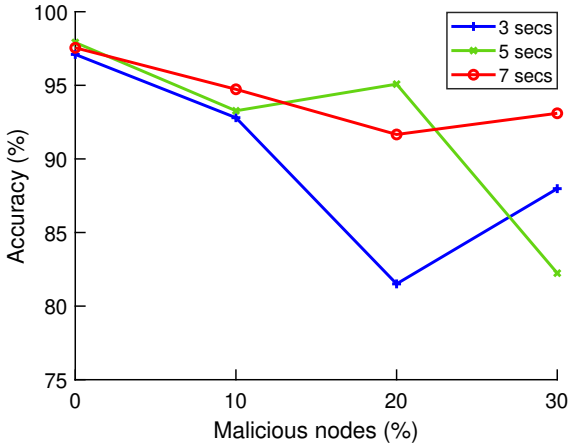

Figure 5.3: AToM accuracy for a network with variability of 3, 5, and 7 seconds.

nections to random peers of the network. Changes in the network (i.e., nodes connecting/disconnecting) happen with a predefined average frequency, referred to as *network variability*. To cope with the scale of the simulation, we use a higher variability than the real Bitcoin network (see Section 5.5.3). In particular, we performed three series of experiments, setting the variability at 3, 5, and 7 seconds. These values were chosen to be respectively lower, equal, and higher than the default AToM scan frequency, which was set to 5 seconds.

In each series, we performed 4 runs, varying the percentage of malicious nodes in the network, from 0% (i.e., with all honest nodes), to 30%. Each simulation has been run for 5 minutes, probing monitors every 10 seconds. Accuracy was calculated as the percentage of correct links in the snapshot against the ground-truth topology (both incorrect and missing connections were counted).

Results are shown in Figure 5.3. In a fully honest setting, accuracy is around 97%, independently from the network variability, showing that AToM well adapts to changes in the network. As expected, the presence of malicious nodes affects accuracy, especially with higher variability values. Nonetheless, when the variability is lower than the default scan frequency, we obtained over 90% accuracy, even with 30% of malicious nodes. This shows our monitor is highly resilient to misbehaving nodes as long as its scan frequency is close to the network variability. Furthermore, in any of our experiment accuracy went below 80%.

Note that results with high concentrations of malicious nodes (20% and 30%) are somewhat incoherent. We attribute this phenomenon to the fact that such nodes were randomly connected to each other, thus introducing a certain level of unpredictability.

Chapter 6

UNREACHABLE NODES: THE INVISIBLE BEDROCK OF BLOCKCHAIN NETWORKS

In Section 2.2.3.2, we saw that unreachable nodes are different from other nodes in that they only maintain outbound connections (see Section 2.3.1.1). In Section 3.1, we also saw that these nodes constitute the 90% of the whole Bitcoin P2P network. However, most research focuses exclusively on reachable nodes, which are considered more influential and thus more worthy of being studied.

In this chapter, we study the importance of unreachable nodes with special regard to the propagation of messages. In particular, we show how increasing the participation of these node can be beneficial to the whole network. We propose minor protocol changes targeting unreachable nodes, which can improve the efficiency of data propagation. Furthermore, we propose a novel transaction relay protocol that leverages unreachable nodes to improve anonymity.

6.1 Improving Bitcoin Transaction Propagation by Leveraging Unreachable Nodes

Many works in the literature studied the characteristics, efficiency and security of the Bitcoin network by focusing exclusively on its reachable portion [64, 21, 147, 163]. It is implicitly assumed that reachable nodes are more important for the connectivity of the network. Indeed, as noted in [18], these nodes are essentials in a permissionless blockchain like Bitcoin as they allow anyone to join the system. In particular, the number of reachable nodes is an indicator of the openness of the network. Moreover, given the centrality of these nodes in the topology, they are likely to be a core element of data propagation.

For all these reasons, the importance of the reachable part of the network is sufficient to not question the validity of the results obtained in the literature. Nevertheless, this portion only accounts for 10% of the whole network. As such, it is undeniable that unreachable nodes are somewhat overlooked, and that the sheer number of such nodes should be an indicator of their relevance in the network.

In this chapter, we analyze the role of unreachable nodes in the propagation of messages and show how their participation can be beneficial to the network. In particular, we study the characteristics of these nodes, as emerged from state-of-the-art research, and identify some of their strengths and weaknesses compared to reachable nodes. We then propose changes to the protocol to improve the connectivity of the network and the efficiency of message propagation.

Additionally, we show that unreachable nodes are inherently protected from a certain type of adversaries. Based on this characteristic, we design ReAP, a novel transaction propagation protocol that potentially improves security against deanonymization attacks. Our solution explicitly involves unreachable nodes in the propagation pattern and exploits their position in the network to conceal the source of the message. We thoroughly justify our design choices and study the security of our protocol against an eavesdropper adversary [150]. Our solutions are simple to implement and can effectively bring immediate benefits to the Bitcoin network.

6.2 Unreachable Nodes in the Bitcoin Network

In the Bitcoin literature, reachable and unreachable nodes are often named servers and clients, respectively. This naming recalls the ability of ones to accept connections and the fact that the others connect to them. However, as nodes of the same P2P network, there is no actual client-server relationship among them. Moreover, there is no distinction between the two types of nodes in the Bitcoin protocol. A more precise classification, commonly used in other P2P-related papers, distinguishes between *routable* and *non-routable* peers, and calls *unreachable* those peers that are routable but cannot be contacted (e.g., because they are offline or only accept connections from known peers) [65, 164]. In this thesis, we use the terms reachable and unreachable to indicate routable and non-routable. Furthermore, in this chapter, we will also denote reachable and unreachable nodes by *R nodes* and *U nodes*, respectively.

As previously mentioned, U nodes have been marginally covered by state-of-the-art research. In fact, most network-wide analyses [16, 64, 65, 21] leave U nodes out of scope. The reason for this is that in order to observe the network behavior at a global level, measuring tools connect to all nodes at the same time. However, this is not possible with U nodes, which only establish outgoing con-

nections. Therefore, the only way to study U nodes, at a global scale, is to deploy numerous R nodes and have U nodes connect to them. By adopting this approach, Wang et al. [22] were able to study U nodes in detail. As of today, their work is the only known network-wide analysis of U nodes.

Besides it, the only papers focusing on U nodes are those targeting them for deanonymization [110, 111]. The specificity of these attacks stems from the difficulty of targeting U nodes with standard approaches, where the adversary connects to all nodes and observe the propagation of transaction messages. In fact, like measuring tools, the adversary is unable to include U nodes in its scope. Furthermore, NATted nodes are also hard to distinguish, since they can share the same IP address. This is why deanonymization attacks targeting U nodes usually require fingerprinting techniques [110, 111].

For these reason, U nodes are hard to target in observation-based attacks [150] and unsolicited-message-based attacks [165]. Similarly, U nodes are excluded from many network-level attacks, such as eclipse attacks [103], topology-inferring attacks [21], and partitioning attacks [100].

On the other hand, U nodes are very susceptible to deanonymization attacks when they directly connect to the adversary [22]. At the same, studies showed that regular Bitcoin users tend to use U nodes [147, 22], meaning that the majority of transactions is actually generated by U nodes. As such, it is fundamental to improve their security and efficiency.

Additionally, we argue that addressing unreachable nodes also helps improving the robustness of the whole network. In fact, as noted in [18], U nodes increase connectivity and are harder to attack for adversaries without access to core infrastructure. An example of this was given in the previous chapter with reference to partitioning attacks (see Section 5.2.1.1): by being invisible to the adversary, U nodes make these attacks much harder to succeed.

In this chapter, we leverage the specificities of unreachable nodes to improve the efficiency and security properties of the whole Bitcoin network.

6.3 Background

There are different reasons for a node to be unreachable. In some cases, nodes purposely choose not to accept incoming connections. This can be the case of measuring tools or nodes run by mining pools [114]. However, the most common reason is that the device running the node is behind some sort of barrier that prevents outer entities from reaching the device. This barrier can be a firewall, a proxy server or, most commonly, a NAT device.

Unreachable nodes are typically associated with NATs. In fact, Carrier-Grade NATs (CG-NATs) are the primary cause for the unreachability of a node, as most

ISPs use this technology to grant access to a larger number of devices. In this section, we provide more details about NAT and describe the current Bitcoin transactions propagation protocol, which we will modify to provide better anonymity.

6.3.1 NAT and P2P networks

Network Address Translation (NAT) [58] is a method to map IP addresses between incompatible networks. The most common type, known as *Network Address and Port Translation* (NAPT), is often used to connect private networks to the Internet without the need to assign a unique address to each device. NAPT is often regarded as a solution to the IPv4 address exhaustion problem [166], since it allows a large number of devices to connect through a shared IP address. As a side effect, such devices cannot be reached from the Internet, unless they first open a connection.

While this is not a problem in a client-server setting, it is a serious limitation for P2P networks. Notably, it prevents NATted nodes from connecting to each other. To overcome this limitation, *NAT traversal* techniques have been devised [167].

The Bitcoin reference client implements *Universal Plug-and-Play* (UPnP), which, however, is incompatible with CG-NATs, as it needs direct access from the host. Furthermore, the UPnP option is disabled by default due to a known vulnerability in the protocol. As a consequence, NATted nodes only establish outbound connections, which in the reference client are limited to just 8.

6.3.2 Transaction Propagation and Anonymity

In Section 2.3.2 we described how transactions are propagated through the network. The propagation pattern of each transaction is determined by the relay step. In the *diffusion* protocol, used in Bitcoin, nodes relay transactions to each neighbor with an independent, exponential delay. The same protocol is used by the source node of a transaction to begin its propagation. In other words, new transactions are transmitted the same way as relayed ones.

As we showed in Section 3.3.2.6, the pattern generated by this protocol leads to possible deanonymization attacks. In particular, Fanti et al. [24] showed that, by connecting to all reachable nodes, an *eavesdropper adversary* can reach very high levels of accuracy. Furthermore, this is true even when the adversary adopts a naive strategy like the *first-spy* estimator, which simply links a transaction to the first node that relays it. The reason at the base of such a poor anonymity lies in the symmetry of propagation from each node to its peers. Similarly to [109], in this chapter, we propose an asymmetric propagation protocol to enhance transaction anonymity. In particular, we do this by leveraging the inherent protection of

unreachable nodes.

6.4 Related Work

Unreachable nodes have been extensively studied by Wang et al. [22]. In order to perform their analyses, they deployed around 100 nodes, through which they collected information on more than 100 K unreachable peers, which generated more than 2 M transactions. Their findings show that most connections last for less than 60 seconds, while, at the same time, most transaction propagations are sent over long-lived connections (more than 100 seconds), showing a high degree of centrality. Finally, they show a method to deanonymize transactions coming from unreachable peers, with the help of an external listener node. Their results show that unreachable nodes are also susceptible to the first-spy estimator attack. In Section 6.6, we propose a new protocol that makes this attack much less effective, by having unreachable nodes relay new transactions to a single peer and mix their messages from those received from other peers.

In relation to transaction anonymity, which was extensively discussed in Section 3.3.2.6, it is worth mentioning that our protocol is partially influenced by Dandelion[25, 109], especially in the ideas breaking the symmetry of propagation and transaction mixing. However, while their protocol has some relevant complexity and only applies to R nodes, our design is simple and applies to the whole network.

Another attack targeting U nodes is the one by Byriukov et al. [110]. Their attack target U nodes at a global level by making use of a fingerprinting technique based on the propagation of `addr` messages. In Section 6.5.1.3, we propose a change to the protocol that effectively makes such a technique ineffective.

6.5 Modifications to the Protocol

We propose some changes to the network protocol, which leverage the specificity of unreachable nodes to improve the efficiency and security of the network. In particular, we propose the following changes:

- Explicitly distinguish reachable and unreachable nodes;
- Increase connections from unreachable nodes;
- Disable advertisement of unreachable addresses;
- Adopt the ReAP propagation protocol described in Section 5.4.2.1.

6.5.1 Network Changes

We first describe the changes to basic network protocol behavior that allow us to improve security and efficiency.

6.5.1.1 Explicitly distinguish between R and U nodes

Although there is some difference in the behavior of R nodes and U nodes in the reference client, the Bitcoin network protocol does not make any explicit distinction between them. However, our solution is based on the different characteristics shown by the two types of node, as shown in Section 6.2. As such, explicitly distinguish between R nodes and U nodes is a necessary step.

Different strategies can be followed by a node to determine its reachability. A naive approach would be to verify if the client accepts incoming connections. However, it might be the case that a node is accepting connections but its address is unreachable from the outside. A better approach is to have the node connect to its own address, as seen by its peers, and set itself reachable, if the attempt succeeds, and unreachable, otherwise.

