Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bitcoin versus Bitshark – Estimating Privacy via Data Collection and Combination

Master Thesis

Lorenzo Wölckner
lorenzow@student.ethz.ch

August 10, 2014

ETH Zürich
Department of Computer Science
Institute of Information Security
System Security Group

Advisors:
Prof. Dr. Srdjan Capkun
Dr. Ghassan Karame
Dipl.-Ing. Arthur Gervais

Abstract

Bitcoin is becoming more and more influential as an alternative type of money; the cryptocurrency advertises fast, decentralized, secure payments, but what are the privacy risks? This thesis has the objective of inferring as much information possible about Bitcoin addresses, the pseudonyms of the system, finding the entities they belong to (or not), by discovering related or unrelated addresses. We can then indirectly estimate the privacy of its users; as the latter is provided in Bitcoin via pseudonyms, we demonstrate how it is often possible to find a big amount of related and, more importantly, unrelated addresses, leading to a decrease in the provided pseudonymity level. We analyse the information that can be passively extracted from the communications and data stored in the network, proposing a way of combining different knowledge coming from multiple sources. We show how the advertised pseudonymity level coming from using different and new Bitcoin addresses is much lower than expected, since many addresses can be shown to be unrelated to each other. A custom built client being run on multiple servers is used to collect new, additional types of data, that allow for a more detailed and precise view of the users participating in the Bitcoin system through the assignment of transactions to connected peers. Timing analysis is used to form a link between Bitcoin addresses and connected IPs with their relative online time, thus offering new possibilities for identifying related and unrelated addresses. We aggregate previously known and new data collected about the network and its Bitcoin addresses in a fully automatic way, reporting the needed computational resources and obtained hiding set sizes describing the estimated privacy of different addresses provided by the Bitcoin network.

Keywords:
Bitcoin, Cryptocurrency, Bitshark, Anonymity, Privacy, Pseudonymity

Acknowledgements

I would like to thank my advisors Dr. Ghassan Karame and Arthur Gervais for their invaluable support, ideas and motivation that helped me research, experiment and describe everything I did in my thesis. I also appreciate the efforts of Prof. Dr. Sdrjan Capkun and his research group and ETH for providing me with everything I needed to run all the software and for giving me the possibility of working on such an interesting topic. Working together in this field and having the freedom of developing my own ideas and approaches in order to solve whatever challenge appeared before me has been a wonderful and invaluable experience. My thanks also go to my family and friends, who supported me throughout the entire project.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

With my signature I confirm that:

- I have committed none of the forms of plagiarism described in the "Citation etiquette"[1] information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature**
Zürich, 10.08.2014	[illegible]

[1] **https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf**

Contents

1 Introduction **1**
1.1 Related Work 1
1.2 Contribution and Goals 5
1.3 Structure 6

2 Theoretical Background **7**
2.1 Economics 7
2.2 Cryptography 8
2.2.1 Hash 8
2.2.2 Merkle Tree 9
2.2.3 Digital Signature 9
2.2.4 Elliptic Curve DSA 10

3 Bitcoin **11**
3.1 Components 12
3.1.1 Address 12
3.1.2 Wallet 13
3.1.3 Transaction 13
3.1.4 Block 15
3.1.5 Network 19
3.1.6 Communication Protocol 21
3.1.7 Client 22
3.2 How Everything Works Together 23
3.3 Simplified Payment Verification 23
3.4 Security and Attacks 25
3.5 Anonymity and Privacy 27
3.6 Alternative Uses 29

4 Alternative Cryptocurrencies 31
4.1 Selected Altcoins . . . 31
4.1.1 Namecoin . . . 32
4.1.2 Litecoin . . . 33
4.1.3 Mastercoin . . . 33
4.1.4 Peercoin . . . 34
4.1.5 Dogecoin . . . 35
4.1.6 Zerocoin/Zerocash . . . 36
5 Bitshark 37
5.1 System Description . . . 37
5.1.1 Architecture and Deployment . . . 38
5.1.2 Workers . . . 40
5.1.3 Middleware . . . 43
5.1.4 Worker-Middleware Communication . . . 46
5.1.5 Database and Logged Data . . . 47
5.1.6 Analyser Server . . . 49
5.2 Statistics . . . 49
5.3 Changes and Improvements . . . 50
6 Data Analysis 53
6.1 Transaction Relay Details . . . 53
6.2 Assignment of Transactions to Entities . . . 57
6.3 Analyser . . . 61
6.3.1 Software Description . . . 62
6.3.2 Utilized Information . . . 63
6.3.3 Information Combination . . . 66
6.3.4 Program Flow and Structure . . . 69
7 Experiments 71
7.1 Set-up and Connections . . . 71
7.2 Results and Observations . . . 73
7.2.1 Resource Usage . . . 73
7.2.2 Heuristics' Contribution . . . 76
7.2.3 Hiding Set Size . . . 79
8 Conclusion 85
8.1 Bitcoin Improvements Proposals . . . 85
8.2 Future Work . . . 87
Glossary 89
Appendix 95
A.1 MongoDB Document Examples . . . 95
A.2 Bitshark Protocol Functions . . . 97
A.3 Bitcoin Client Code Extracts . . . 100
Bibliography 105
Additional References 109

Chapter 1

Introduction

Since its introduction years ago, Bitcoin has seen a rapid increase in popularity, quickly becoming the most popular and used cryptocurrency around the world. The system ensures secure fast transactions in a decentralized peer-to-peer network, where all information is publicly transferred and shared. Transferring bitcoins is (as of now) considered secure, while privacy is ensured by the utilization by its users of multiple different addresses, used as pseudonyms for sending and receiving money.

Due to its novelty, its raising global acceptance and increasing volume of transferred money, Bitcoin is a very interesting subject for the IT community, especially for discussions regarding the security and soundness of the system. This thesis wants to be a compendium of the information currently available about Bitcoin and its promised level of protection. It will focus then on the data that external entities could extract by listening to the communications in the network, on what can be inferred by combining all the information together from different sources and on what this means for Bitcoin users' pseudonymity and privacy.

1.1 Related Work

Following Bitcoin's growth, more and more papers and articles have been written about the subject, since the system is relatively new. All literature consists of documents written and published during the last five years; we concentrate in particular on texts dealing with the subject of privacy, anonymity and security in the Bitcoin system.

The ideas at the base of Bitcoin have been originally published in the paper "Bitcoin: A Peer-to-Peer Electronic Cash System" **[Nak08]**, in 2008. The system saw a rapid increase in number of users and academic interest after few years, especially since the community assumed the development work.

In one of the first papers, "Bitcoin Gateway – A Peer-to-Peer Bitcoin Vault and Payment Network" **[Sye11]**, a proposal for a peer-to-peer system for Bitcoin wallets and payments is made. The idea is to add an outer layer to the whole network, to simplify the usage of the Bitcoin service for new users and at the same time improve the security of the system.

In "Bitcoin: An Innovative Alternative Digital Currency" **[Gri12]** a review of the status of Bitcoin is presented; the focus is on its effects on the economic world and the consequences for money laws around the world. "Bitter to Better – How to Make Bitcoin a Better Currency" **[Bar12]** presents a series of problems found in the way Bitcoin works, followed by possible solutions and improvements for the cryptocurrency's situation. In the paper "On Bitcoin and Red Balloons" **[Bab12]** the authors explain the problem of incentives and competing entities which can be found in the Bitcoin system, proposing modifications of the protocol that would ensure information propagation through the Bitcoin network and all of its participating nodes.

The document "Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency" **[Bec13]** discusses the viability of the Proof of Work system used in Bitcoin; it is shown if and how, in different situations, the system remains secure. The topic is further discussed in "Majority Is Not Enough: Bitcoin Mining Is Vulnerable" **[Eya13]**, where through calculations it's demonstrated how Bitcoin can be vulnerable to some types of attacks where miners collude against the system; the authors show why the mining system cannot be incentive compatible and how it can be exploited. In "The Economics of Bitcoin Mining, or Bitcoin in the Presence of Adversaries" **[Kro13]** the mining system is again analysed; Bitcoin is examined as a consensus game and it's argued that, to prevent certain attacks on the currency, a completely decentralized system cannot possibly be maintained in the future and has to be changed by making compromises.

The paper "Anonymity of Bitcoin Transactions – An Analysis of Mixing Services" **[Mös13]** takes a look at mixing services and their security; it's shown how their real effect on privacy varies and how risky it is to use such systems at present. In "Beware the Middleman: Empirical Analysis of Bitcoin-Exchange Risk" **[Moo13]** the authors analyse the risks involved in the usage of exchanges to switch between bitcoins and real currencies; statistics are also presented about the security of multiple exchange services.

In "Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network" **[Kon13]** the authors perform a thorough analysis of the transactions and money moved in the Bitcoin system; statistics about addresses and their utilization are presented, together with data regarding many different properties of the bitcoins circulating (and resting) in the network.

The document "Is Bitcoin a Decentralized Currency?" **[Ger13]** focuses on the issue of complete decentralization in the Bitcoin system; it is shown how some operations and decisions are taken by a restricted set of entities, effectively controlling the system. Proposals are made to solve the current issues and enhance the system to be more decentralized than it currently is.

"Zerocoin: Anonymous Distributed E-Cash from Bitcoin" **[Mie13]** presents a series of promising improvements to Bitcoin's protocol and system, dealing with the insufficient privacy provided to users. It is shown how Bitcoin doesn't provide good anonymity guarantees to its users and can leak information about money and executed transactions, while proposing a new currency (extension of Bitcoin) that provides advanced features in this regard through the usage of new cryptographic primitives. With "Zerocash: Decentralized Anonymous Payments from Bitcoin" **[Ben14]** the group of persons who worked on Zerocoin shows the latest developments in the project, which is now presented as a completely new alternative cryptocurrency (altcoin). Zerocash advertises strong anonymity features, with transactions revealing neither inputs nor outputs nor the amounts of money being transferred. In the paper "Hiding Transaction Amounts and Balances in Bitcoin" **[And14]** the authors provide additional (alternative) improvements for the original Zerocoin system; they improve its anonymity guarantees by keeping information regarding money balances and transferred amounts hidden and impossible to find by looking at public information.

"Increasing Anonymity in Bitcoin" **[Sax14]** proposes a new method aimed at improving the users' anonymity in the Bitcoin system. Through the introduction of a new cryptographic primitive, the authors advertise the creation of transactions which would no longer link input to output addresses, increasing the anonymity degree and bringing additional advantages.

Several publications deal with the privacy of the users of the system, information propagation through the network (plus related attacks), and analyses about what's possible to infer from data that is publicly available for everyone. These topics are at the core of the research presented in this thesis; the most relevant documents are here presented in order of time.

In "An Analysis of Anonymity in the Bitcoin System" **[Rei11]** the authors try to map external information coming from the TCP/IP layer and web scraping data to associate Bitcoin addresses to IP addresses; in addition to that they try to follow interesting movements of money across multiple transactions in history. The network of transactions and addresses is built and analysed, in an effort to show the difficulty involved in relating and grouping different addresses and de-anonymizing users.

"Evaluating User Privacy in Bitcoin" **[And12]** presents a metric for quantifying privacy in the Bitcoin network and provides two new important heuristics, which allow to automatically analyse public data from the blockchain to relate and group addresses, forming information about entities and their links. The success of behaviour-based clustering techniques involving the previously shown new methods is demonstrated through practical experiments by simulating usage scenarios of the Bitcoin network.

In the document "Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin" **[Kar12]** the authors concentrate on fast payments through Bitcoin, their weaknesses and possible attacks. An analysis of the behaviour of clients accepting unconfirmed transactions and of message propagation is presented; it is shown how double-spending attacks via fast conflicting transaction messages are possible through practical experiments, possible solutions to the problem are proposed.

In the paper "A Fistful of Bitcoins: Characterizing Payments Among Men With No Names" **[Mei13]** a series of improvements of the previous two heuristics is proposed, together with new results of their application and their implication for privacy. The authors use a mix of automatic tools and manual tagging of data to build a detailed clustering method, leading to a very accurate map of the relations between addresses and to extensive statistics about the network, its users and their links with real-world entities.

"Have a Snack, Pay with Bitcoins" **[Bam13]** presents again the problem of double-spending attacks in relation with fast payments and unconfirmed transactions. Improvements of the protocol in these scenarios are proposed, following an analysis of node communication through the bitcoin network.

In "Quantitative Analysis of the Full Bitcoin Transaction Graph" **[Ron13]** it is shown how it's possible to navigate the history and track movements of money across addresses, finding interesting transaction patterns and relations. The document focuses on the static analysis of data in the Bitcoin blockchain, and demonstrates how easy it is to build a detailed log of exactly where money arrived and went for specific entities.

The document "Structure and Anonymity of the Bitcoin Transaction Graph" **[Obe13]** presents statistics about the amount and types of entities interacting in the network, and gives more information regarding address usage over time in relation to the automatic heuristics. An analysis of the relations between different network parameters, entities and addresses is presented.

The paper "Information Propagation in the Bitcoin Network" **[Dec13]** analyses in detail the broadcast and relay of blocks and transactions in the Bitcoin network. Models and statistics about different times and behaviours are presented, with proposals for increasing information propagation speed.

"The Bitcoin P2P Network" **[Don14]** gives statistics about clients and their behaviour in the Bitcoin network; statistics about their online time, stability and other related data are shown. A series of measurements about transaction and block propagation times through the different nodes is given.

"An Exploration of Bitcoin Anonymity" **[Woo14]** presents an accurate description of Bitcoin's anonymity issues and a survey of the works about the subject. Security problems and attacks are shown with their relative proposed solutions, together also with possible privacy improvements.

Finally, the document "An Analysis of Anonymity in Bitcoin Using P2P network Traffic" **[Kos14]** demonstrates that particularly rare patterns in the timings of received message inside the network can lead to direct and very probable associations of Bitcoin addresses to IP addresses. The authors explain how the different possible scenarios can be identified and categorized, recognizing how this type of association can very negatively affect privacy.

1.2 Contribution and Goals

Bitcoin is not an anonymous system; every transaction is logged publicly, indicating the amount of money being transferred, when it happened, and the addresses involved. Bitcoin is known however to provide privacy through the pseudonymity given by the multiple different addresses everyone is free to create and use, masking the identities of the users in the network.

The goal of our work is to infer as much information as possible about observed addresses, finding all possible related and unrelated addresses, and building a map of these relations. We combine old and new analysis techniques in order to reach our objective and find associations between addresses and entities. We present estimates of the pseudonymity level provided by Bitcoin, given by the number of addresses an address can "hide" into. In an optimal scenario this number should be equal to all the existing pseudonyms, while here it is lower, signifying a decrease in privacy. We define the set of addresses in which an address can hide, its "hiding set"; the observed addresses outside the set are the ones for which we are able to demonstrate to be unrelated to said address. The privacy of each user is given by the fact that no one should be able to infer anything about addresses' related/unrelated addresses; for Bitcoin we see this is not the case.

All the previous research we know of has been focused on finding related addresses; the objective was to form as many relations as possible between them and to group addresses into big entities, which could then in some cases be identified and linked to real-world users. With this work we want to utilize this knowledge about related addresses, but merge it together with something new: information about unrelated addresses. Thanks to this new view on the links between all bitcoin pseudonyms we make it possible to estimate the hiding set sizes for different addresses in multiple situations.

In addition to simply collecting and reporting available information about Bitcoin and its security and privacy features, our work utilizes various concepts taken from the previously mentioned publications as well as new ideas about how to obtain new insights on the privacy level of the system's users.

We propose a completely automatic analysis system which is capable of combining results coming from multiple heuristics, generating information about the relations between different Bitcoin addresses. In addition to the already known methods to obtain related addresses from publicly available data, we exploit newly found sources of information to also be able to say when addresses are unrelated to each other. We describe a new way of obtaining data about transactions and their association with real-world entities, through a timing analysis of messages received from thousands of peers in the network. By associating transfers of money with IPs, a new series of heuristics is presented, capable of bringing additional information about both related and unrelated bitcoin addresses, and allowing for a more accurate and in-depth analysis of the available data. The association of transactions to online nodes could be used in the future for focused attacks on users and could lead to additional privacy risks.

Multiple experiments are run on different data and periods of time; we show the results of our analysis software, given as sizes of the hiding set for sampled addresses. We use the hiding set, the group of addresses in which the sampled address can hide as we don't know if it's unrelated to them, as measure of privacy (or better, pseudonymity) of the Bitcoin system. We show how, given a certain analysed period where new bitcoin addresses are observed, users have in many cases reduced privacy, due to many other addresses being demonstrated to be unrelated to the observed addresses.

1.3 Structure

The thesis begins with a small theoretical introduction to some concepts of economics and cryptography in chapter **2**. Chapter **3** contains a detailed description of the Bitcoin system, its components, features and provided privacy/security will be discussed. Chapter **4** instead presents alternative cryptocurrencies that extend Bitcoin or were created from it.

The document continues in chapter **5** with a description of the infrastructure, hardware and software used to perform analyses on the Bitcoin network. Chapter **6** takes a look into our theories, why it is possible to associate transactions to IP addresses, which information we use and how everything is combined together to form the results; the latter are presented with the relative performed experiments in chapter **7**.

The thesis ends with a conclusion in chapter **8**, looking at proposals for improvements of the Bitcoin protocol regarding privacy, additional possible work that can be done on the subject, and the future of Bitcoin.

Chapter 2

Theoretical Background

2.1 Economics

What is considered to be money? “Money is any object or record that is generally accepted as payment for goods and services and repayment of debts in a given socio-economic context or country.” **[1]**. Originally money was created as representation of a tangible resource for which it could be exchanged, it was made of a precious or rare material that gave it intrinsic value; in order to make more money, one would have to find and use more of the said material. Nowadays however most, if not all, of the money in the world is so called “fiat currency”, meaning that its value is not given by the material with which it is made, but instead given by a (central) authority, which decides how much a unit of the currency is worth.

Digital currency is a form of money which is created, stored and used exclusively electronically; this is usually done via computer networks, the Internet and digital storage mediums. A big part of modern money is stored and used digitally: the digital representation of real money used by banks all around the world. Cryptocurrencies (like Bitcoin) are a subset of digital currency; they rely on cryptography to implement a secure, peer-to-peer, decentralized money exchange system **[Ger13]**. Digital money can be exchanged for real physical goods, services, or money by the people or organizations that support this type of transactions.

To be considered money **[2]**, an object must have these basic functions:

- **Medium of exchange** – It can be used as intermediary in a trade of goods, to avoid the inconveniences of a barter system.
- **Store of value** – It can be saved, stored and retrieved at a later time, with predictable future usefulness depending on its value.
- **Unit of account** – It can be used as a (standard) monetary unit to measure the cost or value of goods, assets or services.

In addition to that, money should also have the following characteristics:

- **Acceptability** – It is widely accepted as a method of payment.
- **Divisibility** – It can easily be divided into smaller denominations.
- **Durability** – It must be durable and not break or ruin during use.
- **Limited supply** – There must be a limited supply of it.
- **Portability** – It must be easy to carry around and use for transactions.
- **Uniformity** – All versions of a denomination have the same value.

Cryptocurrencies possess all the functions and characteristics specified before;[1] with a noticeable advantage over physical currencies in the divisibility, durability and portability areas. Bitcoin, as the main representative of cryptocurrency, can thus for all intents and purposes be considered money. This type of money cannot be regulated **[3]**, improves on the current credit card payment system for privates **[4]**, and offers solid advantages that could make it the next step to the future for the world economy **[5] [6] [Gri12]**.