6.5.1.2 Increase U nodes connections

The second modification we propose is to increase the number of outbound connections of U nodes. This change has several effects. Firstly, it helps leveling the imbalance of connectivity between R and U nodes. In fact, while R nodes can reach 125 connections, U nodes only maintain up to 8, corresponding to their outbound peers. On the other hand, inbound slots are often underutilized by R nodes [16], which means they can handle a higher number of connections.

Secondly, increasing the number of peers means receiving, and relaying, more transactions per amount of time. Given the great number of U nodes, even a small increase in their connections might produce a significant improvement in the propagation speed of transactions and blocks.

Furthermore, from a security perspective, it has been shown how increasing the number of outbound connections can improve resistance against DoS attacks [18], eclipse attacks [103], and isolation attacks [96].

Finally, a higher number of connections for U nodes can be beneficial for the anonymity of our propagation protocol, as we will show in Section 5.4.2.1.

6.5.1.3 Do not advertise U nodes addresses

In the current protocol, U nodes, like R nodes, advertise their public address to their peers. These addresses represent 90% of those being spread through the network [22, 21]. However, being unreachable, these addresses are of no use to

any other node. At the same time, they increase network traffic [110] and likely produce a high number of failed connection attempts. Additionally, they potentially reduce the availability of reachable addresses, since new (often unreachable) addresses replace old ones in the node database when the address pool is full.

From a security perspective, these addresses enable fingerprinting techniques, which allow for deanonymization attacks [110, 59]. Disabling their advertisement to outbound peers would effectively hinder deanonymization of U nodes.

6.6 The ReAP Protocol

In the following, we discuss and describe our new *Reachability-based Anonymous Propagation* (ReAP) protocol design. The protocol explicitly leverages U nodes to improve resiliency against deanonymization. Similarly to Dandelion [25, 109], we include two essential concepts in our design: proxied broadcast and transaction mixing. Proxied broadcast consists in delegating the diffusion of a new transaction to another node (called *proxy*), allowing to hide the real origin of the transaction. Mixing consists in sending to the proxy other new (proxied) transactions, received from other peers. This makes it hard for the proxy to distinguish between transactions generated by the sender and others relayed by the sender, but generated by other nodes.

Given what said about U nodes in Section 6.2, we want to leverage their protected position in the network to conceal the origin of a transaction. The core idea is to make R nodes, which are more susceptible to deanonymization, use U nodes as proxies for their transactions. This way, such transactions will look as generated by their proxies instead of the actual source. Additionally, we hinder proxies from distinguishing such transactions by mixing them with transactions from other nodes.

Before detailing the propagation protocol, we define our adversary model and motivate our design choices.

6.6.1 Network and Adversary Model

To describe our protocol, we model the Bitcoin network as in Figure 6.1. We call O the R node running the protocol and generating new transactions. Other nodes are denoted by R_i, if reachable and U_i, if unreachable, for $i = 1, 2, ...$ The adversary A aims at deanonymizing transactions generated by O and can control various nodes, both reachable and unreachable, which we denote by R_i^A and U_i^A, respectively. A can connect to all reachable nodes and also create multiple connections to the same node (including O). Nonetheless, A cannot directly connect to other U nodes. To that respect, A can only deploy multiple R nodes to increase

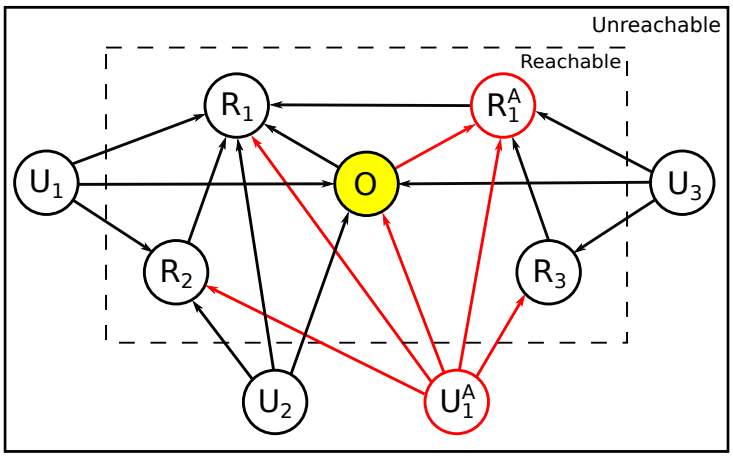

Figure 6.1: Our view of the Bitcoin network: the origin O of a transaction is connected to R and U nodes. The adversary (colored in red) deploy both R and U nodes and connect to all reachable nodes.

the chance of having honest U nodes connecting to it. Additionally, the adversary can create and transmit transactions, as well as relay or retain others received from its peers.

6.6.2 Design

In the ReAP protocol, R nodes leverage U nodes as proxies and use transactions coming from other U peers for mixing. Instead, U nodes use R nodes as proxies and mix new transactions with those coming from other R peers.

This scheme allows protecting both R and U nodes. In fact, U nodes cannot distinguish between transactions generated by their R peers and those proxied by such peers but generated by other U nodes. Similarly, U-generated transactions are indistinguishable to R nodes from those generated by other R nodes and proxied by their U peers.

However, a naive design could lead to easy deanonymization attacks, and also to an ineffective propagation of new transactions through the network. Therefore, we need to define (1) which peers are used for proxying and (2) which transactions are used for mixing.

As for point (1), an R node can select one, all, or a subset of its U peers. Note that an adversary can control a large subset of U peers of R nodes. This increases her probability of being selected as the first proxy for many R-generated transactions, allowing an effective use of the first-spy estimator. At the same time, if we send all transactions to a single proxy, it will be easy for this one to narrow down the set of transactions possibly generated by the sender. As such,

we first select a subset of peers to be used as proxy and pick a random one within this subset for every proxy operation. We call this subset the *proxy set*. In order to distribute transactions among all nodes and minimize the risk of a proxy collecting all new transactions from a node, we change the proxy set at a certain rate. We call *epoch* the time frame in which a proxy set is used.

As for point (2), we first need to identify which transactions are suitable for mixing. Note that transactions received by an R node from other R peers following our protocol have already been diffused, making them unsuitable for mixing (since the adversary might already know them). Similarly, transactions diffused by U peers might have already been received by the adversary. On the other hand, it is easy to see that proxied transactions are the least likely to be known to the adversary, and thus best suite for mixing.

Therefore, we need to identify which transactions are being proxied and which are being diffused. To do so, we mark proxied transactions and distinguish between two propagation phases: the *proxying phase* and the *diffusion phase*. We call transactions in the proxying phase *proxy transactions*. When a new transaction is created is marked as proxying and sent to a node of the proxy set. As for mixing, nodes use proxy transactions coming from their peers. We call the set of proxy transactions used for mixing, the *mixing set* of a node. Transactions in the mixing set are relayed through the same path as newly-generated ones so as to make them indistinguishable from each other.

Ideally, we would like the mixing set to be as large as possible. However, if we used all incoming proxy transactions, they would never be diffused. Instead, we include only a fraction of such transactions in the mixing set, and diffuse the rest. To do so, we need to decide which transactions to diffuse and which to relay. A possible strategy is to select some peers in each epoch, and only use transactions coming from them. However, if an adversary controls many of these peers and also the selected proxy, she could track most transactions in the mixing set of the target, leading to an easy deanonymization. To avoid such a risk, we select proxy transactions from all of our peers and probabilistically include them in our mixing set. In particular, for each proxy transaction, we keep proxying it with a certain probability p and diffuse it otherwise. This way, despite being able to track or inject proxy transactions for a specific node, an adversary cannot affect the number of honest transactions included in its mixing set. A correct choice of p will be fundamental for the effectiveness and efficiency of our protocol.

To further protect R nodes from adversaries controlling many inbound connections, we adopt the *bucketing* strategy used in Bitcoin Core for managing addresses. This mechanism is used to prevent an adversary from filling up the address database with malicious IPs, and it is based on the assumption that the attacker only controls nodes from a limited address space [95]. In particular, each bucket contains addresses from a different subnet. Similarly, we make R nodes se-

Algorithm 5 ReaP: Proxy procedure

procedure PROXY(Transaction tx)
    Pick a random peer P from the proxy set
    Send tx to P and set a timeout t
    When t expires:
    if The majority of outbound peers advertised tx **then** Return
    elseRepeat

Algorithm 6 ReaP: Propagation Rules for R nodes

Divide time into epochs
if New epoch begins **then**
    Select subset S from U peers uniformly at random from different buckets
    Set S as the proxy set
if Create new transaction tx **then**
    Mark tx as $proxying$
    Run PROXY(tx)
if Receive a $proxying$ transaction tx_m from a U peer **then**
    with probability p, run PROXY(tx_m)
    otherwise, run DIFFUSE(tx)

lect proxies and transactions for the mixing set uniformly at random among peers from different buckets.

Finally, to cope with the risk of a transaction not being diffused, due to a DoS attack by a proxy or to an excessively long proxying phase, each node sets a timeout t for every proxied transaction. When t expires, the node verifies if the transaction has been diffused by checking if the majority of outbound peers have advertised it back to us. We choose to monitor outbound peers to minimize the risk of an adversary deceiving an R node by relaying proxied transactions from other adversary-controlled U peers. The same rule is applied to both new and relayed transactions, so as to avoid deanonymization due to rebroadcast. In the current protocol, in fact, a rebroadcast is only done by the source of the transaction, and can thus reveal its origin [105]. In our protocol, instead, rebroadcast applies to all proxied transactions, thus leaking no new information.

PROTOCOL RULES To detail the propagation rules of the ReaP protocol, we first define the *proxy* operation on a transaction tx as in Algorithm 5. Next, we define the propagation rules for R nodes as in Algorithm 6. U nodes follow the same rules, except they use R peers instead of U peers and do not use buckets.

6.7 Analysis of the ReAP Protocol

6.7.0.1 Limitations

The ReAP protocol requires R nodes to have U peers connected to them. However, newly-joined R nodes usually have to wait some time to have other peers connect to them. We address this limitation by having new R nodes using the Diffusion protocol until they have a sufficient number of U peers. Additionally, to prevent an adversary from taking advantage from this situation (by filling up all inbound slots), we also adopt the bucketing strategy. Specifically, we make R nodes use our protocol only when enough U peers from different buckets are connected.

6.7.0.2 Propagation and anonymity

To better understand our protocol it is useful to depict the propagation pattern of a transaction.

Let us consider an R node O generating a transaction tx. The following sequence of events happens:

1. R selects a proxy P among its proxy set, mark tx as $proxying$ and sends it to P;
2. P receives tx and proxy it with probability p, or diffuses it otherwise;
3. If proxying tx, P selects a node R from its proxy set S and sends it tx;

Proxying transactions are relayed through a sequence of R and U nodes until it gets diffused. Diffusion can happen at any step, except for the first one. Propagation from an U node follows a similar pattern.

A major risk of proxied broadcast is that a transaction might take too long to diffuse, or not be diffused at all. As for diffusion time, we can statistically guarantee to diffuse every transaction within a reasonable time. Since at every hop, the transaction tx is diffused with probability p, it is possible to tune this value to obtain a target number of hops through which tx is proxied on average. The use of timeouts allows dealing with a transaction not being diffused.

With respect to anonymity, ReaP is designed to be resistant against a first-spy estimator. This type of adversary connects to all R nodes and links each transaction to the first node from which it has been received. As demonstrated in [24], this strategy is very effective with the current propagation model. However, the changes introduced by our protocol make it very unlikely for a node to first receive a transaction from its source. On the contrary, most of the times, transactions will be received by a node different from the origin, thanks to proxying. Furthermore, each transaction is mixed with many others generated by nodes in the proxying

path, which are indistinguishable from each other to the receiving node. This means that any claim about the origin of a transaction can be easily denied.

Note that ReaP is designed to resist against very powerful adversaries controlling several nodes and maintaining multiple connections to all reachable nodes. The adversary can combine information from all of its nodes and coordinate them to influence or track the mixing set of a target node. However, we showed in the previous section how such an adversary has limited capabilities to affect the security of the protocol.

6.7.0.3 Ephimerality of U nodes

A possible issue in our design is the short time of connection of many U nodes. In fact, while R nodes are relatively stable [161], U nodes often experience very short-lived connections [22]. This behavior might affect the efficiency of the protocol. However, the timeout mechanism is also meant to deal with this kind of problems and can be fine-tuned independently by each node, depending on the experienced churn.

Moreover, the presence of short-lived proxy nodes, if properly exploited, might serve as an added value to the anonymity level of our protocol, as it makes it harder to track back a transaction to its origin.

6.7.0.4 NAT adoption

Another potential limitation of our solution is that it is based on the unreachability of NATted nodes. However, if IPv6 gets adopted by the majority of nodes, it is possible that NATs will cease to be used. The introduction of IPv6, in fact, was mainly intended to deal with the IPv4 address exhaustion problem and remove the need for NATting [168].

Although growing, the adoption rate of IPv6 seems to be variable [169] and not uniform worldwide, with statistics strongly dependent on the adopted metrics [170]. As for the Bitcoin network, Neudecker et al. [147] showed that, unlike IPv4, IPv6 connections have not grown over the past two years.

Either way, most optimistic estimates predict a complete adoption within 7-8 years [171]. In this perspective, the ReaP protocol should be considered as a medium-term solution, likely able to work for the next decade.

Chapter 7

TRANSACTION PROPAGATION: RAISING THE BAR OF BITCOIN ANONYMITY

In Section 3.3.2.6 we saw that the current Bitcoin transaction propagation protocol is vulnerable against deanonymization attacks. In particular, a work by Fanti et al. [24] demonstrated that transaction broadcast inherently leaks information about the source of a transaction to a network-wide observer. The reason behind this issue is the fact that new transactions are transmitted by the source to all connected peers. This allows an adversary that connects to all nodes to determine the source of a transaction by simply observing the first node to announce it.

The Dandelion protocol [25, 109] solves this issue by splitting transaction spread in two phases: the *stem* phase, in which the transaction is spread linearly, and the *fluff* phase, where the transaction is spread using Diffusion. In other words, nodes in the propagation line of the stem phase act as proxies: new transactions are not broadcast by their source but one of the nodes in the line.

In the previous chapter, we proposed an alternative protocol that leverages unreachable nodes to prevent an adversary from observing the spreading pattern of new transactions. In particular, reachable nodes proxy their transactions through unreachable nodes, and viceversa.

However, there are some flaws in its design. First of all, the presence of unreachable nodes depends on the use of NAT, which could eventually be completely replaced by IPv6. Secondly, when reachable nodes have too few unreachable peers (for example, right after joining the network), they are prevented from adopting the protocol. Finally, reachable peers are exposed to an adversary using unreachable nodes. In particular, this adversary can open multiple connections towards a target node to increase the chances of being selected as the first proxy. Additionally, the more inbound connections she controls, the more transactions

she can track in the proxy set of the target, thus narrowing down the its deanonymization set.

At the base of all these flaws there is the use of inbound connections for the relay of new transactions. In this chapter, we generalize the ideas of the previously proposed protocol and simplify its design to improve its efficiency and security.

7.1 Anonymous Transaction Propagation for the Bitcoin P2P Network

We propose a new transaction propagation protocol, called Clover, that breaks the symmetry in the relay pattern to protect the anonymity of the source. Our new protocol does not require determining the reachability of a peer and drastically reduces the probability of selecting an adversarial node as a proxy for new transactions. In particular, Clover eliminates the ability of eavesdropper adversaries to gain an advantage by opening multiple connections towards the same node. This is obtained by by separating the relay paths of inbound and outbound connections thus preventing the use of inbound peers as proxy nodes for new transactions. In other words, when a new transaction is created it is always relayed through an outbound peer. At the same time, we keep spreading proxied transactions by relaying messages from inbound peers to other inbound peers.

This approach is both secure and straightforward to implement. We evaluate it experimentally by using a Proof-of-Concept implementation in a simulated environment. Our results show that compared to Diffusion, our protocol reduces the lower-bound precision of an eavesdropper adversary applying the first-spy estimator from 0.6 to just 0.05, in the best case scenario, and from 0.7 to 0.3 in the worst case scenario.

7.2 The Clover protocol

This section describes our protocol, detailing and motivating its design with reference to the adversary model.

7.2.1 Adversary Model

We consider the eavesdropper adversary defined in [24], which is based on practical attacks such as [110] and [105]. This adversary makes use of a supernode that connects to all reachable nodes in the network. For each node, multiple connections can be established by using different IP:port addresses, making them look as

coming from different entities. In particular, the adversary can fill up all unused inbound slots of a target node.

The supernode listens to all the messages relayed by connected nodes, logging all contents and timestamps. Additionally, we extend this adversary to deploy reachable nodes in order to get peers of the network open connections toward them. This extension better represents the capabilities of the adversary against our protocol.

The goal of the adversary is to determine, for each transaction, its source in the network. We call *deanonymization set* the set of transactions known to the adversary that can be possibly generated by a specific node.

SOURCE ESTIMATION Different strategies can be employed to estimate the source of a transaction. We consider the so-called *first-spy* estimator, which has been shown to have high levels of accuracy against the Diffusion protocol [24].

This estimator follows a simple, yet effective, strategy: it associates each transaction to the first node that announces it. The effectiveness of this strategy is based on the network-wide observation of the adversary on the network: since the supernode is connected to all nodes, and since nodes announce new transactions to all of their peers, it is likely for the adversary to first receive a transaction from the node that originates it. Additionally, having multiple connections to each node further increases this probability.

7.2.2 Protocol Overview

Similarly to previous solutions [25], our protocol is based on two core features: proxied broadcast and transaction mixing. By *proxied broadcast* we mean the act of delegating the initial diffusion (broadcast) of a transaction to another node. The goal of proxying is to move the origin of the spreading process from the source of the transaction to a different node in the network.

A major risk in this approach is the ability of the *proxy node* to distinguish proxied transactions from diffused ones, which in turn allows for easy deanonymization. To mitigate this risk, we employ *transaction mixing*, which consists in having nodes sending transactions originated by other peers along with their own ones. This strategy allows to reduce the precision of the adversary when using the first-spy estimator. In particular, the more the transactions used for mixing, the lower the precision of the adversary. We call transactions used by a node for mixing its *mixing set*. In order to have transactions for the mixing set, we use multi-hop proxying, with each transaction relayed through multiple nodes before being broadcast. As a result, transactions are propagated in two phases: first, they are relayed through a number of proxy nodes, and then they are broadcast using

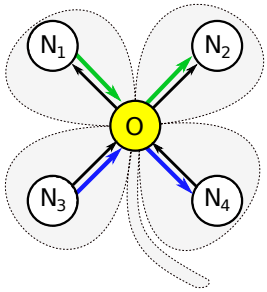

Figure 7.1: Clover relay

Diffusion. We call *proxying phase* the part of the propagation in which a transaction is relayed and *diffusing phase* the other part. A transaction in the proxying phase is called a *proxy transaction*.

To maximize the mixing property, we explicitly distinguish transactions in the proxying phase from those already diffused. This choice is based on the assumption that the eavesdropper adversary has knowledge about transactions that have already been diffused. Consequently, mixing proxy transactions with diffused ones can actually lower the quality of mixing, increasing the adversary precision. On the other hand, proxy transactions are likely unknown to the adversary, thus making them indistinguishable from those generated by the node that relays them. Moreover, by knowing which transactions are being proxied, it is possible to optimize their management for mixing, thus maximizing its effectiveness.

To further protect nodes, we only use outbound peers as proxies. This approach is motivated by the fact that the adversary can control an arbitrary number of inbound connections. For the same reason, we include in the mixing set only transactions received from outbound peers. In fact, an adversary controlling many inbound connections of a node can track all the transactions relayed to it, thus effectively reducing the actual size of the mixing set of the target. Instead, we relay proxy transactions from inbound peers to other inbound peers, thus allowing a correct propagation.

In summary, when a proxy transaction is received from an outbound peer, it is relayed to another outbound peer; instead, when a proxy transaction is received from an inbound peer, it is relayed to another inbound peer. We depict this scheme in 7.1. The name clover has been chosen to remind the four-way pattern of our scheme.

7.2.3 Protocol Design

Each transaction is created in the proxying phase and relayed to an outbound peer. At each hop, during this phase, the transaction is either broadcasted (via

Diffusion) or relayed to another node. In particular, when a node N receives a proxy transaction t, it behaves like follows: if t is received from an outbound peer, N relays it to a random outbound peer; if t is received from an inbound peer, N broadcasts it with probability p; otherwise, it relays it to a random inbound peer. When a transaction gets broadcast, it enters the diffusion phase and follows the standard Diffusion relay protocol.

Note that the broadcast step is only possible when t is received from inbound peers. This is done to ensure all transactions received from outbound peers are used for mixing.

PROXY TRANSACTIONS Proxy transactions are transmitted through a new protocol message, called `ptx`. This message contains the full transaction data, like the `tx` message.

Unlike the Diffusion protocol, we do not employ the three-step transmission via `inv` messages. In fact, these messages are meant to avoid sending a transaction to nodes that already have it. However, this is rarely supposed to happen in Clover. Instead, the receiver is always expected not to know the transaction being sent. This modification allows us to substantially reduce the delay introduced by the proxying phase, as each relay operation only requires one message instead of three.

TIMEOUT To deal with the risk of a transaction not being diffused, due to a DoS attack or an excessively long proxying phase, nodes set a timeout for each proxied transaction. To verify if a transaction has been diffused, nodes monitor `inv` messages from their outbound peers. If, before the timeout expires, the majority of the outbound peers has advertised the transaction, it is considered as correctly diffused. Otherwise, the transaction is broadcasted using Diffusion.

Again, we exclusively use outbound peers because they are the least likely to be controlled by the adversary. On the other hand, relying on inbound peers would allow an adversary controlling many of them to pretend the diffusion of a transaction by simply advertising it, thus generating a denial of service.

Note that the timeout-induced broadcast is applied to both own transactions and relayed ones so that the adversary is unable to distinguish which ones were created by the sender.

A default value for the timeout should be defined through experiments. However, each node might choose its own value.

PROPAGATION RULES We first define the *proxy* operation on a transaction tx and as in Algorithm 7.

Algorithm 7 Proxy procedure

ENV: t; $NodeSetOutPeers$

procedure PROXY(Transaction tx, NodeSet set)
    $proxyNode = pickRandomNode(set)$
    Send `ptx`(tx) to $proxyNode$

    $wait(t)$
    if $(|OutPeers|/2) + 1$ outbound peers announced tx **then**
        Mark tx as $diffused$
    else
        DIFFUSE(tx)

This operation consists in picking a random node from a set of peers and sending it a `ptx` message containing tx. If the message is being relayed, its sender is excluded from the candidates (to avoid sending a message back).

After sending the message, a timeout t is set. While t is not expired, the node collects `inv` messages from its outbound peers. When t expires, the node checks if the majority of outbound peers has announced tx. If not, the transaction is diffused.

Next, we define the following propagation rules:

- when a new transaction tx is created, run $Proxy(tx, outbound)$;
- when a transaction tx is received from an outbound peer, run $Proxy(tx, OutPeers)$;
- when a transaction tx is received from an inbound peer, run $Diffuse(tx)$ with probability p, otherwise run $Proxy(tx, InPeers)$.

The pseudocode of the Clover protocol is shown in Algorithm 8.

7.3 Analysis

In this section, we study the anonymity properties of the Clover protocol against an eavesdropper adversary using the first-spy estimator.

TERMINOLOGY We use R to denote the set of reachable nodes in the network, and A, to denote the subset of reachable nodes controlled by the adversary. Without loss of generality, we let I and O represent the average set of inbound and outbound peers of a node in the network.

Algorithm 8 Clover Propagation Rules

ENV: Probability p; NodeSet $OutPeers$,$InPeers$
if Create new transaction tx **then**
    $Proxy(tx, outbound)$
if Receive `ptx`(tx) from node $sender$ **then**
    if $sender$ is $outbound$ **then**
        $Proxy(tx, OutPeers)$
    else
        $d = getRandProb()$
        if $d < p$ **then**
            $Diffuse(tx)$
        else
            $Proxy(tx, OutPeers)$

7.3.1 Security

We first consider the eavesdropper adversary described in Section 7.2.1. As we will show, the adversary gains no advantage by connecting to all nodes, nor by establishing multiple connections towards the same node. In fact, in our protocol, new transactions are only relayed through outbound peers, making all inbound connections to the adversary useless for deanonymization. Instead, our adversary gains precision by deploying more reachable nodes, as this increases its chances of being selected as a proxy node for new transactions.

As such, two important aspects must be studied. First, we want to know the probability of selecting an adversarial node as proxy for new transactions. Then, we want to determine the size of the average mixing set for each node.

Using these values, we can calculate the precision of the adversary in deanonymizing proxy transactions, as well as its overall precision against all transactions in the network. Note that our adversary will mainly target proxy transactions, as the first-spy estimator will unlikely to be announced by its source.

PROXY SELECTION Assuming each reachable node has the same probability of being selected as outbound peer when a new node joins the network[1], we compute the probability of selecting an adversarial node as a proxy for a single transaction as follows:

Lemma 1. *Let $P_{AdvProxy}$ be the probability of selecting a node in A as a proxy for a new transaction. Then:*

[1] Although this assumption is theoretically sound, in the real Bitcoin network, well-established nodes tend to have more connections, especially compared to newly-joined nodes.

$$P_{AdvProxy} = \frac{|A|}{|R|}$$

Proof. As each node establishes $|O| = 8$ outbound connections, the probability of selecting a node in R as an outbound peer is $8/|R|$. Given the adversary control $|A|$ nodes in R, the probability of selecting a node in A as an outbound peer is $8|A|/|R|$.