2.2 Cryptography

Cryptography, as the name implies, is one of the fundamental building blocks of cryptocurrencies. More precisely, cryptocurrencies try to incorporate principles and constructs of cryptography in order to provide the seemingly impossible service of worldwide decentralized money creation, storage and exchange. Bitcoin uses four main ideas/algorithms/functions taken from the cryptography world, which are resumed in the following sections in order to better understand the internals of the system.

2.2.1 Hash

A hash is the value resulting from the application of a hashing function to an input; this special family of functions is used to produce a fixed-length output that depends from the given data of arbitrary length. In the context of cryptography we need to use (cryptographic) hashing functions that, in addition to being easy to compute, also respect a certain set of properties:

- **Pre-image resistance** – Given only a hash, it is very hard to find any input data that would produce said hash (one-way function).
- **Second pre-image resistance** – Given an input, it is very hard to find a another different input which, once given to the hashing function, would result in the same hash result as the first one.

[1] The only debatable exception could be "acceptability": more and more entities accept this type of currency (especially Bitcoin), however it is not of course accepted in all places and for all purposes, since world-wide acceptance is not (yet) possible.

- **Collision resistance** – It is very hard to find two different inputs which, once given to the hashing function, would produce the same hash.

Hashes are used in Merkle trees (**2.2.2**) and digital signatures (**2.2.3**), in addition to this, Bitcoin also uses them in the vast majority of transactions to refer to destination addresses (**3.1.1**). Due to the previous three properties, a cryptographic hashing function thus also intrinsically has the characteristic that even the slightest change in the input would result in a completely different hash result; this is used in Bitcoin's Proof of Work (**3.1.4**).

The specific hash functions used by Bitcoin are *SHA-256* **[NIS01]** for most of the functions in the whole system, and *RIPEMD-160* **[Dob96]** (in combination with the previous) specifically for the encoding of output addresses.

2.2.2 Merkle Tree

Merkle trees **[Mer79]** (also called hash trees), are special tree structures used to efficiently store and verify hashes of objects. Each node is a hash of its children nodes, the data to be inserted (and in future checked) is passed to a hash function and its hashes are then appended as leaves in the tree, the parent nodes are then recursively updated by re-hashing. Thanks to this structure it is possible to verify the presence of a certain object in the tree by calculating its hash and comparing it only to a small part of the node hashes: the logarithm of the total node in the tree; it is also possible to only verify parts belonging to only a branch of the structure.

Merkle trees are used in Bitcoin for the storage and verification of the presence of transactions inside blocks (**3.1.4**) in the blockchain.

2.2.3 Digital Signature

A digital signature is a mathematical scheme that describes how to sign and verify arbitrary data through asymmetric cryptography,[2] in order to provide authentication, integrity, and/or non-repudiation.

A digital signature system consists of key generation, signing, and verification algorithms. In the first, through different mathematical properties depending on the scheme, it is possible to generate a pair of related keys, private and public, where the first can be used for signing, and the second can be used to verify the signature; the private key must be kept secret while the public one can be distributed freely. In the second algorithm input data to be signed is provided, this is usually hashed and then, together with the private key, used to generate a signature for it, which simply consists in a sequence of bits. With the third algorithm, by having as input the data that is

[2] In asymmetric cryptography, two different (related) keys are used to encrypt/decrypt or sign/verify information, allowing for advanced protocols and situations.

supposed to be signed and the signature, it is possible with the public key to verify the correctness of the signature and thus authenticate the information.

Bitcoin uses one type of digital signature algorithm, defined in the next section, for creating addresses (**3.1.1**) and signing transactions (**3.1.3**).

2.2.4 Elliptic Curve DSA

Elliptic Curve DSA (*ECDSA*) **[Joh01]**, is a variant of the Digital Signature Algorithm (*DSA*), described in the Digital Signature Standard **[FIP13]**. The main difference between the two is that *DSA* exploits the discrete logarithm problem **[7]**, while *ECDSA* utilizes properties of elliptic curves over finite fields **[8]** and thus exploits the elliptic curve discrete logarithm problem **[9]**. The resulting distinction between the two is that, at the same "level of security", *ECDSA* is slightly slower, but requires much smaller keys.

Bitcoin addresses (**3.1.1**) are actually public *ECDSA* keys; the properties of authenticity, integrity and non-repudiation given by signatures are used to ensure the correctness of transactions (**3.1.3**).

Chapter 3

Bitcoin

Bitcoin is a digital, decentralized, peer-to-peer cryptocurrency, described first in a paper published in 2008 **[Nak08]** under the pseudonym "Satoshi Nakamoto", the real person or group of people behind it is, as of now, still unknown.[1] At the time of writing, the currency has been officially scrutinized by several dozen of the biggest country governments around the world and has generally been considered as a legal and accepted money exchange method; it's being used by thousands of small to big merchants **[10]** and is being exchanged by multiple organizations and companies from/to many different real currencies **[11]**.

Bitcoin has been introduced as an open-source software that allows people utilizing the system to create, store and transfer virtual money online between entities in a secure way. The code base is now supported and constantly updated and further developed by the community **[12]**; the contributors and users of the system are mostly concentrated in the official Bitcoin forums **[13]**, wiki **[14]**, Stack Exchange Q&A **[15]** and IRC Channel **[16]**.

The cryptocurrency uses as basic units *bitcoins* (lowercase), often abbreviated as *BTC* or ₿, while the smallest (arbitrarily chosen) denomination is called *satoshi*: one satoshi is 10^{-8} bitcoins, or one bitcoin can be seen as 100 million satoshis. Bitcoin uses several cryptographic constructs to ensure the safety of one's money, irreversibility and verifiability of transactions, and regulated currency creation in a decentralized heterogeneous network; it has to be noted that the system does not include any encryption mechanism: all transactions, with their address pseudonyms, are public and can be observed by everyone **[17] [18] [19]**.

[1] It is interesting to note that it is believed this person (or group) possesses a very big amount of bitcoins (roughly one million), due to their initial, almost exclusive, involvement.

The latest statistics show how the number of publicly known and accessible users (*nodes*) active in the network at any point in time is around 10′000 **[20]**, with many more (behaving differently, only actively connecting to others when needed) estimated to exist.[2] There are about 65′000 transactions being executed per day and the current bitcoin exchange rate of 1 bitcoin for 625$ is accompanied by a daily estimated transaction volume of around 200′000 bitcoins, or 125 million dollars **[18] [19] [21]**. The Bitcoin market capitalization is at around 13 million bitcoins, or 7 billion dollars **[22]**.

In the following sections, the different parts and ideas of Bitcoin will be described (**3.1**) and put together (**3.2**) to demonstrate how the whole system works. *Simplified Payment Verification* Bitcoin clients are explained (**3.3**); aspects of security (**3.4**), anonymity and privacy (**3.5**), plus other possible alternative usages of the deployed network and data will be explored (**3.6**).

3.1 Components

3.1.1 Address

A Bitcoin *address* is a pseudonym identifier that users utilize as source and/or destination of bitcoins inside *transactions*; everyone can, offline and by themselves, create and keep as many addresses as wanted, with no initial nor upkeep cost.[3] Addresses have an always updated balance, publicly known and stored in a world-shared unchangeable log called *blockchain* (**3.1.4**). Bitcoins are not actually stored or sent, they are assigned to certain addresses depending on transaction data (**3.1.3**), that says which addresses just lost or gained a certain amount of bitcoins.

Addresses are derived from *ECDSA* key pairs (**2.2.4**): the private key is not disclosed, but kept in order to sign transactions utilizing the address as input, while the public key is used, in its original or hashed form, as the address in respectively the input or output of a transaction.

Frequently, to display an address in human readable form to receive donations or payments, the bytes composing the address are encoded using the custom *Base58Check* transformation. This encoding takes the raw hash of the public key data, the network for which the address is used (**3.1.5**) and a calculated checksum; it then converts everything to a string by assigning a lowercase letter, uppercase letter, or number, with the exclusion of the 'O', '0', 'I', 'l' characters to avoid confusion due to similar looking strings. An example would be: `1LoreWx4pwKFpRiqf1HMSMyXL2EhYwz6hS`.

[2] Custom clients or any software on mobile hardware can behave this way.

[3] Addresses are created via a "simple" mathematical operation that can quickly be performed on any hardware/software combination, and don't require any form of maintenance.

3.1.2 Wallet

The user keeps and creates all the owned addresses in a structure called a *wallet*, which can be a local or web application responsible for their management, association with private keys and use in transactions (**3.1.3**).

The wallet sums all the owned addresses' balance to know the current total owned bitcoins,[4] decides which addresses to combine and use as inputs in a transaction to reach the wanted output, and is responsible for the creation of *change addresses*. It displays data related to the user's owned assets in a way that is easy to understand and interact with.

Change addresses are new addresses often created when doing a bitcoin transfer: since when doing a transaction all of the balances present at that time in the inputs must be used, when the sum of balances is greater than the money needed for the operation, a new address is created and used as additional output, to transfer the remaining sum back to the sender.

Wallets can be seen as a weak link in the Bitcoin system: bitcoins can be stolen by discovering the private keys associated with addresses, users often don't fully understand the need to protect wallet or address data and are frequently at risk. The addresses and relative private keys are often the target of malware which tries to forcefully take bitcoins (**3.4**); while online wallet services require absolute trust in the remote entity, which can't offer any guarantee about what will be done with the provided information.

Wallet programs are already available in different formats, with additional features, supported platforms and security options; there is also a proposal and development of a distributed peer-to-peer "payment network", meant to replace wallets with an easier to use system **[Sye11]**.

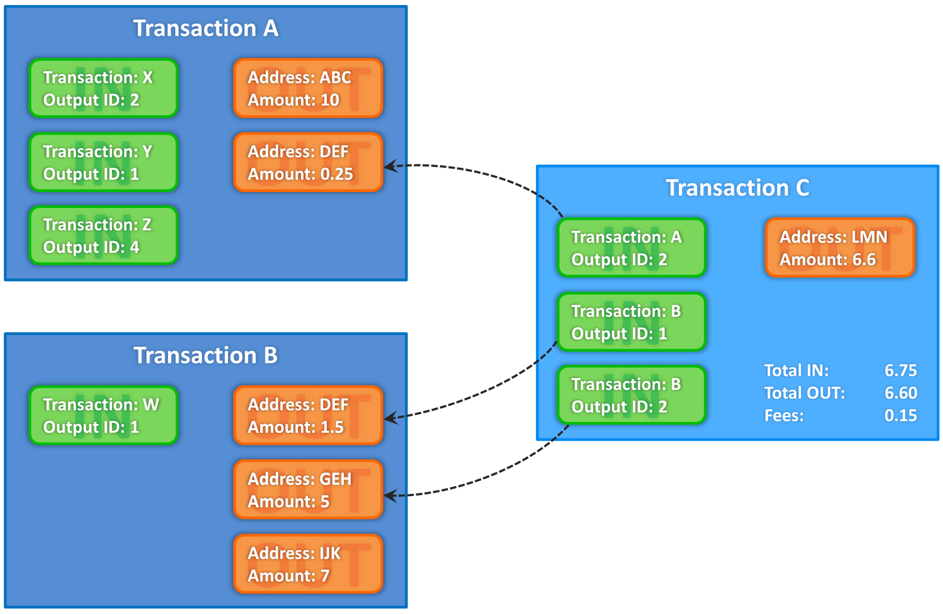
3.1.3 Transaction

A transaction is a transfer of money (bitcoins) from a set of input addresses to a set of output addresses; transactions are publicly announced, broadcasted and relayed through the Bitcoin network to be then included into *blocks* to form a log history (**3.1.4**). The balance of the inputs is completely cleared, while for each of the outputs it is specified how much of the total amount is to be transferred; what remains from the total available sum after having removed all the output values is the *transaction fee*, which is given as incentive to *miners* to include the transaction in the block chain (see the next section for more details about the block inclusion procedure).

[4] By looking at all the publicly known history of transactions related to them.

The input part of the transaction is a series of entries of variable length (at least one): each entry is made of a reference to a previous transaction through a hash, the index of one output, and the public key form of the address together with a signature made with the related private key. What this means is that, for each input address we use, we refer to the balance contained in it when it was used as output in a previous transaction,[5] and provide a signature over the currently made transaction with the address' private key to prove the fact that we own the address. A transaction is called an "orphan" when it mentions in at least one input a previous transaction which cannot be found in the blockchain, it is then kept in a special waiting queue by the clients receiving it, awaiting the missing referred transaction(s).

The output section consists of another series of entries: each one is made of a numeric value, telling how many satoshis are to be transferred to that output, and an address, which is usually given as an hash of the public *ECDSA* key (known by the sender through any channel the receiving entity decided to use to advertise the address). As previously explained, the output list commonly contains an address forged specifically to send the "change" money back to the source user of the transaction, as a consequence of the necessary complete usage of the input addresses' balance and sum needed to be sent to the other outputs.



Transaction A
Transaction: X Output ID: 2
Transaction: Y Output ID: 1
Transaction: Z Output ID: 4
Address: ABC Amount: 10
Address: DEF Amount: 0.25
Transaction B
Transaction: W Output ID: 1
Address: DEF Amount: 1.5
Address: GEH Amount: 5
Address: IJK Amount: 7
Transaction C
Transaction: A Output ID: 2
Transaction: B Output ID: 1
Transaction: B Output ID: 2
Address: LMN Amount: 6.6
Total IN: 6.75
Total OUT: 6.60
Fees: 0.15

Figure 3.1: Simplified view of the relations between Bitcoin transactions.

[5] The same input address can thus appear multiple times (in the same or multiple transactions), each time utilizing the bitcoins gained in a different previous transaction.

Figure **3.1** shows a simple diagram of what the inputs and outputs for a transaction *C* look like. It can be observed how the inputs refer to previous transactions and are used to collect money to be sent to selected outputs. Not displayed here are the signatures provided by whoever created the transaction in order to make it valid: for each input, a signature on data about the previous and the current transaction must be provided through the private key of the address mentioned in the input.

Bitcoins are generated and assigned to selected output addresses through special *coinbase transactions* (explained in more detail in the next section), which contain a single specially crafted input that doesn't refer to any previous transaction and transfers a fixed amount of bitcoins.

It has to be noted that in the previous explanation of the transaction structure a simplification has been made: the vast majority of transactions uses normal addresses as outputs and signatures for input verification as specified; Bitcoin however defines a whole custom scripting language *"Script"* that allows for more complex conditions for input and output usage **[23]**.

3.1.4 Block

Blocks are used to store transactions plus additional metadata, every block stores new transactions not stored in any previous block and together they work as a public log or ledger of bitcoin transfers from address to address. Each block holds a reference to the hash of the previous block, up to the first hard-coded genesis block, thus forming what is called a block chain: a (generally) unchangeable common transaction history agreed upon by the Bitcoin system participants.

The most important data stored in the block is the full transaction list and the block header, which in turn is composed of:

- Version number
- Hash of the previous block
- Root of the Merkle tree of transactions
- Timestamp
- Nonce value
- Target value (difficulty)

In figure **3.2** we can see a dimple diagram of the main components of blocks, what data they contain and how they are linked to each other. Note that blocks must contain a minimum of one transaction (coinbase).

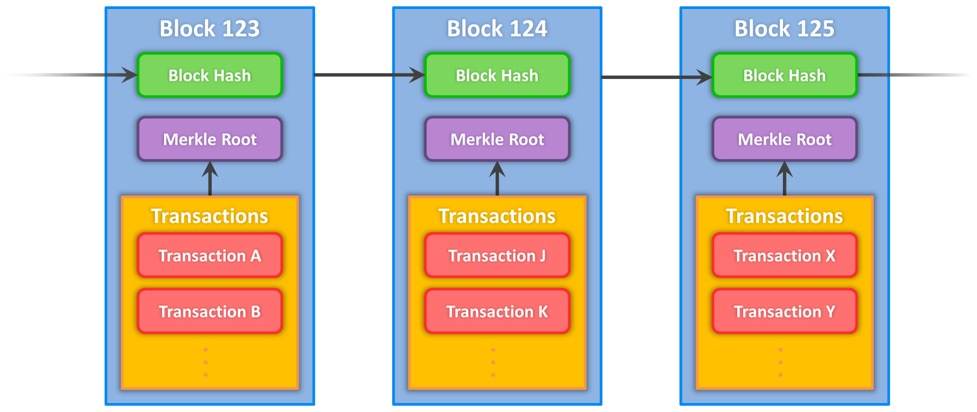


Figure 3.2: Simplified view of the data that can be found inside blocks.

The main block chain, the one that every node in the Bitcoin network should know, refer and append to, is defined to be the chain of blocks with the highest total sum of difficulty, and is also called the "longest". Blocks received by clients in the Bitcoin network that miss at least a predecessor that links them to the current blockchain are called "orphan" and are ignored until the missing blocks are elaborated. It is possible that at certain points in time the current chain branches ("forks") due to simultaneous discoveries, network topology and link delays. This situation however typically resolves by itself once a new block is found and appended to one of the branches, making it the new main and official chain, which merges any transaction only included in the other old branch. An analysis of the forking problem and possible improvements can be seen in **[Dec13]**.

Miners work on the latest block of the main chain, immediately broadcasting to the entire Bitcoin network any new block they found that is a correct solution to the *Proof of Work* (*PoW*), making so that it gets appended, increasing the length of the chain and providing a proof of validity for the latest included transactions and the bitcoin transfers mentioned in them.[6]

Figure **3.3** presents a view of a possible section of the blockchain. The main chain is shown in green, small forks like the one displayed in violet can happen relatively frequently, while extended forks like the one shown in orange are much rarer. The main chain is the line of blocks with the highest total difficulty and its last block is referred to by miners who are working on new blocks that will extend the blockchain.

[6] The validity or "confirmation amount" of transactions increase as more and more blocks are appended to the chain following the logging of the transactions.

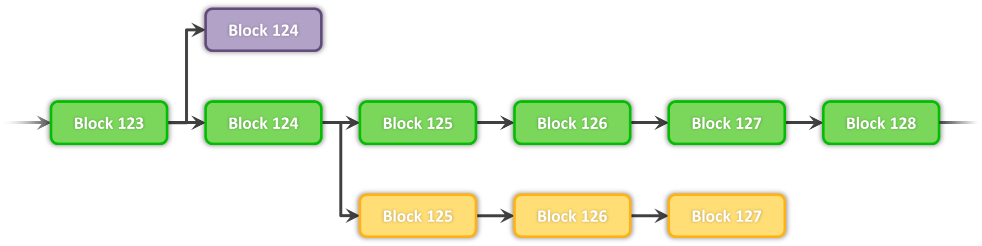


Figure 3.3: Possible section of the blockchain, displaying relations between blocks and forks.

Mining

Starting from the initial genesis block, new blocks can be found and added to the current chain by anyone that participates in the so called *mining* process: the discovery of new blocks that contain a correct solution to the Proof of Work system. The Proof of Work function is meant to ensure that the entities proposing new blocks have done some work in order to be able to propose it: they have found a solution, and thus spent computing power and time towards the current PoW problem which has a certain difficulty. It is hard to find a correct answer to the current problem, but very easy and fast to verify the correctness of a solution.