Since new transactions are sent to a random node in O, the probability of selecting a node in A for a single new transaction is:

$$P_{AdvProxy} = \frac{1}{8} \cdot \frac{8|A|}{|R|} = \frac{|A|}{|R|}.$$

□

As such, in the current Bitcoin network, where $|R| \approx 10.000$, a single-node adversary would have $1/10000 = 0.0001$ probability of being selected as a proxy. Similarly, a powerful adversary controlling 1000 nodes (10% of the reachable network) would have 0.1 probability of being selected for each new transaction being sent in the network.

TRANSACTION MIXING To ease the analysis, we study the mixing property of a node over a period of time T.

We want to calculate the average size of the mixing set of a node, which is the number of `ptx` messages received from nodes in O. In the following we will use the word $transaction$ as a synonym of `ptx` message.

We use ρ_i and σ_i to denote the average number of transactions received from and sent to each node in I, respectively. Similarly, we use ρ_o and σ_o for nodes in O.

We study the size of the average mixing set MIX for a node having a adversarial outbound peers. Note that, when no adversaries are connected, the mixing set contains all transactions received from outbound peers (i.e., $|MIX| = \rho_o|O|$). However, if the adversary controls an outbound peer, the transactions received from such peer does not actually contribute to the mixing property, since they are known to the adversary. Therefore, the number of transactions in the mixing set is $|MIX| = \rho_o(|O| - a)$. Given the above, we have the following:

Lemma 2. *The cardinality of MIX is:*

$$|MIX| = \frac{g(1-p)}{p} \cdot \frac{|O| - a}{|O|}.$$

Proof. We consider the mixing set in the presence of a adversarial nodes among the outbound peers: $|MIX| = \rho_o(|O| - a)$. By definition, $\rho_o = \sigma_i$. Given the rules defined in Algorithm 8, transactions received from nodes in I (α_i) are relayed, with probability $1 - p$, uniformly at random among nodes in I. As such we have:

$$\sigma_i = (\alpha_i |I|(1-p))/|I| = \alpha_i(1-p).$$

By definition, $\alpha_i = \sigma_o$. Let us assume each node generates an average of g transactions during T. Given that each node sends to nodes in O all of its transactions along with those received by other nodes in O, we have: $\sigma_o = (g + \rho_o|O|)/|O|$

Given that $\rho_o = \sigma_i$ and $\alpha_i = \sigma_o$, we have:

$$\sigma_o = \frac{(g + \sigma_o(1-p)|O|}{|O|}.$$

Isolating σ_o, we get

$$\sigma_o = \frac{g}{|O|p}$$

On the other hand, as $\rho_o = \sigma_i = \alpha_i(1-p) = \sigma_o(1-p)$, we obtain:

$$\begin{aligned} |MIX| &= \rho_o(|O| - a) \\ &= \sigma_o(1-p)(|O| - a) \\ &= \frac{g}{|O|p}(1-p)(|O| - a) \\ &= \frac{g(1-p)}{p} \cdot \frac{|O|}{|O| - a} \end{aligned}$$

□

Note that the size of the mixing set is inversely proportional to p. In fact, the smaller this value, the longer a transaction will be relayed before getting diffused. In turn, the more a transaction is relayed, the more it contributes to the mixing of the other nodes.

DEANONYMIZATION PRECISION As previously mentioned, we expect the adversary to exclusively target proxy transactions, since it will be highly unlikely for him to receive diffused transactions from their source. Therefore, we first study the precision of the adversary against proxy transactions only, that is, the ones he receives. Afterwards, we compute the overall accuracy considering all transactions.

First, let us consider the precision against proxy transactions coming from a single node. Note that this only applies to nodes that opened a connections towards an adversarial peer.

Let D_{proxy} be the average precision of the adversary against proxy transactions coming from a single node. Then:

Lemma 3. *The average precision against proxy transactions for a single node is:*

$$D_{proxy} = \frac{p}{1 - \frac{a(1-p)}{|O|}}$$

Proof. We consider a node n generating g transactions, and being connected to a outbound peers controlled by the adversary. As both new and relayed transactions are distributed among nodes in O, each such node receives on average $g/|O|$ new transactions plus $|MIX|/|O|$ mixing transactions. Since the adversary associates all transactions to n, he will get $g/|O|$ correct guesses over $(g + |MIX|)$ transactions received.

By Lemma 2, we get:

$$\begin{aligned} D_{proxy} &= (g/|O|)/((g + |MIX|)/|O|) \\ &= g/(g + |MIX|) \\ &= g/(g + g\frac{1-p}{p}\frac{|O| - a}{|O|}) \\ &= \frac{p}{1 - \frac{a(1-p)}{|O|}} \end{aligned}$$

□

To calculate the overall precision, we consider a network of $|N|$ nodes, $|R|$ of which are reachable. Let $D_{overall}$ be the overall precision of the adversary against transactions generated by nodes in N. We have:

Lemma 4. *The overall precision is:*

$$D_{overall} = \frac{|A|}{|R|}$$

Proof. Let us consider all transactions generated by nodes in N, that is gN. By Lemma 1, each transaction is sent to an adversarial proxy with probability $|A|/|R|$. As such, the adversary will receive $gN(|A|/|R|)$ transactions from their source (thus guessing them correctly). Dividing correct guesses over the total amount of transactions, we have:

$$\frac{N \cdot \frac{|A|}{|R|} g}{gN} = \frac{|A|}{|R|}.$$

□

Therefore, the overall precision only depends on the portion of reachable nodes controlled by the adversary.

7.3.2 Efficiency

Like other similar solutions, the Clover protocol introduces a delay in the broadcast of a transaction. Specifically, this delay is determined by the hops through which transactions go during the proxying phase.

In this respect, two factors must be considered: the number of messages needed for each hop, and the number of hops.

HOP DELAY As described in Section 2.3.2, in the Bitcoin protocol, each transaction propagation hop requires three messages: `inv`, `getdata`, and `tx`. This strategy is used in Diffusion to avoid sending transaction data to nodes that already have it.

In Clover, this is not needed, since proxy transactions are normally unknown to the recipient. Instead, transaction data is transmitted directly with a single `ptx` message. This allows saving two extra messages for each hop, allowing for a larger number of hops without incurring into an excessive delay. In particular, three hops in the proxying phase roughly equal 1 hop in Diffusion.

PROXY HOPS As previously stated, a higher number of relays for each transaction corresponds to a higher mixing property for nodes in the network. Nevertheless, if this number is too high, it can produce an excessive propagation delay. Therefore, it is essential to choose a target value that seeks a compromise between efficiency and effectiveness.

Note that the average number of hops directly depends on the probability p. In particular, the lower this value, the higher the number of hops. Therefore, we can choose p to obtain a target number of hops (h).

In Section 7.4, we calculate the precise relation between p and h, and experimentally evaluate the best target number of hops.

7.3.3 Comparison to Dandelion

Similarly to Clover, the Dandelion protocol [25] and [109] consists of two phases: a first relay phase and a second broadcasting phase using Diffusion. Specifically,

the relay phase follow a random line and require nodes to always relay transactions over the same path to obtain mixing. To that purpose, a network-wide relay graph needs to be constructed among all network nodes. The construction of such graph is not trivial, and gives space to potential attack vectors. For instance, if the adversary learns the relay graph, it can deanonymize transactions with high precision. For this reason, the graph needs to be changed periodically.

In Clover, there is no need to build this graph, since mixing is obtained by using all incoming transactions. This makes its design much simpler, thus easing implementation and analysis.

Propagation delay is also improved, thanks to the use of `ptx` messages, which allow to transmit proxied transactions directly, instead of using the 3-step propagation. As such, the delay of one hop in Clover is roughly one third of the delay of one hop in Dandelion. This allows for longer relay phases, without incurring into excessive propagation time.

7.4 Experimental Results

7.4.1 Proof of Concept

We implemented our protocol by modifying the reference client (Bitcoin Core) and tested it on a simulated network. Clients were run in Regtest mode, using Docker containers. For comparison, simulations were also run against Diffusion, using unmodified clients.

SIMULATION SETTINGS Each test run on a network of 100 reachable nodes randomly connected among each other. Transactions are produced randomly for 10 minutes. In each run, we produce an average of 3 transactions per node (300 transactions in total). For each setting, we run 3 simulations and then computed the average.

Although unreachable nodes are theoretically relevant in Diffusion, studies [22] show how their involvement in the transaction propagation is extremely low compared to their number, with as little as the 0.001% of nodes sending transaction messages. We speculate that the vast majority of transactions is generated by reachable nodes and thus focus experiments on this part of the network. Note that this might slightly affect the results for the Diffusion protocol but has no relevance for Clover, since we showed in Section 7.3 how precision exclusively depends on reachable nodes.

ADVERSARY We varied the number of adversarial nodes from 1 to 30, which corresponds to a range between 1% and 30% of the reachable network. These

nodes are chosen randomly among the ones already deployed, so that they are well connected to the rest of the network. Note that this is the worst-case scenario since it assumes the adversary controls well-established nodes in the network. Additionally, when testing against Diffusion, each node connects to all reachable peers.

All adversarial nodes log incoming `inv` and `ptx` messages. At the end of the simulation, these logs are merged and ordered by timestamp. Then, the first-spy estimator is applied, linking each transaction to the first peer that advertised or transmitted it to any of the controlled nodes.

TIMEOUT Diffusion timeout has been set to fit to the local simulation environment, where transactions are produced and spread much more rapidly than the real network. In particular, verification timeout has been set to 1 minute.

7.4.2 Simulation Results

We evaluated precision against adversaries controlling 1%, 2%, 5%, 10%, 20%, and 30% of the network. Each adversary is tested against Diffusion, as well as Clover, for probability p equal to 0.2, 0.3, and 0.4. Overall precision is calculated as the average among all tests with the corresponding adversarial power. Results are shown in Figure 7.2.

Precision against Diffusion showed to be very high even for a very small number of adversarial nodes. In particular, controlling from 1% to 5% of the reachable network, the adversary has precision as high as 0.6. This value raise to 0.7 when the number of adversarial nodes reaches the 20% of the network.

On the other side, precision against Clover, although growing faster in the number of adversarial nodes, showed to be much lower than Diffusion. Specifically, overall precision is from 10 times smaller (0.05) for adversaries controlling from 1% to 5% of the network to 3 times smaller (0.33) for adversaries controlling from 10% to 30% of the network.

For what concerns precision against proxy transactions, we have, as expected, better results for lower values of p. In particular, experiments with $p = 0.2$ obtain the best results, with an average precision of 0.14 when the adversary controls 1-5%, up to 0.35 when he control 30% of the nodes. For $p = 0.3$, precision range from 0.16 to 0.4, while for $p = 0.4$ it ranges from 0.23 to 0.46.

Note that the increase in the precision against proxy transactions when the adversary controls a high percentage of the network is due to the fact that it is more likely for nodes to connect to more than 1 adversarial outbound peers.

Notably, in no case, the precision against Clover exceeded the one against Diffusion. This means that Clover against the strongest adversary controlling 30%

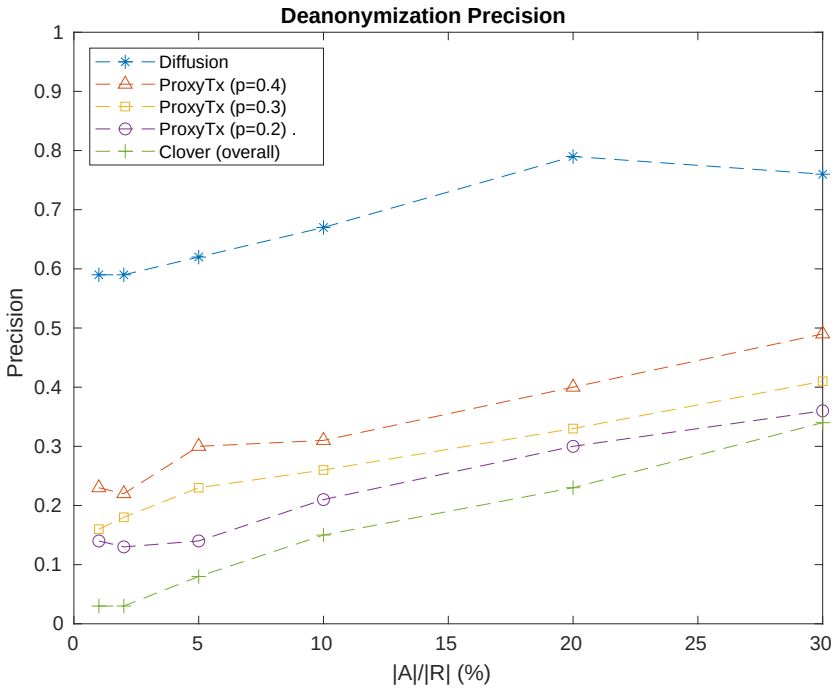

Figure 7.2: Deanonymization precision against Clover

of the network outperforms Diffusion against the weakest adversary controlling 1% of the network.

A major result of our experiments is that it shows how attacking our protocol would be extremely more expensive for the adversary, compared to Diffusion, without even reaching the same levels of accuracy.

HOPS According to our experimental results, the average number of hops is inversely proportional to the probability p. In particular, we found the following relation to hold:

$$h \approx \frac{(1-p)}{0.15}.$$

We believe the best compromise between efficiency and efficacy is to have around 6 hops.

Chapter 8

CONCLUSIONS AND FUTURE WORK

Over the past decade, blockchain networks have revolutionized the world of distributed systems. Although sharing structural and functional concepts with classic P2P networks, blockchains show fundamental characteristics that radically set them aside. This is mainly due to the goals they pursue, which demands different and new requirements. This is particularly true with respect to security aspects of network-layer communications. For this reason, targeted studies are needed to understand the properties of this new ecosystem.

As the first and most popular cryptocurrency, Bitcoin serves as the prototype for countless other permissionless blockchains. In particular, its network-layer protocol is often replicated by other cryptocurrencies with little or no modification. It is then not surprising that most network-level research on blockchain focuses on the Bitcoin P2P protocol.

Numerous investigations allowed to shed light upon many of the relevant security and efficiency properties of the Bitcoin network. Their results allowed to uncover and fix issues in the protocol, leading to major improvements for the network. Nevertheless, some topics still lack full comprehension and thus require further analyses.

With this thesis, we gave our contribution by exploring some of the aspects that had little coverage in the literature. We approached problems from a different perspective, trying not to be influenced by mainstream opinion but building our reasoning upon facts, as emerged from empirical observation and state-of-the-art research.

TESTNET In Chapter 4, we gave a deeper view of the Testnet network. We highlighted its differences from the Mainnet and showed how its peculiarities make it a suitable environment for real-world applications. We proved this by designing a

viable protocol for botnet Command & Control on top of Testnet.

Our design allows bots to be registered, funded, and controlled anonymously from any place. By leveraging non-standard transactions, the botnet is provided with advanced functionalities, such as encryption, authentication, bidirectional communication, and script storage. The proposed system is practical and economically affordable even for a low-resource attacker. Additionally, it is highly resilient to disruption and deanonymization, both appealing features for botmasters.

According to our estimates and experimental results, such a system could be used in real life to run a small spamming botnet or as a resilient component of a larger hybrid botnet. This calls for an effort in either limiting the possibility of misusing Bitcoin Testnet or devising appropriate countermeasures.

Furthermore, the existence of a similar threat demonstrates the fact that Testnet should not be considered as just an experimental environment but as a full-fledged blockchain with its own potential applications. As such, a more careful design of Testnet is needed, as well as broader and deeper analyses of its properties.

TOPOLOGY In Chapter 5, we tackled the problem of topology knowledge in Bitcoin. Specifically, we questioned the general agreement on the fact that the Bitcoin topology should be hidden. In fact, it is common belief that such information represents a security concern with relation to a number of network-level threats.

However, state-of-the-art research proved that the protective measures adopted in the protocol are can be often bypassed, allowing inferring the topology through side channels or by making observations on the information spreading.

At the same time, researchers pointed out how a closed topology prevents measurements that could help identifying weak points and thus improve efficiency and security. It is then licit to ask whether such measures are actually necessary. To answer this question, we empirically studied all the attacks typically related to topology knowledge. Our investigation showed that most of them do not actually depend on such knowledge, or have limited risk in practice.

In light of this, we argued for an open topology and proposed a protocol to reliably compute the topology of the network and monitor its changes over time. Our approach makes use of semi-trusted monitors and actively involves nodes in the protocol. We mitigate the risk of misbehaving actors with a reputation system based on feedback information from the monitors.

We experimentally evaluated our protocol through simulations, and showed it has high levels of accuracy (over 90%) with little overhead. Additionally, our results show high resilience against malicious nodes, even in high concentrations (up to 30%). Our scheme can be employed in any blockchain P2P network, with-

out modifications. Given the above, we endorse an open topology and the use of active monitoring for real-time protection and analysis of the network. We believe this approach would help designing a more efficient and provably-secure cryptocurrency.

UNREACHABLE NODES In Chapter 6, we took unreachable nodes into consideration. We showed how, despite being often overlooked in research, these nodes constitute a relevant part of the Bitcoin network. We argued that, by differentiating reachable and unreachable nodes in the protocol, it is possible to improve the robustness and efficiency of the network.

In that perspective, we proposed targeted modifications to the protocol such as increasing the number of outbound connections and disabling the advertisement of their address to peers. Both changes are trivial to implement and potentially bring benefits to the whole network, without introducing any overhead.

Furthermore, we observed how unreachable nodes are inherently protected from adversaries that need to actively open connections to their targets. Based on that, we designed a new transaction propagation protocol, called ReAP, that helps protecting user anonymity. We theoretically analyzed the security of our proposal against powerful adversaries and discussed the possible limitations of our approach. Our results show that ReAP potentially reduces the ability of an adversary to determine the source of a transaction.

TRANSACTION ANONYMITY In Chapter 7, we addressed the limitations of the transaction propagation protocol proposed in the previous chapter. More specifically, we aimed at reducing the attack surface to further improve the anonymity guarantees.

As a result, we proposed an alternative approach based on the separation between inbound and outbound connections. This new protocol, called Clover, has a simpler and more intuitive design that eases both its implementation and analysis.

We then gave theoretical proof of its anonymity properties and experimentally evaluated its effectiveness using a proof-of-concept implementation in a simulated environment. Experimental results showed that the overall precision of an eavesdropper adversary adopting the first-spy estimator drops from 0.6-0.7% to 0.05-0.3%, depending on the number of colluding nodes.

We believe our solution can be adopted in real cryptocurrency networks and serve as a basis for future developments in the field.

8.1 Future Work

As previously mentioned, the topics we explored in this thesis have not been extensively covered in research. As such, there are numerous directions that can be followed as a continuation of the results obtained. We here review the most relevant and immediate ones for each of the topic addressed.

TESTNET As demonstrated by our Testnet botnet, the use of non-standard transactions can an impact on the network. To date, there are no statistics on the use and characteristics of such transactions on Testnet. A targeted analysis would help understand whether they are used in practice and, if so, in which manner. Similarly, it would allow devising methods for detecting potentially dangerous or malicious activities. At the same time, a study on the impact that a system like the one we proposed could have on the functionality of the Testnet network.

As for the adversarial side, many aspects leave space for improvements. For instance, detectability is one of the main weaknesses of blockchain-based botnets. An improved version of our protocol could hinder detection by employing more random-looking transactions, or even prevent messages from being stored on the blockchain. Another example is the bot-funding system, which seems to represent the weak link of our system. An improved design could make this step harder to disrupt. Furthermore, the overall communication system could be optimized by using alternative data-embedding techniques, and expanded with more advanced capabilities. These kinds of improvements would help understand the actual magnitude of the threat so as to anticipate the development of defensive solutions.

TOPOLOGY Two interesting research directions can be followed to continue our work on a topology monitoring system. On the one side, the centralization of our solution can be an obstacle to its adoption. It is therefore important to design a fully-distributed protocol that avoids the need of trusted parties. On the other side, detection and reaction mechanisms should be devised for potential issues and threats. In other words, topology analysis techniques are needed, that address the requirements of a cryptocurrency network. We believe this kind of solutions would allow advancing towards a more adaptive and secure-by-design network.

UNREACHABLE NODES A variety of studies should be done to fully understand the role of unreachable nodes in the network. For instance, an analysis of their participation in the propagation of transactions would help estimate their actual influence on efficiency and security. Similarly, it would be important to analyze if and how including these nodes in previous security studies might affect their results.

As for the ReAP protocol, a more formal analysis is needed, to verify its anonymity guarantees. Similarly, a proof-of-concept implementation would allow performing experiments and compare results against the Diffusion and Clover protocols.

CLOVER Future work on Clover includes an experimental comparison with Dandelion and an analysis against adversaries using rumor-centrality-based estimators, like the *maximum-likelihood* method described in [24].

SIMULATION FRAMEWORK In many of the works described in this thesis, the simulation of the Bitcoin network played an essential role in the evaluation of the proposed solutions. The use of a realistic and thorough simulation environment is essential for obtaining valid results. Although a tailored environment can be developed for each experiment, having a generic simulation framework would be of great help in obtaining sound and standardized results. The realization of such a framework requires a thorough analysis of the actual Bitcoin network and comparative experiments that validate the simulator against the real-world.

8.2 Final Remarks

With this thesis, we tried to go beyond mainstream research on Bitcoin network layer, and delve into those topics that have been marginally covered in the literature. We proved that Testnet is a real blockchain network and should be regarded as such. We argued against the concealment of network topology and provided a first solution for its active monitoring. We showed the relevance of unreachable nodes and how their participation can be beneficial for the whole network. Finally, based on our results on unreachable nodes, we designed a simple but effective protocol for anonymous transaction propagation. Despite being developed for Bitcoin, most of our results are generic enough to be applicable to any permissionless blockchain.

Our results show that, while important milestones have been set for the understanding of blockchain P2P networks, much remains to be done. Most importantly, they prove that, in research, we should never take anything for granted. In particular, we would like to conclude with the following thoughts:

- The fact that nobody takes something into consideration does not necessarily mean it is not important;
- Unless theoretically or empirically proved, claims in a paper should always be counterchecked;

- Sometimes, important information is hidden in the details: always read the small print.

We strongly believe that these directives can guide researchers to a deeper and more thorough comprehension of this exciting new world called blockchain.

Bibliography

[1] Khan MK. Technological advancements and 2020. Telecommunication Systems. 2020;73:1–2. Available from: https://doi.org/10.1007/s11235-019-00647-8.

[2] Mercer D. Global Connected and IoT Device Forecast Update. Strategy Analytics Consumer Electronics. 2019;p. 6. Available from: https://www.strategyanalytics.com/access-services/devices/connected-home/consumer-electronics/reports/report-detail/global-connected-and-iot-device-forecast-update.

[3] Liao Y, Deschamps F, de Freitas Rocha Loures E, Ramos LFP. Past, present and future of Industry 4.0 - a systematic literature review and research agenda proposal. International Journal of Production Research. 2017;55(12):3609–3629. Available from: https://doi.org/10.1080/00207543.2017.1308576.

[4] OECD. Gross domestic spending on R&D [Internet page]; 2021. Available from: https://data.oecd.org/rd/gross-domestic-spending-on-r-d.htm. (Last accessed: 2021-01-21).

[5] Thames L, Schaefer D. In: Thames L, Schaefer D, editors. Industry 4.0: An Overview of Key Benefits, Technologies, and Challenges. Cham: Springer International Publishing; 2017. p. 1–33. Available from: https://doi.org/10.1007/978-3-319-50660-9_1.

[6] Chen W, Xu Z, Shi S, Zhao Y, Zhao J. A Survey of Blockchain Applications in Different Domains. In: Proceedings of the 2018 International Conference on Blockchain Technology and Application. ICBTA 2018. New York, NY, USA: Association for Computing Machinery; 2018. p. 17–21. Available from: https://doi.org/10.1145/3301403.3301407.

[7] Wang C, Li B. Peer-to-peer overlay networks: A survey. 2003;Available from: https://www.researchgate.net/

profile/Bo_Li16/publication/2944067_Peer-to-Peer_Overlay_Networks_A_Survey/links/553dfab00cf2c415bb0f882d.pdf.

[8] Pourebrahimi B, Bertels K, Vassiliadis S. A survey of peer-to-peer networks; 2005. Available from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.64.1218&rep=rep1&type=pdf.

[9] Palomar E, Estevez-Tapiador JM, Hernandez-Castro JC, Ribagorda A. Security in P2P Networks: Survey and Research Directions. In: Zhou X, Sokolsky O, Yan L, Jung ES, Shao Z, Mu Y, et al., editors. Emerging Directions in Embedded and Ubiquitous Computing. Berlin, Heidelberg: Springer Berlin Heidelberg; 2006. p. 183–192. Available from: https://link.springer.com/chapter/10.1007/11807964_19.

[10] Delgado-Segura S, Pérez-Solà C, Herrera-Joancomartí J, Navarro-Arribas G, Borrell J. Cryptocurrency networks: A new P2P paradigm. Mobile Information Systems. 2018;2018. Available from: https://doi.org/10.1155/2018/2159082.

[11] Nakamoto S. Bitcoin: A peer-to-peer electronic cash system; 2008. Available from: https://bitcoin.org/bitcoin.pdf.

[12] Bitcoin Core; 2020. (Last accessed: 2020-07-02). Available from: https://bitcoincore.org/.