The difficulty of the problem is represented by the *Target* value stored in the block, it is adjusted dynamically so that the discovery of new blocks remains as constant as possible in time despite the varying number of entities and machine speeds; the objective for Bitcoin is the creation of one new block every ten minutes on average.

Miners receive a reward for their computational effort in case of success and the discovery of a new valid block. This block must contain any new transaction that the entity chooses to include; as a reward for their work, any fee contained in the included transactions, plus a fixed initial amount of bitcoins, is given to a set of output address chosen by the block creator in the custom created coinbase transaction. This special single transaction can only be written by the author of the new block, it creates new bitcoins by specifying a single all-zero input address which sends a fixed amount of satoshis to the listed outputs.

The fixed reward for the creation of new block is set to slowly decline over time for Bitcoin, until zero is reached. However after that happens blocks will keep being mined and appended to the block chain indefinitely in order to provide transaction history and verification, the reward for the miners will at that point only be composed by fees given by the transactions included in the block.

The Bitcoin protocol is incentive-compatible: it encourages miners to follow the commonly defined rules in order to gain the greatest possible benefit for their efforts; the system is also protected against a minority of colluding entities. Miners should announce all their newly discovered blocks immediately to everyone and only work on the main block chain, they should not work in private on their own "private fork" of the chain or the system could become compromised if the computing power is high enough, they should also relay all the received transactions.

While Bitcoin is currently considered safe from a certain amount of selfish entities, there are discussions on the information propagation problem in the network **[Bab12]**, analyses are also being conducted about the resistance of the system and possible improvements and modifications for facing this type of menaces **[Eya13] [Kro13]**.

Proof of Work

The Bitcoin Proof of Work system is based on hashing "randomness": the property of cryptographic hashes for which it is difficult (impossible) to predict what output an input will produce until it is tried, related to the fact that a small change in the input greatly affects the output. The challenge of the PoW specifies that the *SHA-256* hash (**2.2.1**) of the newly created block, calculated on the block header data, must be under a certain value, depending on the difficulty; this can also be seen as producing a hash value beginning with a certain amount of zeros, defined by the *Target* value stored in the block. The difficulty is a globally known value, set to increase over time, calculated by the Bitcoin client.

Miners continuously try to create a block with a new different hash by adding more and more transactions (or modifying the coinbase transaction), thus modifying the *Merkle tree root* (**2.2.2**), updating the *Timestamp*, and trying different *Nonce* values. As soon as a new valid block is found, it is broadcasted through the Bitcoin network; everyone, once the validity is checked, relays and appends it to their current block chain, all the miners then start working to add another block after the newly found one. The probability of finding a new correct solution is directly proportional to the utilized computational resources, due to the system being a bruteforce random generation and check of hashes.[7]

[7] There is no known algorithm for predicting the results of a (secure) hash, quicker than executing the hash itself. Faster machines thus simply can try more random hashes in the same amount of time, thus increasing the probability of finding a valid one for the PoW.

Proof of Work ensures that the "majority" of the system, that basically decides which is the correct public log of transaction (block chain), is given by the majority of computing power. An evil entity would need to possess more computing power than the sum of all the honest entities combined together in order to be able to control the block chain and modify previously stored data. Transactions which are added to a new, valid block in the chain are considered "confirmed", or with 1 confirmation; each block produced and appended afterwards to the chain counts as an additional confirmation for the transaction. As more blocks are added, it gets exponentially more difficult for an adversary with less computational power than the whole honest part of miners taken together, to modify previously logged transactions.

There are some doubts regarding the general efficiency of Bitcoin's Proof of Work system and how well it could work in different scenarios **[Bec13]**; the main recognized problems are the waste of energy, the insecurity due to entities with very high computing power (intrinsic to a PoW system) and the rise of specialized hardware (*FPGA*, *ASIC*) for fast hash calculation due to the *SHA-256* algorithm used. There is also a future possible onset of a *Tragedy of the Commons*: a market failure scenario where something is consumed more than wanted or produced less than wanted; in the Bitcoin world this could be due to the ever-decreasing rewards for block mining.

Proposed solutions, to be used by themselves or in conjunction with Bitcoin's Proof of Work, are *Proof of Stake* **[24]**, where entities are the "majority" when they own the bigger part of the total amount of money in the system; and *Proof of Burn* **[25]**, where miners show proof of having virtually consumed currency as a way to demonstrate work done. There are also alternatives to the specific hashing function used by Bitcoin, built to prevent entities from developing specialized hardware or exploit GPUs for faster parallel processing, an example of such alternative functions is *SCRYPT* **[Per12]**, a key derivation function meant to be intentionally slow and very difficult and costly to execute on custom or parallel hardware, due to the extremely high memory requirements. Practical uses of these alternative methods will be shown in chapter **4**, where alternative cryptocurrencies branched from Bitcoin are presented and analysed.

3.1.5 Network

The Bitcoin system works thanks to clients communicating between each other in a decentralized (or distributed) Peer-to-Peer network **[26]**; the clients of the online system are called nodes. Any device running a Bitcoin client application is automatically connected to the network, where it sends/receives messages via TCP to/from a limited number of connected peers, following the defined protocol rules **[27]** and specifications **[28]**.[8]

[8] Nothing prevents however running a custom clients that only follows some of the rules and behaves differently from the other participants depending on the situation.

Bitcoin's main (and most known) network is called *mainnet*, it is in opposition to *testnet*, a parallel network which utilizes virtually the same protocol, but has a different block chain and utilizes different default ports for communication. Testnet bitcoins have no value (or at least, they are not intended as a currency): they can very easily be created and exchanged, available to whoever needs them. The network is used as a kind of public testing platform, where everyone can experiment with new or modified clients without affecting the real Bitcoin system.

Nodes connected to the network have several different ways to discover and connect to new peers **[29]**, these methods can be used by the client when it connects for the first time, or regularly to maintain a list of possible nodes to connect to. Besides manually provided IP addresses, a node can get an initial list of nodes to connect to from special DNS requests to a series of hostnames written in the code of the client; as last resource it is always also possible to reach a limited group of fixed nodes hardcoded in the software. Once communicating with several peers, the client keeps an updated list of reachable addresses, which are then exchanged through various messages in the whole network in order for everyone to discover new clients.

A node can react differently to incoming messages and send several outgoing messages depending on the running client software. As the Bitcoin network is public and the minimal protocol requirements are very simple, many custom client programs exist that have different purposes in the system. Nodes differ in how they connect and react to other peers: depending on the local network and on the software, a node can be set to only actively create new connections, to only connect when needed, to (not) relay/broadcast certain messages, and/or to only communicate with a restricted or selected amount of other nodes during their online time.

A so called "full" node is a client connected to the network which is particularly useful for all the connected peers, the usefulness is given by the fact that such a node accepts incoming connections, supports many connected peers, tries to connect to many nodes, and provides updated verified transactions and blocks, which are always punctually relayed through the network. Other nodes could simply passively listen to the communications, or only relay some information, depending on the running software.

Any client connected to the network can be a miner (**3.1.4**), even though usually the computers performing the expensive mining operations are not directly connected in order to save computing power, and instead use a proxy node to interface with the net. It is very important for a miner to be very well connected to many peers, in order to receive new transactions to include in the block and, more importantly, to quickly receive and transmit newly mined blocks.[9]

Most miners organize in groups and form what are called *mining pools*: in such pools the work of finding a new block is split between all the participating users; when a new block is found by the group, its reward is split, usually depending on the amount of work contributed in the PoW tries for the block. Pools usually posses one or more nodes responsible for the management of the workload and the receipt of updated data.

3.1.6 Communication Protocol

Each peer in the Bitcoin network communicates with each other through a specific restricted set of messages. When a new connection is created between two nodes, the protocol dictates that the actively connecting peer sends a *version* message, specifying identity details and various supported versions of protocols, this is then followed by an acknowledgement by the receiving party; the procedure is then repeated in reverse. After this initial handshake, any message can be exchanged between the nodes, until one of them disconnects, a timeout is reached in the connection, or a peer gets temporarily banned by the other for misbehaving (for instance due to a denial of service attack or too many malformed messages).

We will now describe the most important message types that can be sent and received. For more details, the official wiki contains a thorough definition of all messages and their included data in the Bitcoin protocol **[28]**.

- **version:** Advertises client name and version, list of supported features, protocol version and additional information. It is sent from two communicating nodes to ensure they exchange comprehensible messages.
- **verack:** Signals that a peer received and accepted the other node's just received *version* message, sent as answer right after it.
- **getaddr:** Requests a peer for any recently active node it knows of, used for network discovery, to keep a list of known possible peers and connect to new entities now or in the future.
- **addr:** Lists a certain amount of known active peers which can then be accessed by other nodes in the network. Sent as answer to a *getaddr* message or at any moment to random connected nodes, in order to ensure an update knowledge of reachable nodes to the entire network.

[9] Working on a blockchain which is not updated and not knowing which block is the current latest one results in wasted work due to the production of unacceptable solutions.

- **inv:** Standing for "inventory", this message is sent to all connected peers whenever the node creates, or receives and validates a new transaction or block. It contains a hash of the advertised new object, which the receiving peers then decide to request if needed/wanted.
- **getdata:** Requests for an entire object (transaction or block) to be transmitted, by providing a hash to identify it. Usually sent right after receiving an *inv* message for something the node didn't already know about, in order to receive and process it.
- **tx:** Message representing an entire transaction object, sent as answer for a *getdata* request, asking for the identifying hash.
- **block:** Message representing an entire block object, sent as answer for a *getdata* request, asking for the identifying hash.
- **ping/pong:** A *ping* message contains a nonce, when sent to a peer it will respond with a corresponding *pong* message. Used automatically or manually to check if a connection is still valid and active.

3.1.7 Client

The software used to connect to the Bitcoin network and interact with other peers can be called "Standard Client", "Satoshi Client" or simply "Bitcoin"; recently, to avoid confusion, it has been re-branded to "Bitcoin Core".[10]

The original program was released together with the initial paper by the creator Satoshi under an open source license,[11] the community took the project and is now continuously reviewing and updating it. Users can download pre-compiled binaries for Windows, Linux, or Mac OS; the source code is freely available on the official Github page of the Bitcoin project **[12]**. As the network and the reference code are easily accessible, multiple branches of the Bitcoin Core project are available to use, with various modifications and added functionalities; versions of the software in different programming languages can also be found.

The original client is written in `C++` and is composed of two main parts. The base is a standalone server, connecting to the network and accepting command-line commands or `JSON-RPC` **[30]** calls to execute actions and actively send user-requested messages to the connected peers. The software also contains an optional standard wallet, built with `Qt` **[31]**, interacting with the server and providing an easy to use graphical interface.

[10] Bitcoin release notes: **https://bitcoin.org/en/release/v0.9.0**.
[11] MIT License: **http://opensource.org/licenses/MIT**.

3.2 How Everything Works Together

We will now have a look and summarize in a simplified and concise way how all the previously explained components work together in order to provide all the Bitcoin cryptocurrency services.

Everyone can generate new Bitcoin addresses and can use as many as wanted, bitcoins actually only exist assigned as balance to an address. Money is transferred from one address to the other through transactions: an operation moving bitcoins from a series of inputs to a series of output addresses (plus optional fees). A transaction can be made only by the person possessing the input addresses, only they can sign the transaction, but everyone can then verify its correctness. After the transfer of bitcoins, the input addresses are emptied of all their bitcoins, which are then assigned to the chosen output addresses. Every transaction input specifies an older transaction's output (the new input and the old output addresses are the same) to indicate the amount of bitcoins that it contains and will be used for the transaction.

When a new transaction is generated, it is broadcasted publicly throughout the Bitcoin network, each node verifies the validity of the transaction by looking at the source of the bitcoins used as inputs before relaying or working on it. Miners are special nodes that work using computational power to find a solution to a difficult Proof of Work, trying to include any transaction they receive in order to then gain as a result its fees. They continuously try to generate a new block, containing all new transactions since the last block, to be appended to the existing block chain; if they succeed, they gain a fixed amount of bitcoins plus any fee contained in the included transaction. The blockchain acts as a public ledger of all transactions in history; as the blockchain is unique for all the Bitcoin participants and requires work for each block to be added, altering previous data becomes increasingly hard the more time passes.

Every peer in the network always keeps its block chain updated (for it to be the same as everyone else's), relays its own or others' transactions and blocks to all its neighbours, and verifies the correctness of data before doing so. All the nodes continuously exchange information about other reachable entities and don't relay on any central authority, forming a peer-to-peer network used to exchange bitcoins safely, on the base of the security of the underlying cryptographic primitives, protocols and publicly known information.

3.3 Simplified Payment Verification

Every node in the network should posses the full block chain; this is needed for transaction verification, required whenever information coming from a peer must be relayed to other peers, and to also to know if bitcoins sent or received in own transactions have been accepted and recorded by the peers.

The block chain naturally grows bigger and bigger with time; as it must contain all the transactions in the history of Bitcoin, it also is of considerable size.[12] This fact, united with the growing number of transactions being executed every minute, would make it very hard (if not simply impossible) for devices with limited storage and/or network capabilities to run the bitcoin client and thus utilize the cryptocurrency.

Simplified Payment Verification (*SPV*), makes it possible to run what can be seen as a ”partial” Bitcoin node: a client using it doesn't need to download and posses the entire block chain, and in exchange is slightly less secure and cannot help its peers by relaying blocks or transactions.[13]

With SPV the client maintains exclusively the headers of the blocks in the blockchain, and announces to its peers to only relay to it “relevant” transactions, filtered by a special data structure transmitted at the beginning of the communication. The headers only allow the node running the software to verify that they indeed constitute a correct block chain (by verifying difficulty length and chaining of hashes), and if a certain transaction is included in a block (if provided together with a Merkle tree branch, linking it to the block header). Simplified Payment Verification also saves more space by only keeping a limited amount of block headers: the most recent ones known, meaning the top part of the blockchain.

The advertisement of “relevant” transactions that a SPV client wants to receive happens through *Bloom Filters* **[32]**, probabilistic space-efficient data structures which test the presence of an element in a set with possible false-positives but no false-negatives. The client sends to its peers a certain bloom filter, containing the transactions it is interested in, the peers will then check every transaction about to be relayed with the filter, transmitting it only if it's contained in the filter. A node running Simplified Payment Verification can make a compromise between the rate of false-positives (uninteresting transactions received) and the level of privacy (given by the implicitly advertised “own” transactions due to the information included in the filter).

This way of participating in the Bitcoin system can be seen as somewhat selfish, as it doesn't help neighbouring nodes and “steals” resources; it is however the only way to allow resource-limited devices to connect and use Bitcoin. SPV clients are less secure, as they can't completely verify a transaction validity by looking at the origin of its inputs; a transaction is deemed valid depending on the sheer quantity of blocks in the chain after the block containing it. Simplified Payment Verification depends on good, reliable connected peers to ensure the correctness of received data.

[12] At the time of writing, the blockchain has a size of around 18GB and is steadily growing (Chart of the total size: **https://blockchain.info/charts/blocks-size**).

[13] A SPV node only listens to any block/transaction received but doesn't relay it.

3.4 Security and Attacks

Bitcoin is currently generally considered secure, its underlying algorithms for hashes, `SHA-256` and `RIPEMD-160` (**2.2.1**), and digital signatures, `ECDSA` (**2.2.3**), have no known practical attack available and the research community approve their usage for Bitcoin's purposes. In the protocols used there is no apparent major flaw that has been found, and as encryption is never actually used in Bitcoin (all the data, except private keys, is public), there are no other related problems to consider.

Bitcoin's "51% attack" could be considered its main weakness; it stems from the protocol's absolute decentralization and dependence on a majority calculated by raw given processing power. It's the majority that in the long run decides what gets accepted and included in the blockchain; if someone (which could of course also be a group of people) has the majority of the computing power and has malicious intents, they could craft arbitrary transactions, spend the same bitcoins multiple times or prevent other legitimate transfer from ever being accepted. Although this attack is possible and in recent times an abstract entity even obtained (temporarily) the majority of the computing power, its effect would be limited, as thanks to Bitcoin's publicly known data everyone would immediately notice something is wrong; the debate is still on about the reality and effect of this issue **[33]**. Nonetheless, additional schemes like Proof of Stake or Proof of Burn could complement the basic Proof of Work system used in Bitcoin and are meant to provide additional protection against this scenario.

Since the Bitcoin network is composed of pure peer-to-peer connections, a completely new node (or one that was offline for a long time) selects at random to which limited set of peers to connect as explained in section **3.1.5**; all these nodes could be controlled by malicious entities. If a client is connected exclusively to other evil nodes, it could receive any kind of crafted data about blocks and transactions, and its request could be ignored and never relayed. The possibility of this happening however, due to the number of possible nodes, established connections, and limited communication with nodes in the same subnet, is extremely remote.

Any full node in the network is publicly reachable through the Bitcoin protocol and can indiscriminately connect to any client that requests to do so, sending messages following the protocol. A malicious entity could set up a Denial of Service attack, exhausting the local resource of the target node by rapidly sending large amounts of data or specially crafted information that takes unusually long to process, thus preventing the node to communicate with other peers and eventually crash. The Bitcoin client however has many protections in place to protects against such attacks, from data rate limiters to a list of temporarily banned nodes; as of now no efficient DoS attacks have been found or observed in practice.

Network segmentation, resulting in two or more group of nodes communicating only within their separated subnetworks, could bring problems if experienced for a long time due to the common blockchain log structure. While the network is segmented, the different parts would add new blocks and transaction to their "local" blockchain, which grows from the common base known before the network split. When the segmentation ceases to exist, we would have two or more different branches of the blockchain whose information must be merged together in one single chain. If the segmentation lasted long enough, transactions using outputs from old coinbase transaction would have to be recursively discarded, possibly causing issues. Given the great amount of nodes around the globe and the number of connections, this scenario is again highly unlikely to ever happen.

Regarding transactions, the Bitcoin protocol as already explained in the previous sections requires that a transaction is confirmed multiple times by being included in the main blockchain before being accepted as "valid". Since however this procedure takes time (a block is generated on average every 10 minutes), in many situations entities who need to receive money in a fast way can decide to accept a transaction before it is confirmed by any (or only one) block. This opens up the system to a series of attacks known as "double-spending", where an attacker connected to the victim creates two transactions spending the same inputs but giving them to a different output, himself or the victim, depending on the transaction.

With the right conditions the attacker can then convince the target that they received the money, while in reality eventually it will be the transaction giving money back to the attacker the one that will be accepted and stored in the main blockchain. The attack was first described in **[Kar12]**, with more variations being listed in the official wiki **[34]**. Again this is an attack which is quite difficult to perform in practice, basically impossible if the receiver follows the proposed Bitcoin usage recommendations **[Bam13]**.

So far it would seem that the whole system is very secure, so why are there so many news circulating about "lost" bitcoins and big heists all over the world? A list of all the known big stories of Bitcoin criminality can be found on the official forums **[35]** and can look daunting. As with many other technologies however, the truth is that for all (or at least the vast majority) of the presented problems the cause can be found not in the instrument, Bitcoin, but in the people using it.

Many persons utilizing Bitcoin wallets don't realize how they should protect their data, private keys are lost or stolen, and with them the bitcoins the unlock access to. Even if the protocol and system are completely secure, if the users don't keep their private information safe and fail to follow the rules, nothing can be done. As we are talking about bits stored in computers and other devices, another issue followed Bitcoin's success: malware.

Since the cryptocurrency's initial usage diffusion there has been a worrying increase in malicious software meant to steal local credentials, including private Bitcoin keys, and to hijack computational resources to mine and make money without the user knowing **[36]**. Users need to be aware of the technology, its risks and how to be secure before risking losing money.

Another "human issue" is related to trust: too many Bitcoin users wrongfully trust blindly remote (web) entities providing exchange or wallet services, with no guarantees **[Moo13]**. There is no way for a website to ensure their service is legitimate and that its users don't run any risk by not having personal control of their data. Using web services inevitably leads to more security risks: the most secure thing to do is to manage all information related to Bitcoin by yourself, on your own devices.

Perhaps the most widely known news related to Bitcoin security was the failure of "Mt. Gox", at the time the most active and used web exchange, which resulted in bankruptcy and what until today remains the biggest amount of bitcoins ever stolen by unknown entities. There are countless articles on the web about the matter **[37] [38]**; resumed in a few words, it all revolved around inexperience, ignoring information about how Bitcoin works, and "transaction malleability" **[39]**. The wrong usage of data provided by the Bitcoin system, in this case the transaction identifier, known to possibly change for the same transaction, resulted in the catastrophic outcome. Transaction malleability was something already known and not viewed as a security problem, but as a feature of the system; however after what happened, to avoid further issues due to people utilizing data in the wrong way, everything was "patched" in the next Bitcoin version.

Bitcoin shares many features with real money, it can be harder or easier to steal depending on the point of view and knowledge of the user. What is sure however is that, while they might be taken via hacks, malware or user misbehaviour, bitcoins can hardly be made disappear, with everyone knowing about everything that's happening in real time **[40]**.

3.5 Anonymity and Privacy

One of Bitcoin's big advantages over normal currencies is its completely decentralized structure and independence from any kind of external entity, it's a system that works by itself thanks to the collaboration of its users.

A big misconception about the cryptocurrency is that it provides anonymity: being able to make payments no one is able to trace and with no leftover information. On the contrary, Bitcoin doesn't provide this, being all transactions public and known by everyone, with involved addresses and amounts; it provides instead privacy in the form of pseudonyms.

Bitcoin's pseudonyms are the (public) addresses used to indicate who receives or sends money in transactions, each user of the system can create as many addresses as needed and thus has at their disposition a theoretically infinite amount of pseudonyms, thus guaranteeing a certain level of privacy.

Despite the use of pseudonyms it is possible, thanks to the publicly available transaction data, to perform many types of analyses revealing relations between theoretically independent addresses and grouping information to form entities. It has been shown how movements of money can be manually tracked across the blockchain history through different transactions, linking several different transactions and discovering transfer patterns **[Ron13]**. Information coming from external sources like data coming from IPs or web scraping can be combined with the data contained in the blockchain, to discover additional relations between entities involved in transactions and form maps **[Rei11]**. Statistics can be inferred about the amount and type of entities interacting in the network, together with additional information about the usage of addresses **[Obe13]**. By looking at address usage and appearance in transactions over time, data about entities and their relation can be automatically built utilizing precise heuristics **[And12]**, this type of analysis has also been further developed to include additional data sources and obtain more precise results **[Mei13] [Kos14]**.

In order to address the cryptocurrency users' growing need of privacy over time and to counter the many possible analyses on data coming from different sources, various solutions have been proposed.

One improvement towards anonymity comes with the possibility of running Bitcoin nodes through the *Tor* Network **[41]**; doing so doesn't change anything related to the address information that is stored in the blockchain, but it helps avoiding the link of TCP/IP network information to blockchain and network data. Utilizing the anonymity network brings however disadvantages like increased delays and possible network weaknesses due to misbehaviours and limited availability of usable nodes.[14] Nonetheless, this possibility is planned to officially appear in one of the newer versions of Bitcoin's Java library used by most of the SPV clients, BitcoinJ.[15]

There has also been a big increase of *mixing* services for Bitcoin **[42] [43] [44]**, which (theoretically) provide stronger guarantees of untraceable transactions by collecting bitcoins that need to be sent from multiple sources and then executing multiple transactions towards the original recipients. While the ideas and executions work, there's an important problem of trust due to Bitcoin's system itself, as the money initially sent to the mixing service is

[14] Given the limited shared exit nodes used by *Tor*, a single evil entity could quickly get all of them temporarily banned on most of the nodes in the Bitcoin network by misbehaving.
[15] BitcoinJ – Bitcoin Java implementation: **http://bitcoinj.github.io**.

never guaranteed to eventually reach its destination in the case of a malevolent service provider. Using such mixers also means incurring in additional fees for every transfer, as multiple transactions are needed for the service to work; plus longer delays for the transfer of money.

Several services of this kind also advertise similar functionalities in addition to utilizing the previously mentioned *Tor* service to connect to the network and interact with their customers **[45] [46] [47]**, often together with web wallet services. Again here the issue lays in the absolute trust without guarantees that must be put in the providers of these services, which could at any time take any amount of bitcoins passing through their mixes, with *Tor* maybe even more easily than without, as the many scam reports can confirm. All in all, mixing services seem at present too insecure to be used, and with limited advantages for privacy depending on the case **[Mös13]**.

Bitcoin is a cryptocurrency providing strong privacy guarantees, however depending on the wanted features they could be considered insufficient. Its level of pseudonymity (as with security) depends on how it is used: connected peers, performed transactions and creation/usage of addresses; ”Bitcoin is as anonymous as you want it to be” **[48]**. If Bitcoin’s privacy is not sufficient, there are also other cryptocurrencies like *Zerocoin*, which can provide more advanced functionalities depending on the specific needs.

3.6 Alternative Uses

Bitcoin has created through its usage a vast, heterogeneous network of devices communicating with each other throughout the entire world; messages can be exchanged and data is stored permanently in a huge log file (the blockchain) accessible by anyone utilizing the system.

However this newly formed infrastructure is not exclusive to Bitcoin, and can be (positively or negatively) exploited for other purposes. As seen in the next chapter, alternative cryptocurrencies are continuously created, some of them taking advantage of the existing Bitcoin network and protocol, utilizing the same existing functions in different ways or extending the protocol.

By wasting bitcoins sending them to an inexistent output in a transaction (effectively losing bitcoins forever) it’s possible to insert messages, which will be then stored forever in the blockchain for everyone to see. This opens up many possibilities, from innocuous chat-like texts, to advertisements and spam; there are also available websites that provide “existence certifications” and “electronic notary” services **[49] [50]**, essentially storing forever the knowledge of some information at a certain moment in time.

It's also possible to create many different types of applications that set up and maintain a peer-to-peer network parallel and similar to the one used by Bitcoin, from email and instant message clients, to any type of distributed application in the cloud. *Ethereum* **[51]** is probably the best example to demonstrate what is possible to obtain by generating and using decentralized networks like the one used by Bitcoin: Ethereum is a developing platform and programming that allows anyone to create the application they want to run on their network of peers, in a secure way.

Bitcoin can be used for several different purposes, not just for exchanging money; its network, protocol and community has encouraged and allowed people to create new and different applications that can securely exchange information between people around the world without a central authority.

Chapter 4

Alternative Cryptocurrencies

Bitcoin is a completely open source software: everyone can freely take the code, modify it as wanted, and release it as a new program/protocol/currency. Following the increasing popularity and exposure of Bitcoin, dozens of alternative currencies have been created, died, or are currently being exchanged online **[52] [53]**. There are of course also cryptocurrencies based on completely different concepts, unrelated to Bitcoin, but here we will have a look at the ones that stemmed from the original *BTC* code base.

Alternative "coins" are often abbreviated as "altcoins", they use their own network of peers, formed by the people running the relative custom client software, and perform transactions from and to addresses related to (and only valid for) the specific currency. The success of alternative cryptocurrencies varies greatly: many are simply not adopted and ignored, another big part is "in the middle", utilized only by a limited set of users, but some have become truly profitable and widely used, with an increasingly high value and exchange rate with regard to Bitcoin **[54]**. The most successful altcoins are exchanged for real money and other major cryptocurrencies on the biggest online exchanges.

4.1 Selected Altcoins

We will now have a look at some of the most important and interesting alternative cryptocurrencies available, shown in figure **4.1**. An altcoin can differentiate itself from Bitcoin simply just by a different name but unaltered code, simple variable tweaks, or completely different and innovative algorithms, protocols and applications. The presented currencies have in common high usages and market shares as shown in table **4.1**; and present interesting ideas, features and histories that differentiate them from the rest.

Figure 4.1: Altcoins' logos: Namecoin, Litecoin, Mastercoin, Peercoin, Dogecoin, Zerocoin.

Name	Available coins	Coin value	Market cap	Volume (24h)
Bitcoin	12'900'000	561	7'200'000'000	31'300'000
Namecoin	8'870'000	2.53	22'500'000	234'000
Litecoin	28'800'000	11.1	320'000'000	4'050'000
Mastercoin	565'000	26.01	14'700'000	1'830
Peercoin	21'500'000	2.34	50'200'000	195'000
Dogecoin	79'700'000'000	0.000398	31'700'000	727'000

Table 4.1: Selected altcoin market data, economic values in US dollars (*retrieved 25.5.2014*).

4.1.1 Namecoin

Namecoin **[55]** was the first altcoin that has ever been created as fork of the original Bitcoin project. It utilizes the decentralized network infrastructure and currency exchange for a completely new and seemingly unrelated purpose: a distributed decentralized DNS[1] service, providing new `.bit` domains.

Namecoin keeps all the functionalities of Bitcoin intact, coins can be spent and received; the transactions however are also used for the added DNS functionalities, to register, update and transfer domains through new commands available to be used when transferring money. The new service can be also used to store any wanted identity information besides an IP address, like e-mails, public keys, Bitcoin addresses, etc. In order for a client to solve a `.bit` domain name, it must posses an updated Namecoin blockchain, where it can lookup the current relevant updated information. This type of DNS is particularly resilient to attacks, censorship and is independent from any central authority, as information is "approved" by the majority of participants.

Merged mining is also another innovative aspect of Namecoin, adopted also by other altcoins developed after it. A miner can choose to utilize its current hashing power to work on multiple cryptocurrencies (supporting merged mining) at the same time, plus in addition also on Bitcoin. The workload for the miner basically stays the same, hashes are generated at the same

[1] Domain Name System (DNS): translation of address strings (host names and web addresses) to IP addresses, necessary for easy to use networks and web communication.

speed, they can however "submit" their results to many different altcoins, thus increasing their success chances of finding a correct solution to the current Proof of Work problem(s). This results in increased gains for miners, who are encouraged to work on many different cryptocurrencies, and at the same time also in increased security for all involved currencies, as their total hashing power used to keep the block chain updated is increased, thus making it more difficult for attackers to gain the majority of resources.

4.1.2 Litecoin

If Bitcoin can be currently seen as the "gold" of cryptocurrencies, *Litecoin* **[56]** is then the "silver": it is the second virtual currency by volume exchange, being clearly in the lead over the other competitors; its market capitalization is around 29 million litecoins, or 320 million dollars **[22]**.

Litecoin implements two main changes if compared with the original Bitcoin code, which made it the most successful altcoin as of yet. First, a different block creation speed: the difficulty is chosen so to have a new block appended to the blockchain on average every 2.5 minutes, compared to Bitcoin's 10 minutes. This is claimed to bring faster transaction confirmation and an increased resistance to double spending attacks (for the same network computing power), in exchange for an increased blockchain size and more frequent orphaned blocks. Second, an alternative algorithm for the Proof of Work problem: instead of Bitcoin's *SHA-256* hashes, Litecoin uses *SCRYPT* **[Per12]**, a key derivation function intentionally very slow and costly to execute on custom/parallel hardware, due to intentional extremely high memory requirements. This different PoW ensures a fairer distribution of computing power in the network, by limiting the performance of hardware which is not a simple CPU and thus giving "normal" users more power.

Litecoin is one of the most supported altcoins, frequently available in most exchanges, and widely adopted and usable for transactions. Its advantages over the other available currencies are its technical innovations and early appearance on the market, united with a strong acceptance by the public.

4.1.3 Mastercoin

Mastercoin **[57]** is a particular altcoin, without its own separate blockchain, first described in detail in the white paper "The Second Bitcoin Whitepaper" **[58]**. Due to its characteristics, it is actually usually not considered an altcoin, but a different cryptocurrency. Mastercoin uses what is called "Master Protocol", it offers a set of new features and its own coins by utilizing the already existing Bitcoin network, data and communications. The system exists as a subset of special transactions that can be found in the Bitcoin blockchain, these transactions contain as their output special addresses (still valid Bitcoin addresses) which actually encode Mastercoin calls and actions: a new programmable layer over Bitcoin.

Mastercoins cannot be mined, all the existing ones have been generated by a special, hardcoded, one-time-only procedure when the currency was born and started creating transactions: via a special public campaign, anyone who sent bitcoins to an advertised Bitcoin address received an amount of mastercoins proportional to the sent coins and the time (earliness bonus) of the transaction, effectively exchanging coins between the currencies.

Saving Accounts are the first feature: specially marked addresses whose transactions can be reversed for a restricted period of time by a linked different address, this works as an additional layer of security. With *Distributed Currency Exchanges* any user can publish a buy/sell order/offer, announcing how many bitcoins are exchanged for how many mastercoins (and vice versa), when the offer is matched the coins are automatically transferred; this will also work for any additional cryptocurrency which can easily be created by anyone as altcoin of Mastercoin. *Forex*[2] *Price Feeds* allow anyone to publish price feeds on the blockchain, telling what value a certain currency or object has; users can then choose reliable sources to receive updated and informed information about the current market situation. Related to the previous feature it is also possible to have an advanced, verified *Betting* system, where users can post bets regarding price feeds, which can then be accepted by other users and are automatically regulated by the Master Protocol.

Mastercoin supports also other interesting additional features, mentioned in the initial paper, like *Escrow Funds* and *Backed Currencies*. Thanks to its integration and usage of the "underlying" Bitcoin system, the protocol is frequently updated and can easily gain more functionalities over time.

4.1.4 Peercoin

The altcoin *Peercoin* **[59]**, also known with the names *Peer-to-peer coin*, *P2P coin* and *PPCoin*, has been originally described in the white paper "PPCoin: Peer-to-Peer Cryptocurrency with Proof-of-Stake" **[Kin12]**; it is the first coin that implemented a combination of Proof of Work and Proof of Stake **[24]** to secure the system and it's one of the most used cryptocurrencies. Peercoin differs with Bitcoin in four main areas: coin limit, transaction fees, inflation/deflation, and the previously mentioned Proof of Stake (PoS).

While Bitcoin has a fixed limit on the number of coins that will ever be mined, Peercoin does not, opting for a steady 1% inflation per year leading to a theoretically infinite amount of coins being mined. The altcoin implements fixed obligatory fees for each transaction, the fees however are not

[2] Forex: Foreign Exchange Market, a global decentralized market for currency trading.

given to the miners discovering the block where the transaction is included, but are instead destroyed, creating deflation and thus balancing the inflation part of the system.[3] With Proof of Stake, Peercoin is more secure against attacks and problems deriving from the "majority" simply being decided by raw hashing power, to do that the currency considers what is defined as "coin age": the product of the amount of money considered times the period of time it was held for. Blocks can be generated through both Proof of Work and Proof of Stake, with the longest, main blockchain being selected however by the highest combined coin age instead of difficulty. When trying to generate a new block through PoS, the miner keeps sending their own coins to themselves, thus consuming coin age in exchange for a reward.

Thanks to Proof of Stake, the Peercoin system is much more energy efficient of Bitcoin, not requiring miners to use too much hashing power to generate new blocks. As time passes, PoW difficulty will increase and lose attractiveness in the network, while PoS will become the main source of coin generation; this means better energy usage, reduced wastes, and the "power" of the system (majority) being held by the user actually possessing the currency itself, thus interested in the well-being of Peercoin.

4.1.5 Dogecoin

Dogecoin **[60]** is an altcoin derived from Litecoin; its name, logo and not so serious objectives derive from the "Doge" internet meme.[4] This cryptocurrency, while not providing any substantial changes and new features if compared to other coins, is one of the most famous around the world, with a big user base and high exchange rate.

Dogecoin's changes over the original Litecoin code and system are: limitless supply and creation of coins (inflationary currency), faster average block creation time (1 minute) and initially random block mining rewards. Dogecoin's community and usual money usage are also in general considered more light-hearted and not as serious as all other cryptocurrencies.

This altcoin can be seen as the perfect example of how a decentralized virtual currency's value is given by the community utilizing it: it started as a funny and playful social experiment on the wave of dozens of newly founded altcoins, its increasing adoption rate brought it to be one of the top cryptocurrencies in circulation. Dogecoin is very frequently used as an internet tipping system between users; it has at the moment limited real-world

[3] This is due to the "network effect", in the words of the developers:
"If inflation increases, then transaction costs become cheaper which increases the urge to participate in transactions. However, if deflation increases, then transaction costs become more expensive which increases the urge to save more, which in turn causes inflation.".

[4] The "Doge" internet meme – a Shiba Inu dog with colourful broken English monologue lines in Comic Sans MS font: **http://en.wikipedia.org/wiki/Doge_(meme)**.

commercial applications, due however to its rising popularity, the value and exchange possibilities are continuously increasing.

4.1.6 Zerocoin/Zerocash

Project *Zerocoin* **[61]** originally started as an extension of the Bitcoin protocol providing advanced privacy features, as described in the white paper "Zerocoin: Anonymous Distributed E-Cash from Bitcoin" **[Mie13]**. After a series of collaborations and more work however the project has been renamed to *Zerocash* **[62]**, providing improvements over the original and a deployment as a new separate altcoin utilizing the Bitcoin network.

Zerocash should not be confused with the already available *Anoncoin* **[63]** which claims to implement some of the basic features of Zerocoin; at the moment of writing Zerocash has not yet been released but should see the light in the upcoming months following further work. Further improvements related to privacy in Zerocoin have already been proposed in **[And14]**.

The Zerocash protocol generates a new altcoin by modifying and extending the existing Bitcoin client, utilizing the same network. It provides two new coins: *zerocoins* and *basecoins* which are interchangeable and respectively anonymous and non-anonymous. Payments done via zerocoins guarantee anonymity via zero-knowledge proofs[5], not revealing sources, recipients or amounts of money being transferred while at the same time proving the correctness of the transactions ot others.

Users are free to convert bitcoins to basecoins/zerocoins using *mint transactions*, the bitcoins are cryptographically committed[6] to the new coin having a certain value, identifier and owner address, with the first two not being revealed thanks to the commitment. *Pour transactions* can then be used to transfer the new coins to different owners in complete anonymity: by using zero-knowledge proofs a user can demonstrate that they are the owner of the coins used in the transaction plus other information needed for a valid transaction, without revealing anything about themselves.

Zerocash advertises (and has been proven to guarantee) strong anonymity features in an easy to use fashion through complex cryptographic primitives. It is a large and very promising altcoin project, probably one of the more complicated, offering advanced privacy features which were long requested by the Bitcoin community. The technical paper describing protocol and used function, "Zerocash: Decentralized Anonymous Payments from Bitcoin"**[Ben14]**, has already been scrutinized by several different parties and looks sound, while it's only a matter of time for the practical implementation to be used and analysed.