[13] Böhme R, Christin N, Edelman B, Moore T. Bitcoin: Economics, Technology, and Governance. Journal of Economic Perspectives. 2015 May;29(2):213–38. Available from: https://www.aeaweb.org/articles?id=10.1257/jep.29.2.213.

[14] Blockchain.com - Blockchain Explorer; 2020. (Last accessed: 2020-12-11). Available from: https://www.blockchain.com.

[15] CoinMarketCap - Cryptocurrency Prices, Charts And Market Capitalizations; 2020. (Last accessed: 2020-12-11). Available from: https://coinmarketcap.com.

[16] Decker C, Wattenhofer R. Information propagation in the Bitcoin network. In: IEEE P2P 2013 Proceedings; 2013. p. 1–10. Available from: https://doi.org/10.1109/P2P.2013.6688704.

[17] Pappalardo G, Di Matteo T, Caldarelli G, Aste T. Blockchain Inefficiency in the Bitcoin Peers Network. EPJ Data Science. 2018;7(1):30. Available from: https://link.springer.com/article/10.1140/epjds/s13688-018-0159-3.

[18] Neudecker T, Hartenstein H. Network Layer Aspects of Permissionless Blockchains. IEEE Communications Surveys Tutorials. 2019;21(1):838–857. Available from: https://doi.org/10.1109/COMST.2018.2852480.

[19] Cao T, Yu J, Decouchant J, Verissimo P. Revisiting Network-Level Attacks on Blockchain Network. 2018;Available from: https://orbilu.uni.lu/bitstream/10993/38142/1/bcrb18-cao.pdf.

[20] Franzoni F, Abellan I, Daza V. Leveraging Bitcoin Testnet for Bidirectional Botnet Command and Control Systems. In: Bonneau J, Heninger N, editors. Financial Cryptography and Data Security. Cham: Springer International Publishing; 2020. p. 3–19. Available from: https://link.springer.com/chapter/10.1007/978-3-030-51280-4_1.

[21] Miller A, Litton J, Pachulski A, Gupta N, Levin D, Spring N, et al. Discovering bitcoin's public topology and influential nodes. 2015;Available from: https://allquantor.at/blockchainbib/pdf/miller2015topology.pdf.

[22] Wang L, Pustogarov I. Towards Better Understanding of Bitcoin Unreachable Peers. CoRR. 2017;abs/1709.06837. Available from: https://arxiv.org/abs/1709.06837.

[23] Franzoni F, Daza V. Improving Bitcoin Transaction Propagation by Leveraging Unreachable Nodes. In: 2020 IEEE International Conference on Blockchain (Blockchain); 2020. p. 196–203. Available from: https://doi.org/10.1109/Blockchain50366.2020.00031.

[24] Fanti GC, Viswanath P. Anonymity Properties of the Bitcoin P2P Network. CoRR. 2017;abs/1703.08761. Available from: http://arxiv.org/abs/1703.08761.

[25] Bojja Venkatakrishnan S, Fanti G, Viswanath P. Dandelion: Redesigning the Bitcoin Network for Anonymity. vol. 1. New York, NY, USA: ACM; 2017. p. 22:1–22:34. Available from: http://doi.acm.org/10.1145/3084459.

[26] Swan M. Blockchain: Blueprint for a new economy. O'Reilly Media, Inc.; 2015.

[27] Nofer M, Gomber P, Hinz O, Schiereck D. Blockchain. Business & Information Systems Engineering. 2017 Jun;59(3):183–187. Available from: https://doi.org/10.1007/s12599-017-0467-3.

[28] Zheng Z, Xie S, Dai HN, Chen X, Wang H. Blockchain challenges and opportunities: a survey. International Journal of Web and Grid Services. 2018;14(4):352–375. Available from: https://www.inderscienceonline.com/doi/abs/10.1504/IJWGS.2018.095647.

[29] Peterson LL, Davie BS. Computer networks: a systems approach. Elsevier; 2007.

[30] Bowden R, Keeler HP, Krzesinski AE, Taylor PG. Block arrivals in the Bitcoin blockchain. CoRR. 2018;abs/1801.07447. Available from: http://arxiv.org/abs/1801.07447.

[31] Cachin C, Vukolic M. Blockchain Consensus Protocols in the Wild. CoRR. 2017;abs/1707.01873. Available from: http://arxiv.org/abs/1707.01873.

[32] Neudecker T, Hartenstein H. Short Paper: An Empirical Analysis of Blockchain Forks in Bitcoin. In: Goldberg I, Moore T, editors. Financial Cryptography and Data Security. Cham: Springer International Publishing; 2019. p. 84–92. Available from: https://link.springer.com/chapter/10.1007/978-3-030-32101-7_6.

[33] Antonopoulos AM. Mastering Bitcoin: unlocking digital cryptocurrencies. O'Reilly Media, Inc.; 2014.

[34] De Leon DC, Stalick AQ, Jillepalli AA, Haney MA, Sheldon FT. Blockchain: properties and misconceptions. Asia Pacific Journal of Innovation and Entrepreneurship. 2017;Available from: https://doi.org/10.1108/APJIE-12-2017-034.

[35] Politou E, Casino F, Alepis E, Patsakis C. Blockchain Mutability: Challenges and Proposed Solutions. IEEE Transactions on Emerging Topics in Computing. 2019;p. 1–1. Available from: https://doi.org/10.1109/TETC.2019.2949510.

[36] Viriyasitavat W, Hoonsopon D. Blockchain characteristics and consensus in modern business processes. Journal of Industrial Information Integration. 2019;13:32 – 39. Available from: http://www.sciencedirect.com/science/article/pii/S2452414X18300815.

[37] Wust K, Gervais A. Do you Need a Blockchain? In: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT); 2018. p. 45–54. Available from: https://doi.org/10.1109/CVCBT.2018.00011.

[38] Ethereum.org; 2020. (Last accessed: 2020-12-18). Available from: https://ethereum.org.

[39] Hyperledger.org - Fabric; 2020. (Last accessed: 2020-12-18). Available from: https://www.hyperledger.org/use/fabric.

[40] Ripple.com; 2020. (Last accessed: 2020-12-18). Available from: https://ripple.com/.

[41] Bitcoin.org;. (Last accessed: 2020-12-18). Available from: https://bitcoin.org.

[42] Monero.org; 2020. (Last accessed: 2020-12-18). Available from: https://www.getmonero.org/.

[43] Wu M, Wang K, Cai X, Guo S, Guo M, Rong C. A Comprehensive Survey of Blockchain: From Theory to IoT Applications and Beyond. IEEE Internet of Things Journal. 2019;6(5):8114–8154. Available from: https://doi.org/10.1109/JIOT.2019.2922538.

[44] Reynard C. The 10 most popular cryptocurrencies in 2018; 2018. (Last accessed: 2020-12-22). Available from: https://www.telegraph.co.uk/technology/digital-money/top-10-popular-cryptocurrencies-2018/.

[45] Bitcoin Clients; 2020. (Last accessed: 2020-12-22). Available from: https://en.bitcoin.it/wiki/Clients.

[46] Bitcoin Protocol Documentation; 2020. (Last accessed: 2020-12-22). Available from: https://en.bitcoin.it/wiki/Protocol_documentation.

[47] Bitcoin Core integration/staging tree; 2020. (Last accessed: 2020-12-22). Available from: https://github.com/bitcoin/bitcoin.

[48] Neudecker T. Security and anonymity aspects of the network layer of permissionless blockchains. Ph.D. Thesis, KIT; 2019.

[49] Teachbitcoin.io - Bitcoin Transaction Overview; 2019. (Last accessed: 2020-12-26). Available from: https://teachbitcoin.io/presentations/transaction_build.html#/.

[50] Bitcoin Stack Exchange. What is meant by Bitcoin dust?; 2013. (Last accessed: 2020-12-26). Available from: http://bitcoin.stackexchange.com/questions/10986/what-is-meant-by-bitcoin-dust.

[51] Wiki B. OP_RETURN; 2018. (Last accessed: 2019-09-22). Available from: https://en.bitcoin.it/wiki/OP_RETURN.

[52] Bartoletti M, Pompianu L. An Analysis of Bitcoin OP_RETURN Metadata. In: Brenner M, Rohloff K, Bonneau J, Miller A, Ryan PYA, Teague V, et al., editors. Financial Cryptography and Data Security. Cham: Springer International Publishing; 2017. p. 218–230. Available from: https://link.springer.com/chapter/10.1007/978-3-319-70278-0_14.

[53] Merkle RC. A Digital Signature Based on a Conventional Encryption Function. In: Pomerance C, editor. Advances in Cryptology — CRYPTO '87. Berlin, Heidelberg: Springer Berlin Heidelberg; 1988. p. 369–378. Available from: https://link.springer.com/chapter/10.1007/3-540-48184-2_32.

[54] Wikipedia. Blockchain; 2020. (Last accessed: 2020-12-27). Available from: https://en.wikipedia.org/wiki/Blockchain.

[55] OnionCat - An Anonymous VPN-Adapter; 2020. (Last accessed: 2021-01-02). Available from: https://www.onioncat.org/.

[56] Bitnodes - Global Bitcoin Nodes Distribution; 2020. (Last accessed: 2020-12-16). Available from: https://bitnodes.earn.com/.

[57] Tor Project - Anonymity Online; 2020. (Last accessed: 2021-01-02). Available from: https://www.torproject.org/.

[58] Srisuresh P, Holdrege M. IP network address translator (NAT) terminology and considerations; 1999. Available from: http://www.hjp.at/doc/rfc/rfc2663.html.

[59] Biryukov A, Pustogarov I. Bitcoin over Tor isn't a Good Idea. In: 2015 IEEE Symposium on Security and Privacy; 2015. p. 122–134. Available from: https://doi.org/10.1109/SP.2015.15.

[60] Bitcoin Wiki - Protocol Documentation; 2019. (Last accessed: 2019-12-03). Available from: https://en.bitcoin.it/wiki/Protocol_documentation.

[61] Birman K. The Promise, and Limitations, of Gossip Protocols. SIGOPS Oper Syst Rev. 2007 Oct;41(5):8–13. Available from: https://doi.org/10.1145/1317379.1317382.

[62] Wikipedia. Poisson distribution; 2020. (Last accessed: 2020-12-31). Available from: https://en.wikipedia.org/wiki/Poisson_distribution.

[63] de Kwaasteniet A. Ranking crypto's by number of nodes; 2019. (Last accessed: 2020-12-28). Available from: https://medium.com/coinmonks/ranking-cryptos-by-number-of-nodes-57a12e4ae51a.

[64] Donet Donet JA, Pérez-Solà C, Herrera-Joancomartí J. The Bitcoin P2P Network. In: Böhme R, Brenner M, Moore T, Smith M, editors. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2014. p. 87–102. Available from: https://link.springer.com/chapter/10.1007/978-3-662-44774-1_7.

[65] Feld S, Schonfeld M, Werner M. Analyzing the Deployment of Bitcoin's P2P Network under an AS-level Perspective. Procedia Computer Science. 2014;32:1121 – 1126. The 5th International Conference on Ambient Systems, Networks and Technologies (ANT-2014), the 4th International Conference on Sustainable Energy Information Technology (SEIT-2014). Available from: http://www.sciencedirect.com/science/article/pii/S187705091400742X.

[66] Delgado-Segura S, Bakshi S, Pérez-Solà C, Litton J, Pachulski A, Miller A, et al. TxProbe: Discovering Bitcoin's Network Topology Using Orphan Transactions. CoRR. 2018;abs/1812.00942. Available from: http://arxiv.org/abs/1812.00942.

[67] Samant K, Bhattacharyya S. Topology, search, and fault tolerance in unstructured P2P networks. In: 37th Annual Hawaii International Conference

on System Sciences, 2004. Proceedings of the; 2004. p. 6 pp.–. Available from: https://doi.org/10.1109/HICSS.2004.1265682.

[68] ”bloXroute Team”. The scalability problem, (very) simply explained; 2018. (Last accessed: 2020-12-28). Available from: https://medium.com/bloxroute/the-scalability-problem-very-simply-explained-5c0656f6e7e6.

[69] Visa. 56,582 transaction messages per second!; 2020. (Last accessed: 2021-01-03). Available from: https://visatechmatters.tumblr.com/post/108952718025/56-582-transaction-messages-per-second.

[70] Kim S, Kwon Y, Cho S. A Survey of Scalability Solutions on Blockchain. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC); 2018. p. 1204–1207. Available from: https://doi.org/10.1109/ICTC.2018.8539529.

[71] Wang G, Shi ZJ, Nixon M, Han S. SoK: Sharding on Blockchain. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. AFT ’19. New York, NY, USA: Association for Computing Machinery; 2019. p. 41–61. Available from: https://doi.org/10.1145/3318041.3355457.