[5] Zero-knowledge proof: **http://en.wikipedia.org/wiki/Zero-knowledge_proof**.
[6] Commitment scheme: **http://en.wikipedia.org/wiki/Commitment_scheme**.

Chapter 5

Bitshark

In this chapter we will describe and analyse *Bitshark*, the custom Bitcoin client used to connect and extract "interesting" information from the network. Originally developed by Mathias Wellig for his master thesis,[1] the system has been modified and improved to add new functionalities in order to reach the objectives of this new work. Details regarding the new features and exact changes over the previous work can be found in section **5.3**.

The software is intended to behave as closely as possible as a normal user of the network, not interfering with the normal operations and data exchanges and trying to minimally affect the resources of the connected clients. Bitshark tries to connect (and being connected) to as many network nodes as possible, in order to gather all the needed data; running such nodes with the right resources brings benefits to the Bitcoin system, as information is transferred more quickly through the network.

We start with a description of the whole system, what it does and how it's deployed on different machines. The different components are analysed in detail, followed then by a description of the data the system provides, a list of modifications over the original software and statistics of its behaviour.

5.1 System Description

The Bitshark client is a modification of the freely available Bitcoin Core code (**3.1.7**), re-adapted to log any interesting data we come across while exchanging messages with the peers we are connected to. Using the already existing code allows us to have a system that can easily be updated to support new versions of the protocol, behaves similarly to "normal" clients in the network, and is guaranteed to follow the right rules.

[1] Mathias Wellig: "Bitshark – Real Time Data Analysis of the Bitcoin Network". MA thesis, *ETH Zürich*; 2013.

The work on the program focused mainly on two points: the previously explained logging of data, which mostly comes from added statements logging information which is normally thrown away after use; and the support for a high number of connections to nodes in the network. In addition to that, code has been developed to ensure the synchronization and correct behaviour of multiple clients running the custom Bitcoin client, together with functions and protocols to gather, organize and merge information and logs coming from the multiple sources.

A completely new and separate program, the "Analyser", was developed to extract information from the logged data, perform experiments and gather results as presented in section **6.3**. Although it is not in any way related to the Bitcoin base code, its deployment and interaction with the software and gathered data makes it as well part of the Bitshark system.

The logged data, described in detail in section **5.1.5**, is the final product of the custom Bitcoin client and focuses mainly on transaction and entity (connected peers) information. The data, together with timing info, can be used for some very interesting analyses, explained in the next chapter, that allow us to associate clients and IPs to transactions.

5.1.1 Architecture and Deployment

The Bitshark system is composed of several logical components; depending on the function these components run specific software and access different data and/or external networks. A scheme of the general architecture of the system can be seen in figure **5.1**, a description of the parts will follow.

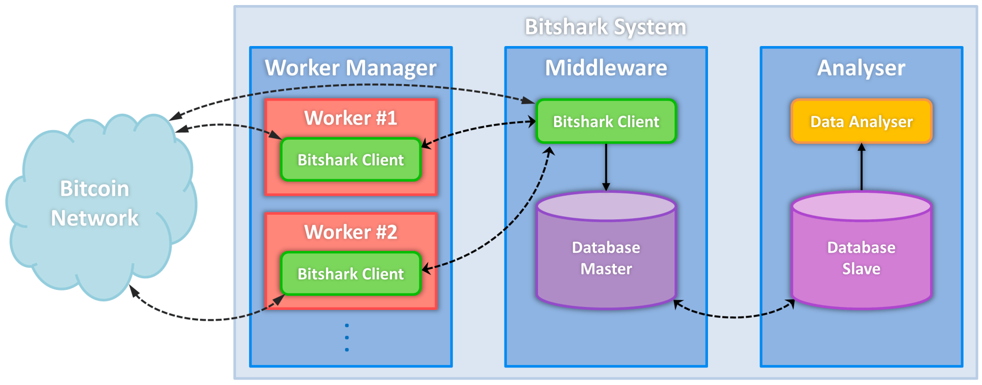


Figure 5.1: Bitshark system architecture.

The first component is the *worker* (**5.1.2**): this part is responsible for communicating with the Bitcoin network to log information; multiple instances of it run concurrently and send information to the *middleware* component (**5.1.3**).

The middleware is responsible of collecting, merging and storing the data received from the workers via the new *communication protocol* (**5.1.4**) in a *database* (**5.1.5**); the latter is also used to keep information required to maintain the whole system active. Finally, the *Analyser* component (**5.1.6**) is in charge of extracting data from the database, analysing it and reporting its findings, shown in the next chapter.

Hardware and Software

The whole system runs on three consumer-grade desktop computers: one machine is for the workers, one for the middleware and the main database, and one for the Analyser software with a secondary database.

The first machine has the hardware configuration described in table **5.1**, the system runs *VMware vSphere* **[64]**, a server virtualization operating system based on the *ESXi* hypervisor **[65]**. Although this is a proprietary solution, a free time-unlimited version is available, limited only in the usage of advanced tools plus with a restriction on a maximum of 8 virtual processors per VM and 32 GB of RAM (limits not reached by our hardware). The workers are run inside *Ubuntu Server* **[66]** virtual machines customized for very low memory usage, which in turn are managed by vSphere[2]. Each VM has at its disposition a virtual CPU with the same details specified in the hardware table, 50GB of disk space and 1.8GB of RAM; the resources are used to exclusively run a Bitshark Bitcoin client. The hypervisor machine has at its disposition a range of public, externally accessible IPs to assign to the virtual machines (plus itself of course), this ensure a correct and optimal behaviour of the clients in the Bitcoin network.

CPU	Intel Core i7-4770, 4 cores, 3.4 GHz
RAM	Corsair Vengeance, 32GB (4x 8GB), DDR3, 1333MHz
HDD	Western Digital RED, 1TB, SATA 6Gb/s
Network	1Gb Ethernet

Table 5.1: Hardware configuration for the workers' (vSphere's) machine.

The second machine (Middleware) contains the hardware listed in table **5.2**, the operating system is *Ubuntu Server 12.04 LTS*. There are two main programs being run on the middleware computer: the first one is the Bitshark client code, run with particular options that make it behave differently from the workers and act as the entity responsible for collecting and merging data; the second one it the main database, which contains all the data to keep the state of the program updated, and to log all interesting information. The

[2] We have a total of 16 workers, all assigned to the same hypervisor.

database system used is *MongoDB* **[67]**, a document-oriented NoSQL storage system; we utilize its replication capabilities over multiple machines to optimize workload separation, this machine contains, as previously mentioned, the primary database (master).[3]

CPU	Intel Core i5-4670, 4 cores, 3.4 GHz
RAM	Corsair Vengeance, 16GB (4x 4GB), DDR3, 1600MHz
HDD (OS)	Western Digital Caviar SE, 160GB, SATA 3Gb/s
HDD (DB)	2x Western Digital RED, 3TB, SATA 6Gb/s , RAID 1
Network	1Gb Ethernet

Table 5.2: Hardware configuration for the middleware machine.

The third machine utilizes the same hardware as the one used to run the hypervisor and is described in table **5.3**, the operating system is *Ubuntu Server 14.04 LTS*. The server runs two programs: a database system, continuously running, and the Analyser software, being run whenever we need to do experiments with the collected data; to collect geolocation information about the recorded IP addresses we also use *MaxMind GeoIP* data **[68]**. We have again *MongoDB*, running as slave, replicating part of the data found on the primary database on the middleware machine. The Analyser program runs and extract information by taking the data stored in the local DB.

CPU	Intel Core i7-4770, 4 cores, 3.4 GHz
RAM	Corsair Vengeance, 32GB (4x 8GB), DDR3, 1333MHz
HDD	Western Digital RED, 1TB, SATA 6Gb/s
Network	1Gb Ethernet

Table 5.3: Hardware configuration for the Analyser server machine.

5.1.2 Workers

Workers run a modified version of the Bitcoin client: a client called Bitshark run in "worker" mode, connecting to the Bitcoin network and behaving mostly like a normal node, differing in the area of connections and logging of data and with new functionalities for the relations with the middleware.

[3] See the MongoDB manual for replication, master-slave replication: **http://docs.mongodb.org/manual/core/master-slave**.

Connections

In the original Bitcoin core implementation a node is limited to 8 outgoing connections and 125 total connections (including the incoming ones), the client has a thread continuously running that keeps trying to connect to new nodes whenever the number of connections is insufficient, the list of possible nodes to connect to is known from previous history and from the advertisements of other clients over time.[4]

Bitshark rises the limits to 700 and 950 for respectively the output and total connections, in addition to that, multiple threads[5] are used to parallelize the connection operations and greatly increase the speed at which peers are connected to actively. In order to have multiple clients collaborate for the collection of data, addresses that are tried to connect to are not the ones locally known: collected advertised addresses are stored and regularly[6] sent to the middleware, new addresses can also be requested at any time again from the middleware. If a client in the Bitcoin network actively connects to one of the workers (which thus accepts an incoming connection) and another worker was also connected to it, the middleware orders the latter to disconnect and forget that address, while passing "control" of the address to the former worker. This way the system can dynamically adjust and decide which set of addresses workers are responsible for and ensure that the whole infrastructure is only connected once to any network nodes.

Subnets

In the default Bitcoin code, a node avoids to connect to multiple nodes if their addresses share the same `/16` subnet,[7] this is made to avoid selecting "related" clients which, considering the limited amount of connections, statistically don't contribute much to updated data and lead to a restricted connection to the Bitcoin network; this limitation is removed in Bitshark.

Network Interaction

As explained in section **3.1.6**, the Bitcoin protocol utilizes several different messages to exchange data across its network. The Bitshark modified client tries to act as much as possible as a normal client, supporting the network during its logging procedures and behaving as passively as possible.

[4] See the `addr` message, in section **3.1.6**.
[5] 50 threads in the current implementation.
[6] The interval is set to 10 seconds, with backoff in case of problems on the middleware.
[7] The specified CIDR notation implies that addresses from XXX.XXX.0.0 to XXX.XXX.255.255 are in the same subnet; see: **http://en.wikipedia.org/wiki/CIDR_notation**.

Bitshark has only one active part in its behaviour: when a new (inbound or outbound) connection with a peer is established, a series of pings is sent over a short period of time in order to try and estimate the link delay between the two nodes. The pings are done via the already present messages in the Bitcoin protocol, they put an insignificant load on the target nodes (who only have to send the relative message back). This active procedure makes later analyses of the data more precise, but is not required to get good results.

The modified client answers to most of the queries made by other peers as the default implementation; a list of the modifications done for performance reasons and the relative involved messages follows.

All transactions are relayed as usual through the `inv-getdata-tx` procedure: Bitshark asks for new transactions advertised via `inv` and, once they are received and verified, rebroadcasted and transferred via further `inv` and `tx` messages. Blocks however are not relayed: the system only asks for new blocks advertised via received `inv` packets, but doesn't relay the data to other connected peers. The reason for this lies in bandwidth consumption: due to heavy block sizes and new clients needing to download big parts of the blockchain, plus the fact that Bitshark is connected to a large amount of nodes, the bandwidth used would be too high; `inv` messages are not sent after the receipt of a new block and `getdata` requests for blocks are rejected.

The Bitshark software also additionally ignores the `mempool` request, which asks for a list of all transactions in memory which have been verified but not yet confirmed. Normal clients usually never need this kind of information, that would also result in higher network usage due to the high number of transactions we keep in memory thanks to our high number of connected nodes. More details about this and other messages can be found in the protocol specification page of the Bitcoin wiki **[28]**.

Data Logging and Transmission

The main purpose of the Bitshark modified client is to log "interesting information"; in the current system this translates to connected entities (network nodes), transactions and bloom filters (**3.3**), together with global timing info. Data is stored on every worker in particular buffers; depending on different rules, data that is old enough or "complete" is transmitted to the middleware and erased from the buffers at regular intervals.[8]

Information about entities is stored in two moments: one being when the connection is established, and the other when the remote client disconnects (or gets disconnected). In the first we immediately store the new just discovered data about the client: besides the current timestamp representing the

[8] A separate thread checks the logged data to be sent and erased every 30 seconds.

"from-time" and the client's IP, a new UUID[9] is generated and assigned to the node's IP during its lifetime (the time it remains connected to us); this data is marked to be sent as soon as possible to the middleware. Before the client disconnects we usually receive a `version` message from it, giving us details about the client name and version used; we also try to discover and estimate the minimum link delay, by sending a series of Bitcoin-pings and listening for answers. When we lose connection with a client any additional information, including the current "to-time" timestamp, is marked as ready to be sent to the middleware.

Transaction data logging is executed whenever a new `inv` message is received advertising a newly discovered (received or created) transaction by a client. What we store is a list of all the clients we are connected to which advertise knowledge of the transaction, together with a timestamp representing the moment we receive the message from that specific node. The worker uses as data structure for this information a map of transaction hashes to a list of pairs (entity id, timestamp). The transaction hash used is simply the one advertised in the `inv` message, while the entity id is the string associated with the node (IP) which sent the message to us. After enough time[10] has passed since the first advertiser of the transaction, we can send the collected information about the specific hash to the middleware.

Bloom filters are publicly advertised by clients running Simplified Payment Verification versions of the Bitcoin software to any peer they connect to via a special `filterload` message. Bitshark records the filter as it is given and passes it to the middleware for storage in the database as soon as possible.

Appendix section **A.1** describes the formats of information as stored on the database, these data structures are almost identical to the ones used internally by the workers and can help to better understand the previously listed details about the gathered info.

5.1.3 Middleware

The middleware machine has three important roles in the system: handling workers, merging and completing received information and storing data permanently in the database for later analyses. There are two programs running on the computer: the Bitshark client, run in a special "middleware" mode, for the first two roles; and a MongoDB instance for database interaction.

[9] Universally unique identifier, see: **http://en.wikipedia.org/wiki/Uuid**.
[10] Currently the limit we use is 30 minutes, though this number could be lowered if needed to save storage space, while not affecting the results of the currently executed experiments.

Worker Management

Workers can run independently, they however need the middleware to coordinate the efforts for covering as many connected clients as possible and storing the logged data accumulated in their buffers permanently.

The middleware knows of all workers that are currently participating in the data logging procedure, new workers check in with this machine and rely on it to receive and update the list of addresses they are responsible for. The middleware continuously checks the online status of workers and keeps information regarding IP assignments in memory and in the database to be always up to date with the situation even in case of unforeseen failures.

Workers can at any given time report newly received IP addresses of nodes (from advertisements) or request for new addresses to work on in addition to the already assigned ones. The middleware will in the first case add any really new IP to the common pool of addresses available to be given to a requesting worker, and in the second case select a certain number[11] of IPs, assign them and send them to the needing worker. Workers can also announce new incoming connections, that the middleware has to check to reassign (if needed) the specified IP from the previous owner (which will then disconnect from it) to the new worker.

Details about the protocol and functions used to communicate between middleware and workers can be found in section **5.1.4**.

Data collection and storage

The middleware puts the data received from the workers in special buffers divided by data type; at regular intervals the collections are traversed by apposite threads which check which data is to be stored on the database and eventually be discarded from the buffer structures.

Bloom filters are logged by the workers as they are received, without any additional information; the same bloom filter is logged in the same way by any worker. Due to this property this data can be stored by the middleware in a simple set without any special care; at regular intervals all entries are always selected to be written to the database (only if the information is not already present) and the buffer is then emptied.

Entity information comes with random UUIDs generated by the workers; as mentioned in the previous section the logging of this information happens in two periods on the worker: on connection and on disconnection of the remote node. Entities are stored in a map on the middleware, using as key the mentioned random unique id; in the case that we receive a new entity which is already in the current buffer, we replace the previous information as

[11] 1000 IP addresses per request with the current settings.

we know the updated entity must come from the same worker and contains more complete data. While iterating through this buffer we again always select all entries to be sent to the database and then be deleted; we however perform a special insert (update) on the DB so that any previous information is overwritten and updated if needed.

Transaction information uses as identifier the transaction hash, used for the same purpose also in the rest of the Bitcoin software; as buffer on the middleware we use again a map, with as key the previously specified hash. Transactions contain two very important types of data: the input addresses and the list of originators with their timestamps. For every new piece of information that arrives to the middleware we first check if it's already present in the buffer, if it isn't we simply add it, otherwise we proceed to merge the originator lists by inserting the new elements in the current list for the transaction. To merge the originator lists we first check for duplicates: cases where a node with a certain IP is connected and sends an `inv` message to multiple workers, resulting in list with entries for originators with different ids but same IP. We solve the (rare) duplicate problem by simply keeping the entry which has the lowest registered timestamp. The rest of the merging procedure ensures that the list is kept in order of time, beginning from the first originator up to the latest one. During the regular check of the buffer for transactions that are ready to be written to the database, for each entry we look at the time it was last added to the map or modified by the previous merge procedure: if enough time has passed we consider the information complete (meaning no other worker should send data about it), write it in the DB and erase it from the buffer.

Transaction information completion

The transaction information arrives to the middleware with no information about its inputs; as explained in section **3.1.3**, Bitcoin transactions don't directly contain input addresses, but instead refer to old transaction outputs. Due to that retrieving them becomes computationally expensive to do on the workers, given the amount of received transaction and the limited resources.

The middleware takes care of filling in this missing information, by keeping its Bitcoin client connected with a limited set of nodes (like a normal client), ensuring updated blockchain and transaction data. Whenever a transaction is old enough to be selected to be written to the database, the software searches in the local memory the needed information about previous transactions and fills the list of address inputs with the retrieved data before sending everything to the DB.

5.1.4 Worker-Middleware Communication

Data transfers and remote function calls between workers and the middleware, both running a modified version of Bitcoin Core, are done through JSON-RPC **[30]**, a lightweight remote procedure call protocol already used in Bitcoin. The middleware is the part implementing the new custom functions, which can be called programmatically by the workers (or "manually" via scripts / command line) to provide different results and/or effects. All the calls done from the workers to the middleware are also used to synchronize time between the two parties. A list of the most important new functions available follows; complete examples of the precise structure of invocations and answers can be found in the appendix at section **A.2**.

- **getIPs:** Workers call this method to signal to the middleware they would need new additional IPs to process (try to connect to); this is the first function a newly created worker calls on the middleware and is also used to register it with a new identifier on the middleware to keep track of the assigned IPs. When there are addresses that are available (not assigned to any worker), the middleware answers by sending back a group of IPs and marking them as assigned to the worker.
- **newIPs:** Workers keep receiving advertisements of IPs for nodes they could possibly connect to; in the Bitshark modified clients these addresses are however only stored in a separate buffer and sent to the middleware through this function call. The middleware will on receipt add them to local collections depending on their known/new status and to a special permanent collection on the database.
- **newLogs:** This function is called by workers on the middleware whenever new data that has been logged locally is ready to be sent and then logged on the database. Depending on the information that needs to be transferred, the method is called with a different payload which could be composed of transactions, entities, or bloom filters.
- **newIncoming:** When workers receive a new incoming connection from a node in the Bitcoin network they announce that to the middleware, which registers the new assignment. If the node was already assigned to another worker the latter is contacted to disconnect and forget it.
- **removeIP:** This is the only function which is called by the middleware on a worker and is related to the previous one in dealing with new incoming connections. When the middleware discovers that a new incoming connection with a certain IP has been established by a worker, it will call this method with the mentioned address on the previous worker responsible for the IP (if any), to make it disconnect and not try to connect again to it in the future.

5.1.5 Database and Logged Data

As database management system we use the already mentioned *MongoDB* NoSQL open source software, running with a master-slave configuration on respectively the middleware and the Analyser server. The master database works as any type of other DB, accepting new data, modification and reading operations; while the slave database works as an always automatically updated copy of the master, accepting only read operations. With this type of deployment we ensure that the master server is always quick and responsive when writing new data coming from the middleware, even if we at the same time run read-intensive operations for our analysis on the other server.

The stored data can be divided into two big databases `bitcoin_state` and `final_logging`: the first is used to save temporary data and information about connected and disconnected nodes in the history of Bitshark, while the second is used to keep track of all the logged transaction, entity and bloom filter information mentioned in the previous sections.

The `bitcoin_state` database contains data about nodes: when they were connected to us, for how long, which version of Bitcoin they use, to which worker they were assigned last, etc. We could use part of this knowledge to recover more quickly in case of an unexpected middleware crash and later recovery, to restart the workers and divide the workload as previously stored, however at the moment we don't make use of it as we never experienced important bugs since the various updates to our software; we keep logging this information as it is easily obtainable and could be useful in the future for other types of analyses. This database is not replicated on the slave server and is only updated and accessible locally on the middleware.

The database `final_logging` contains all the data we need for the analyses presented in this thesis; the data is added incrementally as time passes, excluding some special cases, the whole storage can be seen as a big append-only log. We have three collections in the database that keep the data for respectively transactions, entities, and bloom filters; the structure of the stored data can be seen in **A.1**. In addition to these, we also have the collection "assignable transactions", which can be seen as a subset of the transaction collection. Since workers receive and log all transactions that they receive via `inv` only messages, we end up with a lot of logged data about transactions relayed only by one originator (or rarely few of them), due to them being not well connected, the transaction not being valid, or other issues. For our purposes that will be better explained in the next chapter, we then keep this special collection populated only by transactions with a certain minimum number of logged originators.[12] All the valid transactions that are broadcasted through the Bitcoin network are logged here.

[12] In the current software version, the minimum is of 500 originators.

For each transaction we log the following fields:

- **_id:** The identifier of the transaction is the tx's own hash as used in the Bitcoin network and advertised in *inv* messages, used in string form.
- **originators:** List of originator objects, each originator is composed of an *id* and *timestamp* field; the list is saved in ascending order of timestamps. The *id* field links to the respective entity in the entity collection, the *timestamp* is given as Epoch Time[13] in milliseconds.
- **inputs:** List of Bitcoin addresses used as inputs in the transaction, given in *Base58Check* string form (see **3.1.1**); could be empty due to the receipt of a single *inv*, duplicate addresses are only stored once.

Entities are stored with this list of properties:

- **_id:** The identifier of an entity is a random UUID generated and assigned by the worker connected to it, saved in the default string form.
- **ip:** IP address of the entity (no port information).
- **version_name:** Name of the used client as advertised in the received *version* message; could be empty, in case of multiple advertised user agents[14] the main (first) one is used.
- **version_number:** Version number of the used client as advertised in the *version* message; same indications as for the version name.
- **time_online_from:** Epoch timestamp in milliseconds when we connected, actively or passively, to the entity.
- **time_online_to:** Epoch timestamp in milliseconds when we disconnected for any reason from the entity; a zero indicates that the Bitshark system still has an established connection with it.
- **link_delay_minimum:** The minimum recorded link delay between a series of Bitcoin pings/pongs, measured as half the milliseconds between a sent *ping* and a received *pong*; a value of 1000 indicates an unknown delay due to an insufficient number of answers.

Finally, bloom filters are saved as follows:

- **_id:** A serialized hexadecimal string of the fields contained in the bloom filter, the contained variables are the byte data vector (variable length), number of hash functions (4 bytes), random nonce (4 bytes), and special flags (1 byte); all of them are separated by a dash.

[13] Unix time: **http://en.wikipedia.org/wiki/Unix_time**.
[14] Bitcoin improvement proposal – Protocol Version and User Agent: **https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki**.

5.1.6 Analyser Server

On this computer we run our analysis software which extracts the data stored on the local slave database, transforms it in what we need and tries to estimate related and unrelated addresses, leading to the hiding set size.

Once the data has been logged and stored by the Bitshark workers and middleware we need no further interaction with the Bitcoin network or the blockchain to perform our work, we only need the DB plus a local map of addresses to geographical locations. The analysis process uses all the resources available, thus having a separate machine working on it is essential, especially if the logging and analysis processes are performed in parallel.

The Analyser program's purpose is to utilize all known (old and new) heuristics and ideas to form a map of related and unrelated addresses in a completely automatic way. We look at the addresses appearing in the transactions we logged, which correspond to (at least) the transactions recorded in the blockchain in the same period, and see what we can infer about them in relation to the others over time. As this software and the ideas behind it are the main new product of this thesis and cover a lot of ground, the details regarding them will be presented later in the context of the study and results. A thorough description of the Analyser software, flow of the program, and utilized heuristics will be presented in section **6.3**.

5.2 Statistics

We will now quickly present some statistics about the Bitshark system and the data logged during our experiments. All these numbers refer to the time periods when the system was fully active (without boot and shutdown times) and connected to multiple nodes in the Bitcoin network.

Every hour on average we connected to and logged 2587 entities, from them we received and recorded 8 bloom filters and 2956 transactions, of which 2505 were assignable ones broadcasted throughout the network.

On the database an entity takes on average 0.2KB, a bloom filter 4KB, and a valid (assignable) transaction, with its lists of inputs and originators, 357KB. After three weeks of measurements this leads to 54MB of bloom filters, 354MB of entities and 864GB combining all the different transactions together.

On average at any point in time we had around 7'000 outgoing connections plus 2'000 incoming connections. Each worker maintained, after an initial build-up period, a balanced and stable amount of outgoing connections; while the incoming requests slowly increased over time the more our nodes became known and available to connect. The combined network usage had an average of 1500KB/s in upload and 750KB/s in download.

We connected to a total of 1'303'752 different entities (remember that each client results in a new entity at each connection/disconnection cycle). Of the 1'303'752, 640'395 (49.1%) advertised their versions communicating with us and were not automatic bots; we excluded the clients "Snoopy", "shodan.io", "getaddr.bitnodes.io", and "bitcoin-seeder". In the previously mentioned connected nodes which behaved normally, we have the vast majority composed of *Satoshi* clients, 623'916 (97.4%), 12'303 (1.9%) known SPV clients ("BitCoinJ", "breadwallet", "IceWallet"), while the rest are mostly alternative Bitcoin clients ("libbitcoin", "btcwire", "Gocoin", ...). Chart **5.2** shows the distribution of the main client versions we were connected to; note that this doesn't represent the real percentages of (online) users, but just the connections of our own system over time.

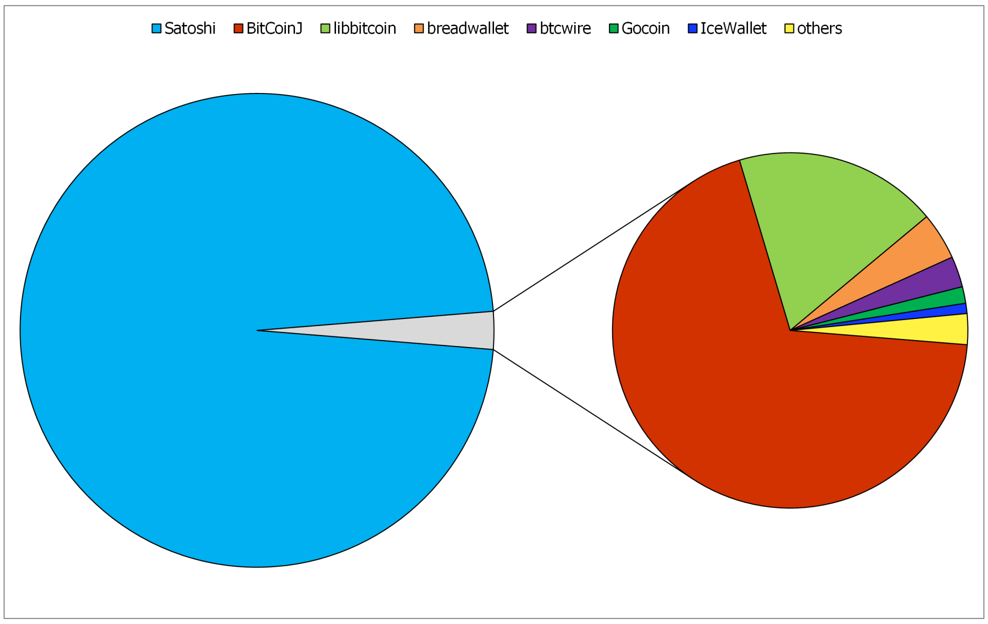


Figure 5.2: Bitcoin client version distribution.

5.3 Changes and Improvements

As mentioned in the beginning of this chapter, Bitshark was initially developed by Mathias Wellig during his master thesis, his work was meant to be the basis for further developments on the Bitcoin topic, which resulted in the current version that has been presented. We will now quickly go over the most important changes and additions/removals from the original software.

First of all the whole code has been repeatedly been ported to new and updated versions of the Bitcoin core software as they came out. The project was originally written for version 0.8.4, released on September 2013; the code has first been brought to version 0.8.6 and finally to the current 0.9.1. Keeping the software up to date not only ensured the obvious performance improvements and security assurances, but also guaranteed an optimal interaction with the Bitcoin network, support for all connected clients, any new protocol feature, and in general a positive status of the node in the environment.

The logged data was initially simple and thorough: all transactions and blocks received were logged as they were received, the transactions contained all the input and output scripts, while the blocks all the headers and contained data. In the current Bitshark version transactions only contain their hash, list of originators and input addresses, while block logging has been temporarily disabled altogether. These decisions allowed us to concentrate on the data we needed and also had the advantage of saving a lot of disk space for the data stored database, allowing for longer logged runs.

Entity logging is a new addition to the software, knowing when different nodes were online is an essential part of our analysis, other logged data like the list of originators in transaction has been modified to link to this new type of information available for analysis.

The Bitshark client running on the middleware didn't initially connect to any other node in the Bitcoin network, it was simply meant to send and receive RPC commands to log the necessary data on the database. In the current version the middleware acts as a normal node in the network, with a default amount of connections, keeping an updated blockchain. This change allows us to receive incomplete transaction information from the workers, and complete data about their input addresses by looking at locally known data on the middleware at the time of writing the information in the database.

There have been numerous performance improvements in several areas of the Bitshark client running on the workers. As each of them runs in a virtual machine with limited resources it is very important to make the software react well under stress and uncommon situations. To avoid additional problems, variables regarding the protection of the client against denial of service and bad behaviour have been tweaked to obtain stricter results.

The Analyser server and the very important software running on it are a completely new addition to the Bitshark system. Beforehand workers were run on two different physical machines, more workers were supported with more resources; now they are restricted and optimized on a single machine, while the other one is used exclusively for the analysis of data.

Lastly, the machine responsible for the virtual machines of the workers initially ran *XenServer* **[69]**, a free hypervisor for server virtualization, instead of the current *vSphere* **[64]** software. While the first was a great open source solution that worked for the initial experiments, after some time we noticed a quite high and inexplicable growth in required memory (RAM) during our prolonged experiments, combined with random OS crashes which threatened to ruin long running logging of data. The switch to the VMware solution brought to new and updated virtual machines, less memory and CPU consumption and in general a more stable environment for our ends.

Chapter 6

Data Analysis

In this chapter details about the data analysis procedure will be described.

A theoretical in-depth background on transaction relay (**6.1**) and the found possibility of assigning transactions to entities (**6.2**) will be given, together with data describing the chances of this procedure in different scenarios.

The chapter will continue with a definition of the Analyser software and all the utilized heuristics, together with an explanation of how different data about related and unrelated addresses can be combined together to obtain an estimation of the "hiding set" size; finally we conclude with a description of the actual program flow and execution (**6.3**).

6.1 Transaction Relay Details

The core part of this work is based on the fact of being able to assign (some of) the logged transactions to entities that were connected to our Bitshark system. In order to do that and understand why this works and the reasoning behind the used variables, it is important to go over how exactly transactions are created and broadcasted through the Bitcoin network, which is what will be explained in the next paragraphs. We concentrate our efforts on the Satoshi Core client,[1] which composes the vast majority of the full nodes (responsible for transaction relay) and, as shown in section **5.2**, in our experiments represented more than 97% of the connected clients.

Other studies **[Dec13] [Don14]** already analysed the broadcast and diffusion of transactions (and blocks) over the Bitcoin network. We are interested, however, in the precise timings involved in relays happening to near neighbour nodes; instead of the slow increase of knowledge over time, we look at the initial "burst" of data generated by clients creating new transactions.

[1] SPV clients running BitcoinJ follow a simplified relay procedure, where the transaction itself is sent immediately after creation to half of the connected clients, without *inv* or *getdata*.

Transactions are transmitted from peer to peer, following the sequence of messages *inv* $\rightarrow$ *getdata* $\rightarrow$ *tx*: node A announcing a newly received or created transaction via its hash, node B requesting the data of the transaction (if not already known), and node A answering with the transaction itself. Once the transaction is received by B, it will proceed to verify it in a usually negligible amount of time and relay it to its other peers. It can then be inferred that the lower limit for time needed to transmit any transaction between two nodes A and B is given by three times the link delay between the nodes.

The whole receive/send procedure of a Bitcoin client is run in a thread which executes the code reported in the appendix at **A.3**. Everything can be summarized as an infinite loop, where first all messages are received from all the peers and then all packets that need to be sent are transmitted again to all connected peers, every iteration usually runs in a negligible amount of time. The loop has a condition at the end that depends on whether there's additional data that has to be sent for various reasons to any of the nodes; if it's not the case, the loop (after sending any needed data to all connected peers) sleeps for 100 milliseconds.

If we consider again the case with the two nodes A and B, we can then see that before B is able to relay the transaction received from A to any other node, not only the link delay, but also the loop time has to be considered. Each time when one of the nodes has to receive a message and react to it there's the chance that the data arrives when the loop is sleeping, thus causing the total relay time to grow with a certain probability.

We collected data about the two variables: link delay and loop time; for the first we used the information already logged by Bitshark (by taking a random selection of 10'000 values), while for the second one a normal Bitcoin client has been left to run multiple times, connecting to a "normal" amount of peers, collecting timing information about the loop with varying amounts of connections and workloads.

Chart **6.1** shows the CDF[2] for the time it takes to perform one iteration of the loop in the Bitcoin Core client. It can clearly be seen how there are two distinct cases: about 83% of the times the loop doesn't sleep and the iteration is performed in negligible time (less than 5 milliseconds), in the rest of the cases the iteration takes slightly more than 100 milliseconds to complete due to the trigger of the sleep condition at the end of the loop.

For our upcoming calculations we will approximate this variable as a random number: 0 with 0.8343 chance and 100 with 0.1657 chance, independently from previous values; the probability of having exactly i iterations in a row without a sleep can be calculated as:

$$NoSleep_i = 0.1657 \cdot 0.8343^{i-1}$$

[2] Cumulative distribution function: **http://en.wikipedia.org/wiki/Cumulative_distribution_function**.

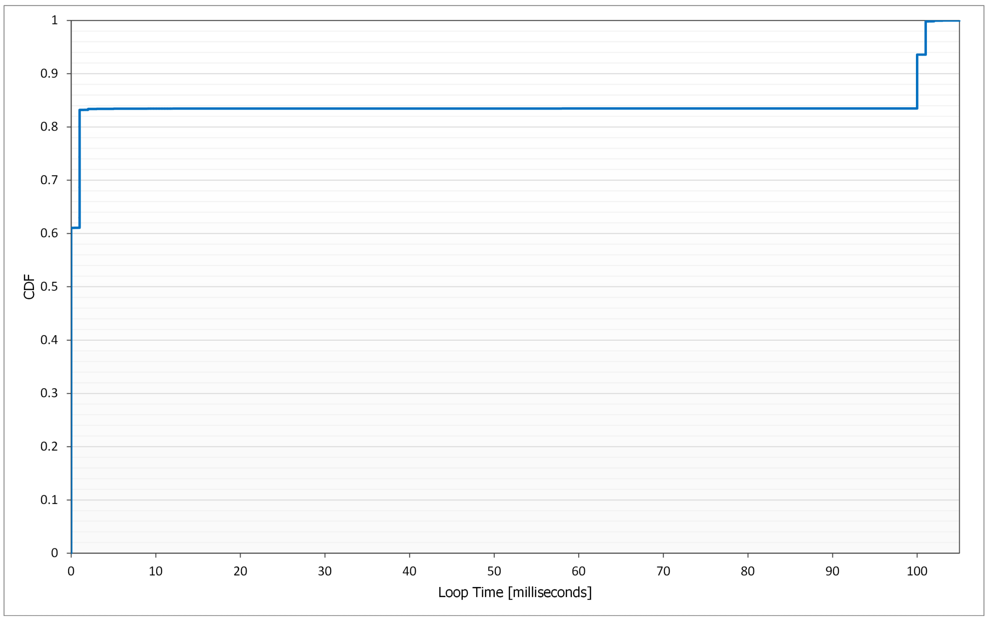


Figure 6.1: Bitcoin client loop time cumulative distribution function.

Chart **6.2** shows the CDF for the link delay, measured by Bitshark through Bitcoin protocol pings/pongs. The peculiar structure of the chart that seems divided in steps of 50 milliseconds is given by the fact that we obtain this information by using Bitcoin messages: we depend on the loop time for receiving the *ping* and the *pong* messages, as they could arrive while the loop is sleeping. As the link delay is calculated to be half the time needed to get an answer after a ping, if a message arrives to either side while the loop is sleeping (the rest of the times the iteration time is negligible) then the link delay is increased by up to 100 milliseconds, resulting in a link delay being near a multiple of 50 with higher probability.

For future calculations it's important to remember that the link delays, which are measured as the minimum ever recorded for each node connected to Bitshark, in the majority of the cases, 97% of the times (0.9722), are equal or greater than 50 milliseconds.

Bitcoin has no mechanism in place for clients to select their "best neighbours" by picking nodes with the best delay or bandwidth; as users of the cryptocurrency are spread all across the world, usually nodes end up with connections between them having high link delays due to random decisions.