[72] Wirdum AV. How Falcon, FIBRE and the Fast Relay Network Speed Up Bitcoin Block Propagation (Part 2); 2016. (Last accessed: 2020-12-28). Available from: https://bitcoinmagazine.com/articles/how-falcon-fibre-and-the-fast-relay-network-speed-up-bitcoin-block-propagation-part-1469808784.

[73] McCorry P, Möser M, Shahandasti SF, Hao F. Towards Bitcoin Payment Networks. In: Liu JK, Steinfeld R, editors. Information Security and Privacy. Cham: Springer International Publishing; 2016. p. 57–76. Available from: https://link.springer.com/chapter/10.1007/978-3-319-40253-6_4.

[74] Zhou Q, Huang H, Zheng Z, Bian J. Solutions to Scalability of Blockchain: A Survey. IEEE Access. 2020;8:16440–16455. Available from: https://doi.org/10.1109/ACCESS.2020.2967218.

[75] Corallo M. Bitcoin Relay Network; 2019. (Last accessed: 2020-12-28). Available from: https://www.bitcoinrelaynetwork.org/.

[76] FIBRE - Fast Internet Bitcoin Relay Engine; 2019. (Last accessed: 2020-12-28). Available from: http://bitcoinfibre.org/.

[77] Falcon - A Fast Bitcoin Backbone; 2016. (Last accessed: 2020-12-28). Available from: https://falcon-net.org/.

[78] Douceur JR. The Sybil Attack. In: Druschel P, Kaashoek F, Rowstron A, editors. Peer-to-Peer Systems. Berlin, Heidelberg: Springer Berlin Heidelberg; 2002. p. 251–260. Available from: https://link.springer.com/chapter/10.1007/3-540-45748-8_24.

[79] Rosenfeld M. Analysis of Hashrate-Based Double Spending. CoRR. 2014;abs/1402.2009. Available from: http://arxiv.org/abs/1402.2009.

[80] Checkpoint Lockin; 2020. (Last accessed: 2021-01-03). Available from: https://en.bitcoin.it/wiki/Checkpoint_Lockin.

[81] Wiki B. Weaknesses; 2020. (Last accessed: 2021-01-03). Available from: https://en.bitcoin.it/wiki/Weaknesses.

[82] Eyal I, Sirer EG. Majority Is Not Enough: Bitcoin Mining Is Vulnerable. In: Christin N, Safavi-Naini R, editors. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2014. p. 436–454. Available from: https://link.springer.com/chapter/10.1007/978-3-662-45472-5_28.

[83] Nayak K, Kumar S, Miller A, Shi E. Stubborn Mining: Generalizing Selfish Mining and Combining with an Eclipse Attack. In: 2016 IEEE European Symposium on Security and Privacy (EuroS P); 2016. p. 305–320. Available from: https://doi.org/10.1109/EuroSP.2016.32.

[84] Ober M, Katzenbeisser S, Hamacher K. Structure and Anonymity of the Bitcoin Transaction Graph. Future Internet. 2013;5(2):237–250. Available from: https://www.mdpi.com/1999-5903/5/2/237.

[85] Herrera-Joancomartí J, Pérez-Solà C. Privacy in Bitcoin Transactions: New Challenges from Blockchain Scalability Solutions. In: Torra V, Narukawa Y, Navarro-Arribas G, Yañez C, editors. Modeling Decisions for Artificial Intelligence. Cham: Springer International Publishing; 2016. p. 26–44. Available from: https://link.springer.com/chapter/10.1007/978-3-319-45656-0_3.

[86] Reid F, Harrigan M. In: Altshuler Y, Elovici Y, Cremers AB, Aharony N, Pentland A, editors. An Analysis of Anonymity in the Bitcoin System. New York, NY: Springer New York; 2013. p. 197–223. Available from: https://doi.org/10.1007/978-1-4614-4139-7_10.

[87] Androulaki E, Karame GO, Roeschlin M, Scherer T, Capkun S. Evaluating User Privacy in Bitcoin. In: Sadeghi AR, editor. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 34–51. Available from: https://link.springer.com/chapter/10.1007/978-3-642-39884-1_4.

[88] Ron D, Shamir A. Quantitative Analysis of the Full Bitcoin Transaction Graph. In: Sadeghi AR, editor. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2013. p. 6–24. Available from: https://link.springer.com/chapter/10.1007%2F978-3-642-39884-1_2.

[89] Meiklejohn S, Pomarole M, Jordan G, Levchenko K, McCoy D, Voelker GM, et al. A Fistful of Bitcoins: Characterizing Payments Among Men with No Names. In: Proceedings of the 2013 Conference on Internet Measurement Conference. IMC '13. New York, NY, USA: ACM; 2013. p. 127–140. Available from: http://doi.acm.org/10.1145/2504730.2504747.

[90] Wu L, Hu Y, Zhou Y, Wang H, Luo X, Wang Z, et al.. Towards Understanding and Demystifying Bitcoin Mixing Services; 2020. Available from: https://arxiv.org/abs/2010.16274.

[91] Conti M, Sandeep Kumar E, Lal C, Ruj S. A Survey on Security and Privacy Issues of Bitcoin. IEEE Communications Surveys Tutorials. 2018;20(4):3416–3452. Available from: https://doi.org/10.1109/COMST.2018.2842460.

[92] Lamport L, Shostak R, Pease M. The Byzantine Generals Problem. ACM Trans Program Lang Syst. 1982 Jul;4(3):382–401. Available from: https://doi.org/10.1145/357172.357176.

[93] Karame GO, Androulaki E, Capkun S. Double-spending Fast Payments in Bitcoin. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security. CCS '12. New York, NY, USA: ACM; 2012. p. 906–917. Available from: http://doi.acm.org/10.1145/2382196.2382292.

[94] Bamert T, Decker C, Elsen L, Wattenhofer R, Welten S. Have a snack, pay with Bitcoins. In: IEEE P2P 2013 Proceedings; 2013. p. 1–5. Available from: https://doi.org/10.1109/P2P.2013.6688717.

[95] Neudecker T, Andelfinger P, Hartenstein H. A simulation model for analysis of attacks on the Bitcoin peer-to-peer network. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM); 2015. p. 1327–1332. Available from: https://ieeexplore.ieee.org/document/7140490.

[96] Apostolaki M, Zohar A, Vanbever L. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In: 2017 IEEE Symposium on Security and Privacy (SP); 2017. p. 375–392. Available from: https://doi.org/10.1109/SP.2017.29.

[97] Saad M, Cook V, Nguyen L, Thai MT, Mohaisen A. Partitioning Attacks on Bitcoin: Colliding Space, Time, and Logic. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS); 2019. p. 1175–1187. Available from: https://doi.org/10.1109/ICDCS.2019.00119.

[98] Rekhter Y, Li T, Hares S, et al.. A border gateway protocol 4 (BGP-4). ISI, USC Information Sciences Institute; 1994. Available from: http://www.hjp.at/doc/rfc/rfc4271.html.

[99] Pilosov A, Kapela T. Stealing the Internet: An Internet-scale man in the middle attack. NANOG-44, Los Angeles, October. 2008;p. 12–15. Available from: https://defcon.org/images/defcon-16/dc16-presentations/defcon-16-pilosov-kapela.pdf.

[100] Tran M, Choi I, Moon G, Vu AV, Kang M. A Stealthier Partitioning Attack against Bitcoin Peer-to-Peer Network. In: 2020 IEEE Symposium on Security and Privacy (SP). Los Alamitos, CA, USA: IEEE Computer Society; 2020. p. 515–530. Available from: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00027.

[101] Apostolaki M, Marti G, Müller J, Vanbever L. SABRE: Protecting Bitcoin against Routing Attacks. CoRR. 2018;abs/1808.06254. Available from: http://arxiv.org/abs/1808.06254.

[102] Singh A, Ngan T, Druschel P, Wallach DS. Eclipse Attacks on Overlay Networks: Threats and Defenses. In: Proceedings IEEE INFOCOM

2006. 25TH IEEE International Conference on Computer Communications; 2006. p. 1–12. Available from: https://doi.org/10.1109/INFOCOM.2006.231.

[103] Heilman E, Kendler A, Zohar A, Goldberg S. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In: 24th USENIX Security Symposium (USENIX Security 15). Washington, D.C.: USENIX Association; 2015. p. 129–144. Available from: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/heilman.

[104] Kaminsky D. Black ops of TCP/IP. Black Hat USA. 2011;44. Available from: https://www.slideshare.net/dakami/black-ops-of-tcpip-2011-black-hat-usa-2011.

[105] Koshy P, Koshy D, McDaniel P. An Analysis of Anonymity in Bitcoin Using P2P Network Traffic. In: Christin N, Safavi-Naini R, editors. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2014. p. 469–485. Available from: https://link.springer.com/chapter/10.1007/978-3-662-45472-5_30.

[106] Neudecker T, Hartenstein H. Could Network Information Facilitate Address Clustering in Bitcoin? In: Financial Cryptography and Data Security. Cham: Springer International Publishing; 2017. p. 155–169. Available from: https://link.springer.com/chapter/10.1007/978-3-319-70278-0_9.

[107] Shah D, Zaman T. Rumor Centrality: A Universal Source Detector. SIGMETRICS Perform Eval Rev. 2012 Jun;40(1):199–210. Available from: https://doi.org/10.1145/2318857.2254782.

[108] Shah D, Zaman T. Rumors in a Network: Who's the Culprit? IEEE Transactions on Information Theory. 2011;57(8):5163–5181. Available from: https://doi.org/10.1109/TIT.2011.2158885.

[109] Fanti G, Venkatakrishnan SB, Bakshi S, Denby B, Bhargava S, Miller A, et al. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. Proc ACM Meas Anal Comput Syst. 2018 Jun;2(2). Available from: https://doi.org/10.1145/3224424.

[110] Biryukov A, Khovratovich D, Pustogarov I. Deanonymisation of Clients in Bitcoin P2P Network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. CCS '14. New York, NY,

USA: ACM; 2014. p. 15–29. Available from: http://doi.acm.org/10.1145/2660267.2660379.

[111] Mastan ID, Paul S. A New Approach to Deanonymization of Unreachable Bitcoin Nodes. In: Capkun S, Chow SSM, editors. Cryptology and Network Security. Cham: Springer International Publishing; 2018. p. 277–298. Available from: https://link.springer.com/chapter/10.1007/978-3-030-02641-7_13.

[112] Lei M. Exploiting Bitcoin's topology for double-spend attacks. ETH Zurich; 2015. Available from: https://pub.tik.ee.ethz.ch/students/2015-FS/BA-2015-10.pdf.

[113] Nick J. Guessing Bitcoin's P2P Connections; 2015. (Last accessed: 2021-01-06). Available from: https://jonasnick.github.io/blog/2015/03/06/guessing-bitcoins-p2p-connections/.

[114] Neudecker T, Andelfinger P, Hartenstein H. Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network. In: 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld); 2016. p. 358–367. Available from: https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0070.

[115] Grundmann M, Neudecker T, Hartenstein H. Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin. In: Zohar A, Eyal I, Teague V, Clark J, Bracciali A, Pintore F, et al., editors. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2019. p. 113–126. Available from: https://link.springer.com/chapter/10.1007/978-3-662-58820-8_9.

[116] Patton M, Gross E, Chinn R, Forbis S, Walker L, Chen H. Uninvited Connections: A Study of Vulnerable Devices on the Internet of Things (IoT). In: 2014 IEEE Joint Intelligence and Security Informatics Conference; 2014. p. 232–235. Available from: https://doi.org/10.1109/JISIC.2014.43.

[117] O'Neill M. Insecurity by Design: Today's IoT Device Security Problem. Engineering. 2016;2(1):48. Available from:

http://www.engineering.org.cn/en/journal/eng/EN/abstract/article_17218.shtml.

[118] Bertino E, Islam N. Botnets and Internet of Things Security. Computer. 2017 feb;50(02):76–79. Available from: https://doi.org/10.1109/MC.2017.62.

[119] Antonakakis M, April T, Bailey M, Bernhard M, Bursztein E, Cochran J, et al. Understanding the Mirai Botnet. In: 26th USENIX Security Symposium (USENIX Security 17). Vancouver, BC: USENIX Association; 2017. p. 1093–1110. Available from: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/antonakakis.

[120] Kolias C, Kambourakis G, Stavrou A, Voas J. DDoS in the IoT: Mirai and other botnets. Computer. 2017;50(7):80–84. Available from: https://doi.org/10.1109/MC.2017.201.

[121] Feily M, Shahrestani A, Ramadass S. A Survey of Botnet and Botnet Detection. In: 2009 Third International Conference on Emerging Security Information, Systems and Technologies; 2009. p. 268–273. Available from: https://doi.org/10.1109/SECURWARE.2009.48.

[122] Mahmoud M, Nir M, Matrawy A, et al. A Survey on Botnet Architectures, Detection and Defences. IJ Network Security. 2015;17(3):264–281. Available from: http://dx.doi.org/10.6633%2fIJNS.201505.17(3).06.