One last remark about transaction relay has to be done about the selection of nodes that will receive the *inv* advertisement for a newly created or received transaction. When a node receives a new transaction (that is verified to be correct) it will, at every loop iteration, select at random one of the connected nodes: this node will receive the *inv* plus any other advertisement that was

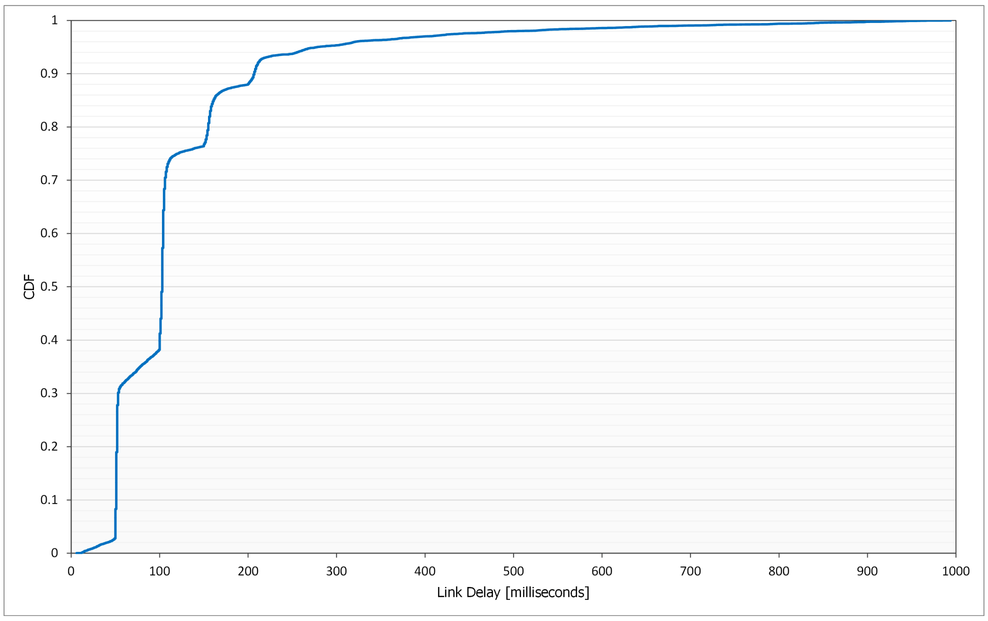


Figure 6.2: Bitcoin network link delay cumulative distribution function.

left to be sent to it. In addition to this, every connected node will also have a 25% probability of receiving any *inv* that had yet to be received (including the new one). The procedure is repeated at each iteration, until eventually the *inv* message gets sent to all the peers; the relevant code segment can be seen in **A.3**. When a node creates a new transaction, the same procedure is followed, but only for clients with versions equal or newer to 0.9.0 (up to the current 0.9.2.1). Nodes running older Bitcoin Core versions, which nowadays usually means from 0.8.0 to 0.8.6, use exclusively the "trickle node": at each loop iteration one peer at random is selected to be the trickle and will have a 100% chance to receive the *inv* for the newly created transaction, while the other nodes will have no possibility of receiving it.

With this information we can calculate the chance of being selected to receive an *inv* for a new transaction after a certain number of loop iterations (on the remote node), depending on the number of connections and client version used by the peer. Assuming we are connected to the transaction-originating node A, with c connections to its peers[3] and at loop iteration i, for respectively the older and the newer versions of the Bitcoin client, our chances of being selected at the current or in a previous iteration are:

[3] Including us, the node for which we calculate the selection probability.

$$\begin{aligned} SelOld_i &= 1 - \left(1 - \frac{1}{c}\right)^i \\ SelNew_i &= 1 - \left(1 - \left(\frac{1}{c} + \frac{c-1}{c} \cdot 0.25\right)\right)^i \end{aligned}$$

Table **6.1** shows some results for chosen example values; note that the chance that we are selected to receive an *inv* from a node which didn't create the transaction (only relaying it) is the same for all versions, given by the same formula used for the newer Bitcoin version.

Loop Iterations	1	2	3	4	5	6	7	8
ver. 0.8, 10 peers	0.10	0.19	0.27	0.34	0.41	0.47	0.52	0.57
30 peers	0.03	0.07	0.10	0.13	0.16	0.18	0.21	0.24
50 peers	0.02	0.04	0.06	0.08	0.10	0.11	0.13	0.15
70 peers	0.01	0.03	0.04	0.06	0.07	0.08	0.10	0.11
90 peers	0.01	0.02	0.03	0.04	0.05	0.06	0.08	0.09
ver. 0.9, 10 peers	0.33	0.54	0.69	0.79	0.86	0.91	0.94	0.96
30 peers	0.28	0.47	0.62	0.72	0.80	0.85	0.89	0.92
50 peers	0.27	0.46	0.60	0.71	0.79	0.84	0.88	0.91
70 peers	0.26	0.45	0.60	0.70	0.78	0.84	0.88	0.91
90 peers	0.26	0.45	0.59	0.70	0.78	0.83	0.88	0.91

Table 6.1: Selection chance for an *inv* message from the node creating a transaction.

6.2 Assignment of Transactions to Entities

We will now have a look at how we can decide if and to whom to assign a certain transaction in our analysis based on the Bitshark logged data. Everything revolves around the stored originators for each transaction: a list of all the entities that sent us the *inv* advertising it, ordered by time of arrival of the message. If a transaction is assigned, it will be linked to the first entity in the list, which is then said to be the source of the transaction.

The concept behind the assignment is quite simple: we compare the time difference Δt between the first and the second originator in our list; if the difference is "big enough" and the next few originators are not separated by a strangely high delay (indicating possible network/transmission problems), we assign the transaction, otherwise we don't. What we are doing is an analysis of the broadcast status of each transaction between the different peers, taking advantage of the fact that the Bitshark system is connected to the vast majority of the reachable full nodes in the network.[4]

If we are connected to the real originator of a transaction and we get an advertisement from it in the initial phases of the broadcast (loop iterations), then there's a high probability the next ones from the other contacted nodes will arrive after a certain delay later in time. If we are not connected to the real originator or receive the advertisement from the real originator too late, after many other entities received it and too much time has passed, then with high probability we will log many originator timestamps very close in time due to the broadcast mechanisms. A more formal definition will follow.

We have to set a precise limit value for Δt, as high as possible to correctly detect false positives and cases where transactions cannot be assigned, but low enough to correctly assign them when we find ourselves in the right situation, as seen later. Depending on the wanted precision level of the transaction assignments, this value can be set to lower (more assignments and false positives) or higher (less assignments and false positives) values.

We base our decision for the Δt limit on the measured link delays together with the information we have about transaction relay. We know that 97.22% of the nodes have a delay which is more than 50ms and that 83.43% of the times the receive/transmit loop doesn't sleep; also basically no connected node had a link delay lower than 17ms (99.95%). We set the Δt limit to **140ms**: we are then very confident that any node which is chosen to receive an *inv* will not be able to relay it before at least 150ms. The probability of a node B relaying a transaction (sending an*inv* advertisement for it) after more than 150ms after A began the relay to it is given by:

$$SlowRelay = 0.9722 + (1 - 0.9722 - 0.0005) \cdot (1 - 0.8343^3) = 0.9836$$

Meaning the chance that the node has link delay higher than 50ms, or the chance that it lies between 17ms and 50ms combined with the loop sleeping at least once during the $A \rightarrow B$ relay. In the following paragraphs it will be shown how this affects the assignment of transactions in different scenarios.

Lets consider the (very frequent) scenario where our Bitshark system is connected to the full node A creating and relaying a new transaction. To be able to assign the transaction we must receive the *inv* for it at a certain time, and receive the next *inv* from another node after at least Δt. We know that:

[4] We can confirm this by comparing the number of connections of our system with the public data available from the many automatic crawlers of the Bitcoin network.

- At A's loop iteration i the (accumulated) probability that we have been chosen is Sel_i, which is $SelOld_i$ or $SelNew_i$ depending on the Bitcoin client version being run on node A.
- At A's loop iteration i, given the number of connections c node A has with its peers, we will have on average $SelNodes_i = Sel_i \cdot c$ total nodes that are selected to receive the *inv*.
- We can safely ignore any node that is selected for relay after the loop on A sleeps or recursively by A's peers, as the passed period plus the time needed for relay would make their *inv* messages certainly arrive late enough in time to not affect our initial originator list positions.

For Δt to be high enough:

- Our Bitshark node must be selected before the loop on A sleeps, after i iterations, if that doesn't happen $SelNodes_i$ will begin relaying *inv* messages about the transaction and with an extremely high probability they will arrive (between each other or together with A's *inv* to us after the sleep) with a short delay between each other.
- Every other node that has been selected, $SelNodes_i - 1$ (excluding ourselves), must not perform a relay faster than 150ms; this is given by the formula $SlowRelay^{SelNodes_i - 1}$.

When these two conditions are met we will be able to assign the transaction to the first originator, which will be node A; when they are not, we will not assign it due to the low Δt. The probability of meeting the two conditions and thus being able to assign the transaction is obtained by the formula:

$$\sum_{i=1}^{\infty} NoSleep_i \cdot Sel_i \cdot SlowRelay^{SelNodes_i - 1}$$

We now consider a situation where none of our Bitshark nodes is connected to the node A, creating and relaying a transaction. In this case it is obviously impossible to correctly assign the transaction, as our chance of being selected is zero and we will never log any originator entry coming from A. We want to see how likely it is that we assign the transaction to the wrong entity, due to the delay between the first and the second originator being larger than our Δt limit. There are three main false positive possibilities:

- Every node that has been selected before the loop sleep, $SelNodes$, doesn't perform a relay faster than 150ms, except for exactly one which has an exceptionally quick total relay time, lower than 10ms. This would lead to a time difference between the first and second originator of more than the Δt limit, causing the assignment of the transaction.

- A single node is selected to receive the *inv* at the very first loop iteration, after that the loop sleeps and other nodes with high link delays are selected. Or, after the first loop sleep, no node could be selected (except the same previous node as trickle node), and the loop could then sleep again and again, until eventually other nodes are selected and will relay the transaction causing a high Δt measurement.
- Node A is connected to an exceptionally small number of peers, causing the probability of the previous case to rise. This would also mean an increased chance of few nodes having very different link delays, thus leading to a high Δt in many situations.

Many other combinations of choices of nodes, link delays, and loop sleeps are of course possible; the possibility of them happening, as well as for the ones presented, has not yet precisely been calculated as it depends on many factors, but is estimated to be very unlikely.

False assignments can of course also happen in the previous scenario, when we are actually connected to the real node creating the new transaction. The cases are similar to the ones presented in the previous paragraphs, with the additional condition that we should not get any *inv* message from the real source node for a long period of time (or it would cause Δt to be too small), thus making them even more difficult to happen.

SPV clients generating and broadcasting transactions[5] are easy to identify as source of transactions: since they never relay any transaction not originating from them, they can then be selected as real originators if and whenever they appear in the originator list. At the same time, in the case we are not connected to a SPV node which creates a new transaction, if said node has a decent amount of peers, it is easy to avoid false positives as half of its peers will receive the new transaction at the same time and will begin relaying it to their peers (which very likely include us).

In the real-world experiments we have noticed a slightly reduced probability of assigning transactions, this is due to the fact that *inv* messages' arrivals and log times can be "distorted" by sleeping times and computationally expensive operations being run on the workers; someone with more resources at their disposition wouldn't have this issue.

Chart **6.3** presents the assignment probability values for the common scenario where we are connected to a full node creating and relaying a new transaction; we show the obtained values with said node connected from 1 to a maximum of 125 peers (the default limit in the default client code).

[5] Even though it's a known fact that a big part of the Bitcoin users connect through SPV clients, we encountered few such nodes online at any time during our measurements; also, the percentage of transactions created and broadcasted by them was almost insignificant.

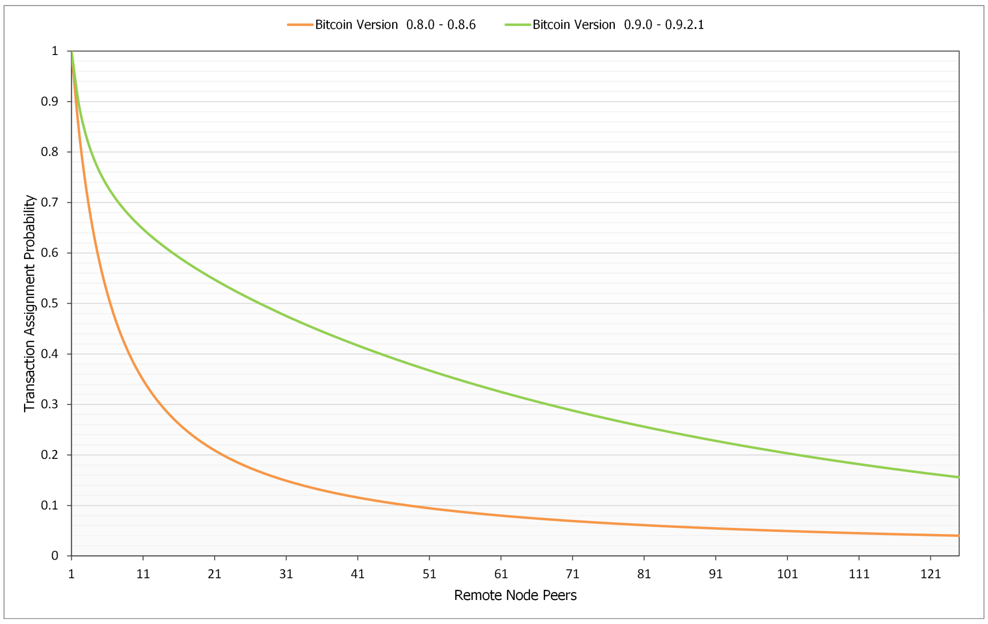


Figure 6.3: Transaction assignment probability when connected to its source node.

The results vary depending on the number of connections on the remote node and the two probabilities of being selected to receive an *inv* message due to the two different Bitcoin Core versions. We consider the chances of succeeding in assigning transaction quite significant depending on the situation, and we will show the practical results and effects of the created links between transactions and entities in our upcoming analyses.

It should be noted that the previous assignment probabilities are related to the fact we have a single connection to the source node, as it's the case of our Bitshark system. We force our clients to be organized and keep at any time one connection to any node, to avoid interfering too much with the normal network behaviour. Malicious entities could easily have more resources and/or focus on specific IP addresses, using different addresses to maintain multiple connections with other nodes: the chances of assigning transactions originating from the connected nodes can then drastically increase.

6.3 Analyser

The analysis of the data logged by Bitshark is done on a separate server (**5.1.6**), which contains a copy of the relevant database information synchronized with the middleware and has all its computational resources available to run the needed analysis software (CPU an memory intensive).

Analyses are performed on specific data registered in a certain time window by the Bitshark clients, the objective is to estimate the size of the "hiding set" for a group of randomly sampled Bitcoin addresses taken from an initial time period and see how it evolves over time considering more information.

The hiding set represents the pseudonymity level of the system: an address *A* can "hide" (only) between all the addresses in its set, meaning we cannot say which other addresses in the set are unrelated to *A*. Ideally for any address this set should be equal to all the addresses being observed in the environment during the measurements, meaning we can't infer anything about all other addresses; in practice we observe it's much lower than that, signifying a pseudonym system with privacy risks.

While many other works about Bitcoin worried about merging groups of addresses together and displaying the relations between them **[Rei11] [And12] [Mei13] [Obe13] [Ron13]**, we (in addition to that) try to combine information about related addresses with newly found data about unrelated addresses. This kind of approach allows us to estimate privacy in the Bitcoin system, measured as the amount of addresses available to hide into, obtained by eliminating from all the observed addresses (the ideal hiding set) all the ones which we are sure are unrelated to the considered address.

6.3.1 Software Description

The analysis program and its utilized heuristics are the main part of this thesis, as previously mentioned its purpose is to go through transaction and entity data logged by the Bitshark workers and estimate hiding set sizes for a random sample of Bitcoin addresses. The hiding set measurements are done at different intervals during a certain chosen time window, to show how this information changes over time as we consider more addresses and newly found related or unrelated data about them.

While the software doesn't directly deal with data stored in the blockchain, Bitshark thanks to its connectivity will implicitly log everything (transaction-related) that would be contained in the chain in the considered time period, with the addition of other transactions and thus addresses that could be lost if only the data stored in the blockchain would be observed.

An experiment performed through the Analyser software begins with the definition of the time window being considered,[6] all the transactions and their addresses logged between the limits will be inspected by the utilized heuristics and (if possible) assigned to an entity. In the initial phase of the program a random sample of addresses is chosen: the Analyser will estimate the hiding set size for each of them. Going forward in time, an increasing amount of data is considered at every step, incrementing the amount of information about related or unrelated addresses being found.

[6] In our performed experiments the time period is of one or two weeks.

Each information (heuristic) we use creates "related" or "unrelated" links between addresses depending on the given data (**6.3.2**). To obtain a final estimation (at every step) for the hiding set size, all the different generated links are considered and combined as explained in **6.3.3**, the set size is then given by the total number of addresses observed so far minus the known unrelated addresses for the considered sample.

It has to be noted that for the majority of the addresses the hiding set cannot be reduced by a considerable amount, our capabilities of doing so by inferring their unrelated addresses depend directly on being able to assign a transaction, where they first are used as inputs, to an entity, or perform the same operation on one ore more related addresses.

The sampled addresses are chosen at random among the ones we know are assigned to at least a transaction in the initial period of the time window being considered, the results are then shown for those addresses. In our results we will present the percentage of transactions we manage to assign; this value can then also be directly used to know the percentage of total addresses for which our hiding set measurements are valid: considering the fact that the assignments only depends on network factors and not by what's included in transactions, the number of addresses for which the results apply is given by the same percentage.

It should be mentioned again that the percentage of transactions we manage to assign highly depends on our connection behaviour. We behave "nicely" with other Bitshark clients, maintaining overall a single connection to each peer, thus obtaining the assignment percentages mentioned in the previous chapter. The obtained results for the hiding set sizes and thus the privacy (pseudonymity) level provided by Bitcoin can then be seen as higher, optimistic bounds. More aggressive and malicious entities are free to maintain multiple connections with any wanted node; this would result in increased transaction assignments, less false positives and more information being available to analyse, leading then to reduced hiding set sizes.

6.3.2 Utilized Information

There are three possible type of relations between any couple of Bitcoin addresses: unknown, related, unrelated. The Analyser software uses multiple heuristics, combined as described in the next section, in order to better decide which is the most probable situation. These heuristics consider different known facts and properties of transactions and users of the network, in order to give information about addresses depending on the given information. We use three heuristics that can tell when two addresses are related, and another three, depending on the assignment of transaction to entities, that can give new important information about unrelated addresses.

Related: Common Transaction

This heuristic was originally proposed in the first known paper that tried to associate Bitcoin addresses together in different entities **[And12]**, there presented as "Heuristic I". This simple yet very strong heuristic assumes that, if multiple addresses appear together as inputs in a valid transaction, then said addresses are related to each other (belong to the same entity), as the user signing the transaction must posses private keys for all the used addresses. Even though it can happen due to special transaction mixing (**3.5**) that we falsely associate addresses, this happens extremely rarely.

Addresses used as inputs in the same transaction are related.

Related: Common Bloom Filter

Bloom filters are used by SPV clients (**3.3**) to indicate the transactions they are interested in and would like to receive, this is usually done by inserting in the filter the addresses owned by the user of the client. Bloom filters are a container with possible false positives when checking for contained elements; users can choose different levels of privacy by setting higher false positive chances, at the cost of more uninteresting transactions received. However most of the default clients use very low false positive rates, making it possible to infer, through an analysis of the amount of data estimated to fit in the filter, if matched addresses are related together. To estimate the number of elements in a filter we use the formula derived from **[Swa07]**:

$$-s \cdot \frac{ln\,(1 - b/s)}{h}$$

Where s is the size of the filter in bits, b are the bits set to one, and h is the number of times the hash function is applied when inserting elements in the filter. The filter is deemed usable if the result is below a certain threshold.

Addresses in the same bloom filter with a low enough false positive rate are related.