[123] Ali ST, McCorry P, Lee PHJ, Hao F. ZombieCoin: Powering Next-Generation Botnets with Bitcoin. In: Brenner M, Christin N, Johnson B, Rohloff K, editors. Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2015. p. 34–48. Available from: https://link.springer.com/chapter/10.1007%2F978-3-662-48051-9_3.

[124] Frkat D, Annessi R, Zseby T. ChainChannels: Private Botnet Communication Over Public Blockchains. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData); 2018. p. 1244–1252. Available from: https://doi.org/10.1109/Cybermatics_2018.2018.00219.

[125] Silva SS, Silva RM, Pinto RC, Salles RM. Botnets: A survey. Computer Networks. 2013;57(2):378–403. Available from: https://www.sciencedirect.com/science/article/pii/S1389128612003568.

[126] Tyagi AK, Aghila G. A wide scale survey on botnet. International Journal of Computer Applications. 2011;34(9):10–23. Available from: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.5081&rep=rep1&type=pdf.

[127] Liu J, Xiao Y, Ghaboosi K, Deng H, Zhang J. Botnet: Classification, Attacks, Detection, Tracing, and Preventive Measures. EURASIP Journal on Wireless Communications and Networking. 2009 Sep;2009(1):692654. Available from: https://doi.org/10.1155/2009/692654.

[128] Binkley JR, Singh S. An Algorithm for Anomaly-based Botnet Detection. SRUTI. 2006;6:7–7. Available from: https://dl.acm.org/doi/abs/10.5555/1251296.1251303.

[129] Abu Rajab M, Zarfoss J, Monrose F, Terzis A. A Multifaceted Approach to Understanding the Botnet Phenomenon. In: Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement. IMC '06. New York, NY, USA: ACM; 2006. p. 41–52. Available from: http://doi.acm.org/10.1145/1177080.1177086.

[130] Livadas C, Walsh R, Lapsley D, Strayer WT. Using Machine Learning Techniques to Identify Botnet Traffic. In: Proceedings. 2006 31st IEEE Conference on Local Computer Networks; 2006. p. 967–974. Available from: https://doi.org/10.1109/LCN.2006.322210.

[131] Gu G, Zhang J, Lee W. BotSniffer: Detecting botnet command and control channels in network traffic. 2008;Available from: http://corescholar.libraries.wright.edu/cgi/viewcontent.cgi?article=1006&context=cse.

[132] Westervelt R. Botnet masters turn to google, social networks to avoid detection; 2009. (Last accessed: 2019-09-22). Available from: https://searchsecurity.techtarget.com/news/1373974/Botnet-masters-turn-to-Google-social-networks-to-avoid-detection.

[133] Wang P, Wu L, Aslam B, Zou CC. A Systematic Study on Peer-to-Peer Botnets. In: 2009 Proceedings of 18th International Conference on

Computer Communications and Networks; 2009. p. 1–8. Available from: https://doi.org/10.1109/ICCCN.2009.5235360.

[134] Nagaraja S, Mittal P, Hong CY, Caesar M, Borisov N. BotGrep: Finding P2P Bots with Structured Graph Analysis. In: Proceedings of the 19th USENIX Conference on Security. USENIX Security'10. Berkeley, CA, USA: USENIX Association; 2010. p. 7–7. Available from: http://dl.acm.org/citation.cfm?id=1929820.1929830.

[135] Saad S, Traore I, Ghorbani A, Sayed B, Zhao D, Lu W, et al. Detecting P2P botnets through network behavior analysis and machine learning. In: 2011 Ninth Annual International Conference on Privacy, Security and Trust; 2011. p. 174–180. Available from: https://doi.org/10.1109/PST.2011.5971980.

[136] Wiki B. Testnet; 2019. (Last accessed: 2019-09-22). Available from: https://en.bitcoin.it/wiki/Testnet.

[137] Lopp J. How to Solo Mine on Bitcoin's Testnet; 2015. (Last accessed: 2019-09-22). Available from: https://blog.lopp.net/how-to-solo-mine-on-bitcoin-s-testnet.

[138] Ali ST, McCorry P, Lee PHJ, Hao F. ZombieCoin 2.0: managing next-generation botnets using Bitcoin. International Journal of Information Security. 2018 Aug;17(4):411–422. Available from: https://doi.org/10.1007/s10207-017-0379-8.

[139] Zhong Y, Zhou A, Zhang L, Jing F, Zuo Z. DUSTBot: A duplex and stealthy P2P-based botnet in the Bitcoin network. PloS one. 2019;14(12). Available from: https://doi.org/10.1371/journal.pone.0226594.

[140] Kurt A, Erdin E, Cebe M, Akkaya K, Uluagac AS. LNBot: A Covert Hybrid Botnet on Bitcoin Lightning Network for Fun and Profit. In: Chen L, Li N, Liang K, Schneider S, editors. Computer Security – ESORICS 2020. Cham: Springer International Publishing; 2020. p. 734–755. Available from: https://link.springer.com/chapter/10.1007/978-3-030-59013-0_36.

[141] Yin J, Cui X, Liu C, Liu Q, Cui T, Wang Z. CoinBot: A Covert Botnet in the Cryptocurrency Network. In: Meng W, Gollmann D, Jensen CD, Zhou J, editors. Information and Communications Security. Cham: Springer International Publishing; 2020. p. 107–125. Available

from: https://link.springer.com/chapter/10.1007/978-3-030-61078-4_7.

[142] Baden M, Ferreira Torres C, Fiz Pontiveros BB, State R. Whispering Botnet Command and Control Instructions. In: 2019 Crypto Valley Conference on Blockchain Technology (CVCBT); 2019. p. 77–81. Available from: https://doi.org/10.1109/CVCBT.2019.00014.

[143] Oliveira A, GonÃ§alves V, Filho GR. Using Ethereum Smart Contracts for Botnet Command and Control; 2020. Copyright - Copyright Academic Conferences International Limited Jun 2020. Available from: https://search.proquest.com/conference-papers-proceedings/using-ethereum-smart-contracts-botnet-command/docview/2453793786/se-2?accountid=14708.

[144] Zarpelão BB, Miani RS, Rajarajan M. Detection of Bitcoin-Based Botnets Using a One-Class Classifier. In: Blazy O, Yeun CY, editors. Information Security Theory and Practice. Cham: Springer International Publishing; 2019. p. 174–189. Available from: https://link.springer.com/chapter/10.1007/978-3-030-20074-9_13.

[145] Correia P, Rocha E, Nogueira A, Salvador P. Statistical characterization of the Botnets C&C traffic. Procedia Technology. 2012;1:158–166. Available from: https://doi.org/10.1016/j.protcy.2012.02.030.

[146] Mascheck S. ARG_MAX, maximum length of arguments for a new process; 2016. (Last accessed: 2019-09-22). Available from: https://www.in-ulm.de/~mascheck/various/argmax.

[147] Neudecker T. Characterization of the Bitcoin Peer-to-Peer Network (2015-2018). Karlsruher Institut für Technologie (KIT); 2019. 1. Available from: https://doi.org/10.5445/IR/1000091933.

[148] Grundmann M, Neudecker T, Hartenstein H. Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin. In: Financial Cryptography and Data Security. Berlin, Heidelberg: Springer Berlin Heidelberg; 2019. p. 113–126. Available from: https://link.springer.com/chapter/10.1007/978-3-662-58820-8_9.

[149] Group DR. Bitcoin Network Monitor - Transaction Propagation; 2019. (Last accessed: 2020-01-16). Available from: https://dsn.tm.kit.edu/bitcoin/videos.html#transactions.

[150] Fanti G, Viswanath P. Deanonymization in the Bitcoin P2P Network. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. NIPS'17. Red Hook, NY, USA: Curran Associates Inc.; 2017. p. 1364–1373. Available from: https://dl.acm.org/doi/10.5555/3294771.3294901.

[151] Kiayias A, Panagiotakos G. Speed-Security Tradeoffs in Blockchain Protocols; 2015. Cryptology ePrint Archive, Report 2015/1019. Available from: https://eprint.iacr.org/2015/1019.

[152] Essaid M, Kim HW, Park WG, Lee KY, Park SJ, Ju HT. Network Usage of Bitcoin Full Node. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC); 2018. p. 1286–1291. Available from: https://ieeexplore.ieee.org/abstract/document/8539723.

[153] Jia Zhao, Jiande Lu. Solving Overlay Mismatching of Unstructured P2P Networks using Physical Locality Information. In: Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06); 2006. p. 75–76. Available from: https://ieeexplore.ieee.org/abstract/document/1698595.

[154] Liu Y, Xiao L, Ni L. Building a Scalable Bipartite P2P Overlay Network. IEEE Transactions on Parallel and Distributed Systems. 2007 Sep;18(9):1296–1306. Available from: https://ieeexplore.ieee.org/abstract/document/4288128.

[155] Dotan M, Pignolet YA, Schmid S, Tochner S, Zohar A. SOK: Cryptocurrency Networking Context, State-of-the-Art, Challenges. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. ARES '20. New York, NY, USA: Association for Computing Machinery; 2020. Available from: https://doi.org/10.1145/3407023.3407043.

[156] Deshpande V, Badis H, George L. BTCmap: Mapping Bitcoin Peer-to-Peer Network Topology. In: 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN); 2018. p. 1–6. Available from: https://doi.org/10.23919/PEMWN.2018.8548904.

[157] Rostami H, Habibi J. Topology awareness of overlay P2P networks. Concurrency and Computation: Practice and Experience. 2007;19(7):999–1021. Available from: https://doi.org/10.1002/cpe.1089.

[158] Yunhao Liu, Xiaomei Liu, Li Xiao, Ni LM, Xiaodong Zhang. Location-aware topology matching in P2P systems. In: IEEE INFOCOM 2004. vol. 4; 2004. p. 2220–2230 vol.4. Available from: https://doi.org/10.1109/INFCOM.2004.1354645.

[159] Zhiyong Xu, Rui Min, Yiming Hu. HIERAS: a DHT based hierarchical P2P routing algorithm. In: 2003 International Conference on Parallel Processing, 2003. Proceedings.; 2003. p. 187–194. Available from: https://doi.org/10.1109/ICPP.2003.1240580.

[160] Oliveira LB, Siqueira IG, Loureiro AAF. On the Performance of Ad Hoc Routing Protocols under a Peer-to-Peer Application. J Parallel Distrib Comput. 2005 Nov;65(11):1337–1347. Available from: https://doi.org/10.1016/j.jpdc.2005.05.023.

[161] info S. Peers; 2020. (Last accessed: 2020-01-16). Available from: https://statoshi.info/dashboard/db/peers.

[162] info S. P2P Messages; 2020. (Last accessed: 2020-01-16). Available from: https://statoshi.info/dashboard/db/p2p-messages.

[163] Mariem SB, Casas P, Romiti M, Donnet B, StÃ¼tz R, Haslhofer B. All that Glitters is not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network; 2020. Available from: https://arxiv.org/abs/2001.09105.

[164] Essaid M, Park S, Ju HT. Bitcoin's dynamic peer-to-peer topology. International Journal of Network Management. 2020;n/a(n/a):e2106. E2106 nem.2106. Available from: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2106.

[165] Rossow C, Andriesse D, Werner T, Stone-Gross B, Plohmann D, Dietrich CJ, et al. SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets. In: 2013 IEEE Symposium on Security and Privacy; 2013. p. 97–111. Available from: https://doi.org/10.1109/SP.2013.17.

[166] Richter P, Allman M, Bush R, Paxson V. A Primer on IPv4 Scarcity. SIGCOMM Comput Commun Rev. 2015 04;45(2):21–31. Available from: https://doi.org/10.1145/2766330.2766335.

[167] Hu Z. NAT traversal techniques and peer-to-peer applications. In: HUT T-110.551 Seminar on Internetworking; 2005. p. 04–26. Available from: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1659&rep=rep1&type=pdf.

[168] Van de Velde G, Hain T, Droms R, Carpenter B. Local Network Protection for IPv6. RFC 4864, May; 2007. Available from: http://www.hjp.at/doc/rfc/rfc4864.html.

[169] McCarthy K. IPv6 growth is slowing and no one knows why. Let's see if El Reg can address what's going on; 2018. (Last accessed: 2020-07-14). Available from: https://www.theregister.com/2018/05/21/ipv6_growth_is_slowing_and_no_one_knows_why/.

[170] Czyz J, Allman M, Zhang J, Iekel-Johnson S, Osterweil E, Bailey M. Measuring IPv6 Adoption. SIGCOMM Comput Commun Rev. 2014 Aug;44(4):87–98. Available from: https://doi.org/10.1145/2740070.2626295.

[171] Howard L. IPv6 Growth; 2019. (Last accessed: 2020-07-14). Available from: https://www.retevia.net/ipv6-growth/.