Related: Common Entity

One of the innovations presented in this work is the possibility of assigning transactions to entities: nodes connected to the Bitcoin network (and to us). An entity is by definition a user of the system utilizing a client to send and receive messages in the network, we can then infer that if multiple transactions are assigned to the same entity, then the addresses contained in any of the transactions will be related to the addresses contained in all the other transactions assigned to said entity. This assignment makes it possible to have new heuristics based on entity characteristics and allowing to find unrelated addresses, as presented in the next paragraphs.

Input addresses in transactions assigned to the same entity are related.

Unrelated: Entity Online Periods

We can be very confident that a user utilizing Bitcoin doesn't use multiple devices (and thus nodes) to access the network at the same time. If we observe any two entities that share some online time (by being both connected to our system), then we can assert that the two entities are different and unrelated. All addresses used as inputs in the transaction(s) assigned to the first "conflicting" entity are unrelated to the ones of the second entity. As the Bitshark system logs the online times of all the connected entities, it is possible to build a kind of timetable allowing to find these conflicts.

Input addresses in transactions assigned to an entity which has overlapping online time with another entity are unrelated to the latter's addresses.

Unrelated: Country Origins

We believe that in the majority of cases Bitcoin users stay inside the same country while using their money. Of course there are known exceptions and people travel to different places, especially if we consider long periods of time, depending on the considered time period this heuristic could be more or less accurate. If we observe two entities, with their assigned transactions and addresses, from two different countries, then we can assume their addresses are unrelated. Thanks to the IP addresses logged by Bitshark we are able to assign a country reliably to almost all the encountered entities.

Input addresses in transactions assigned to an entity connected from a country different from the one of another entity are unrelated to the latter's addresses.

Unrelated: Client Versions

Users of the Bitcoin system can use multiple, completely different clients; however very commonly there's a certain behaviour that can be observed: the same user will run, over time, the same or a newer version of the same client. We can then infer that if we observe a transaction that gets assigned to an entity which has a certain client version, and later on another transaction is assigned to another client with the same client but an older version, then it is very likely they are different and unrelated entities. As every node connecting to each other in the Bitcoin network must advertise its personal information through a *version* message before being able to send or receive anything else, by logging the given client name and version through Bitshark we know most of the clients used by all connected entities.

Input addresses in transactions assigned to an entity using a certain client with a given version number newer than the one used by another entity connecting later in time are unrelated to the latter's addresses.

6.3.3 Information Combination

We now have this group of heuristics that can be used to give information about what kind of link there is between any two addresses, but how can this different (possibly conflicting) data be combined and merged? We wanted to build a system that could also be easily used for future new heuristics: they should be easy to add and integrate to possibly improve results or correct mistakes done by other heuristics.

The analysis software uses what we call a "double matrix" approach, where two (triangular) matrices[7] are used to represent respectively the *related* and *unrelated* links. The columns and rows of the matrices are all the addressed observed in the currently considered time window, the value found at the intersection of two different addresses gives the value of the related/unrelated link, depending on the observed matrix; like in an adjacency matrix. We utilize a value system where zero stands for an unknown link (matrices are initialized to zero), negative values mean unrelated addresses, while positive values stay for related addresses, larger positive or negative numbers signify a higher confidence in the link information.

Each of the heuristics has a different weight assigned to it, positive for related addresses and negative for unrelated addresses, the distance from zero increases with the confidence we have in the heuristic's results. The first three heuristics, dealing with related addresses, will only interact with the first matrix, while the other three heuristics, dealing with unrelated addresses, only with the second matrix. Each time one of the heuristics needs to add a single piece of information, address A and B are related/unrelated, the respective entry in one of the matrices is increased by the fixed weight assigned to the heuristic. The first matrix will then only have values in the range $[0, \infty[$ while the second one in the range $]-\infty, 0]$.

The weights assigned to our used heuristics are based on rough estimations of how effective and precise they are at recognizing related/unrelated links; experiments aimed at finding the ideal values based on large real-world data would be interesting to perform in the future. The following list gives the weight numbers used in our experiments presented later on.

- Common Transaction: 12
- Common Bloom Filter: 8
- Common Entity: 8
- Entity Online Periods: -8
- Country Origins: -5
- Client Versions: -8

[7] In this document the "matrix" structure is used to more easily illustrate the different concepts; in the code adjacency lists are used for memory and performance reasons.

As previously mentioned we start with the two structures initialized to zero, as we get more and more information the heuristics will modify the values stored in the matrices, storing what we call the “direct” related/unrelated links between addresses. Before being able to draw a conclusion on which addresses are unrelated to any address A and thus find its hiding set size we must find “indirect” links combining data from both the matrices. This is a computationally intensive procedure which is only done for the addresses we are interested in during our analysis. The estimated type of link between A and any other address B will then be given by:

$$\gamma \cdot M1(A,B) + \delta \cdot M2(A,B)$$

With γ and δ being fixed values chosen depending on the utilized heuristics and their weights. In our experiments, 1 is used for both of them.

Indirectly Related and Unrelated Addresses

To begin with, the indirectly related addresses of A are found; this is done by doing a Depth-first search[8] starting from A's known related addresses: we recursively go through (related) neighbours and store any newly found address which is indirectly related to A. For every address C ($\neq A$) related to a previously explored address B ($\neq A$) we will modify the first matrix as:

$$M1(A,C) = max\,(M1(A,C), min\,(M1(A,B), M1(B,C)))$$

Meaning the new indirectly related address link has a strength given by the minimum between the two known links A-B and B-C; we will then keep “the strongest” known address relation between A and C.

There are two ways for an address D to be indirectly unrelated to A: D is unrelated to an address indirectly related to A, or D is related to an address directly/indirectly unrelated to A. We must consider both cases (plus the already known directly unrelated addresses already present in the second matrix) to find the total unrelated addresses of A.

For the first case, once we found A's indirectly related addresses we will go through all of them and get their unrelated addresses, for each address B indirectly related to A and for each address C unrelated to B we will set:

$$M2(A,C) = min\,(M2(A,C), f(A,B,C))$$

Where $f(X,Y,Z)$ is a function accepting three addresses, with X related to Y and Y unrelated to Z. The function will return a value for the unrelated link between X and Z which will be in the range $[0, M2(Y,Z)]$, proportionally to $M1(X,Y)$; meaning the stronger the relation between node X and Y is, the

[8] The DFS graph traversal algorithm: **http://en.wikipedia.org/wiki/Depth-first_search**.

higher the value of the unrelated link between X and Z will be, based on the unrelated link between Y and Z. We use the following function:

$$f(X,Y,Z) = \frac{M2(Y,Z)}{max\left(1, \frac{\alpha - M1(X,Y)}{\beta}\right)}$$

Where α and β are fixed values that need to be set depending on the amount of used heuristics and the chosen heuristic weights. In our experiments, the constant α is set to 16, while β has a value of 2.

For the second case we iterate through A's unrelated addresses, these include the indirect ones in the first case. For each unrelated address B we find its indirectly related addresses as described in the initial paragraphs, for every address C related to B we then can set:

$$M2(A,C) = min\left(M2(A,C), f(C,B,A)\right)$$

We utilize the same previously described function to obtain an unrelated link between A and C, in the range $[0, M2(A,B)]$ and proportionally to $M1(B,C)$.

Once these steps for finding the indirectly related and unrelated addresses for an address A are completed, its total amount of unrelated addresses can be obtained by merging the known information and checking the link with any other address B through the previously given formula:

$$\gamma \cdot M1(A,B) + \delta \cdot M2(A,B)$$

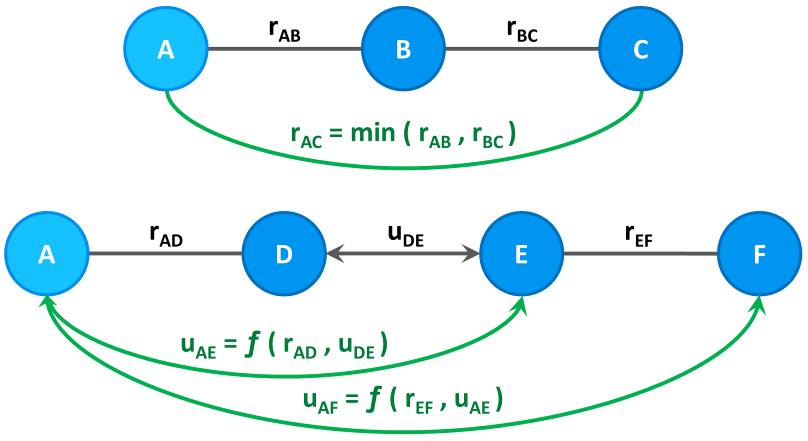

A
B
C
r_{AB}
r_{BC}
$r_{AC} = min (r_{AB} , r_{BC})$
A
D
E
F
r_{AD}
u_{DE}
r_{EF}
$u_{AE} = f (r_{AD} , u_{DE})$
$u_{AF} = f (r_{EF} , u_{AE})$

Figure 6.4: Possibilities for indirectly related and unrelated addresses for address A.

Figure **6.4** shows the previously explained possibilities for indirectly related (top) and unrelated (bottom) addresses. Black lines show the already known directly related and unrelated addresses present in the matrices, green lines show the newly found indirect ones with their relative formulas, given in the same order as they were presented in the previous paragraphs.

6.3.4 Program Flow and Structure

We will now describe how the Analyser software works step by step, what are its inputs and outputs and what are the different parts responsible for the resulting hiding set estimation.

The program is set to automatically connect to the local MongoDB instance to retrieve Bitshark's logged data, various parameters are coded inside the program and cannot be chosen at runtime, but this can of course easily be changed in the future. As arguments the Analyser takes first of all the time window being considered for the experiment, secondly the initial time window in which the sampled addresses are chosen, thirdly the size of the steps to perform until the end of the experiment is reached. At every step the software will add and consider more data as it progresses in time; the hiding set sizes for all sampled addresses are calculated and logged for each step until the end of the experiment is reached.

The steps' size is defined in terms of transactions successfully assigned, as the information added about related/unrelated addresses greatly varies over time. Using this type of measurement unit allows us to roughly estimate the direct and indirect information getting added to the matrices and to get steps of equal "importance", performing the repeated computationally intensive operations only when needed.

The Analyser keeps track of resource usage and statistics about data and heuristics for every step; as more and more information is considered to calculate hiding set sizes, time and memory consumptions increase. As we will see in the next chapter, on average after an analysis of less than a couple of days worth of data the system quickly runs out of memory, and experiences a very fast increase in the computation time needed at every step, following a steep exponential growth. To avoid this and allow us to analyse longer periods of data the program is set up to regularly purge (tentatively) unneeded data, leading to slightly worse result and a loss of precision, but making memory and time stay under control for experiments dealing with data logged for long periods.

At the end of an experiment the Analyser will have provided a log containing statistics about the experiment run; containing for each step the encountered transactions, transaction assignments, entities, entries added to the matrices by the different heuristics, and resources used. For every step a different result file is generated, inside it one can find the hiding set sizes for each of the sampled addresses.

Pseudocode

We will not describe in more detail what the Analyser does during its execution, by providing a simplified pseudocode of the whole program and an explanation of its most important functions being used.

The overall structure of the program can be resumed as follows:

```
while(!experiment_end)
   while(!enough_assigned_transactions)
      extract_data_from_db()
   if(first_step)
      select_sampled_addresses()
   store_direct_info()
   find_indirect_info()
   store_permanent_data()
   purge_unneeded_data()
   save_hiding_set_size()
```

We will now describe the objectives and results of the different functions:

- **extract_data_from_db**: Queries the database for the next potentially assignable transaction in order of time, checks to see if the transaction can be assigned to the first originating entity, stores information about addresses, transaction and entities in local collections used later on.
- **select_sampled_addresses**: Randomly selects the addresses that will be sampled for the rest of the experiment, choosing between the ones contained in all assigned transactions up until now.
- **store_direct_info**: Goes through the previously populated collections with the utilized heuristics; the generated "direct" information is stored in the two matrices for related and unrelated addresses.
- **find_indirect_info**: Performs the operations described in section **6.3.3**, going through the entries in the matrices for the sampled addresses and finding all the indirectly related and unrelated addresses. This function is the main reason behind the high time and memory costs.
- **store_permanent_data**: Stores data about unrelated addresses in a special collection which will be never modified by the memory purges. Each sampled address will have a list of addresses which we, at the moment, think are unrelated; this can be modified over time.
- **purge_unneeded_data**: The matrix pair is always reset; when a memory purge is needed, we also remove from the local collections any data not related to the sampled addresses or that was logged a certain amount of time before the current point in time of the experiment (and not part of the information stored during the first step).
- **save_hiding_set_size**: By looking at the information stored in the permanent collection, save in a new file the measured hiding set sizes for the sampled addresses at the current time of the experiment (step).

The next chapter will go into more details about some of the utilized parameters, as well as present the results of the Analyser in different situations.

Chapter 7

Experiments

In this chapter practical results of the Analyser software will be presented: the program has been given multiple sets of data coming from different time periods where the Bitshark system has been left running, logging the data coming from the Bitcoin network interaction.

In section **7.1** the set-up of the presented experiments is described, explaining which data and time periods are used and why, plus some statistics about the encountered connections with other peers; in section **7.2** the actual analysis results are presented, together with considerations and reports about the software and the performance of the analysis process.

7.1 Set-up and Connections

The Bitshark system is capable of running continuously, logging more and more data while the Analyser software works in parallel, analysing data in real time or concentrating on a previously registered time period. We present a series of experiments that give results for a time period of one week of data; multiple unrelated time periods have been scrutinized, to show how the results are similar and repeatable in different scenarios. In order to have comparable conditions for all the experiments, Bitshark has been manually restarted each time; the logged data which is then considered for the experiments is taken once a considerable number of connections has been established, after a small initial build-up period of a couple of hours.

Chart **7.1** shows the number of connections that can typically be observed after the Bitshark system is started, the data being analysed during the presented experiments begins 2-3 hours after the initial boot.

The outgoing connections rise very quickly to an almost constant value; this is due to the list of IPs being given by the middleware to the workers, which rapidly try to connect to all old and new addresses. The incoming connections increase slowly over time, as our addresses are spread and discovered in the Bitcoin network and more new clients connect to us when needed. The large daily oscillations are due to the number of users utilizing Bitcoin: we can see the effect of a global system and the influence of the time of the day. The small 2-hours peaks in the outgoing connections during the initial phases are caused by the frequency at which workers retry previously disconnected clients. It can be observed how on average during the whole experiment we have 7'000 outgoing and 2'000 incoming connections.

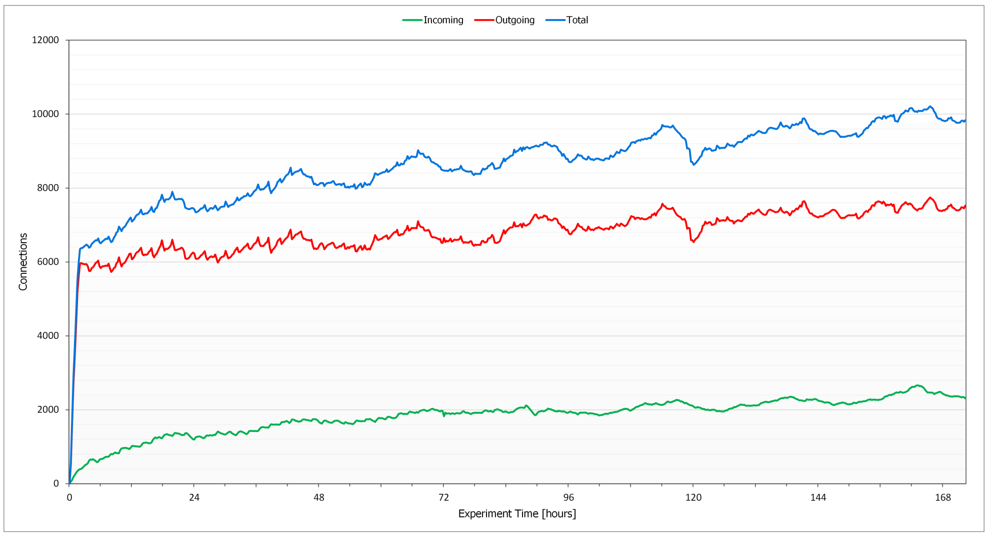


Figure 7.1: Typical connection behaviour over time of the Bitshark system.

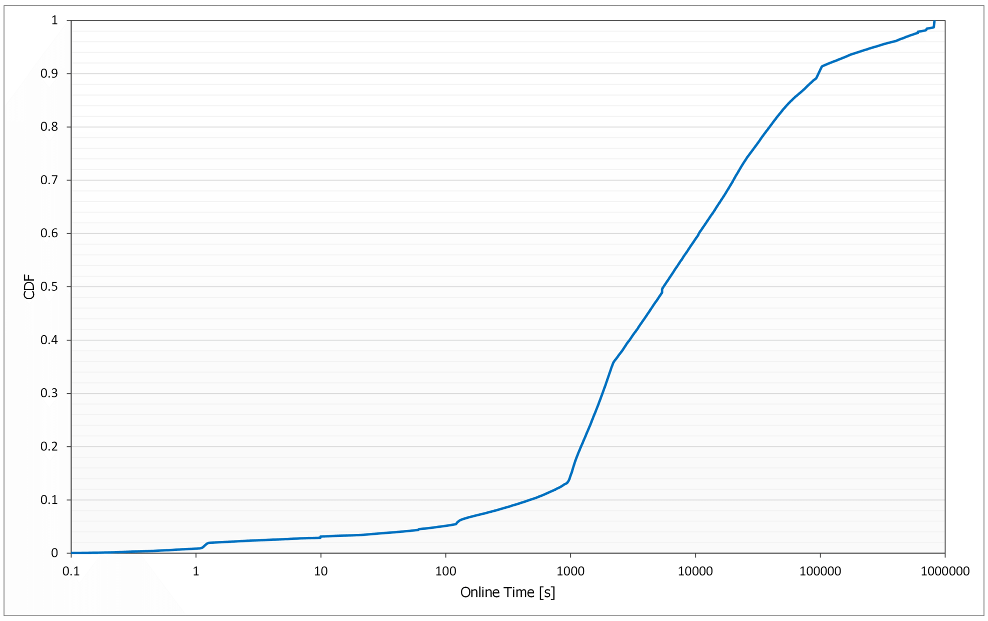


Figure 7.2: Online time distribution for nodes connected to Bitshark (excluding bots).

As seen in section **5.2** the Bitshark system connects to several different client types, with the majority of them being automated bots, not relaying any transaction and only remaining connected for a very short amount of time. Chart **7.2** shows the measured time "normal" nodes remained connected to us during the various experiments. As both us and the remote clients don't usually disconnect in normal situations (no one is misbehaving) once the connection is established, the measured times closely correspond to the time nodes were really online in the network. We can appreciate in general a long online permanence of many clients; the more they remain online, the more information can be extrapolated through our heuristics for the analyses.

7.2 Results and Observations

We will now present the result of our Analyser software, looking at three different windows of data of one week. For each of them a random group of addresses is decided to be sampled as explained in section **6.3.4**. By looking at an increasing amount of data logged as time passed we then, step after step, build a graph of related and unrelated addresses; in order to finally obtain the hiding set size, representing the pseudonymity (privacy) level.

7.2.1 Resource Usage

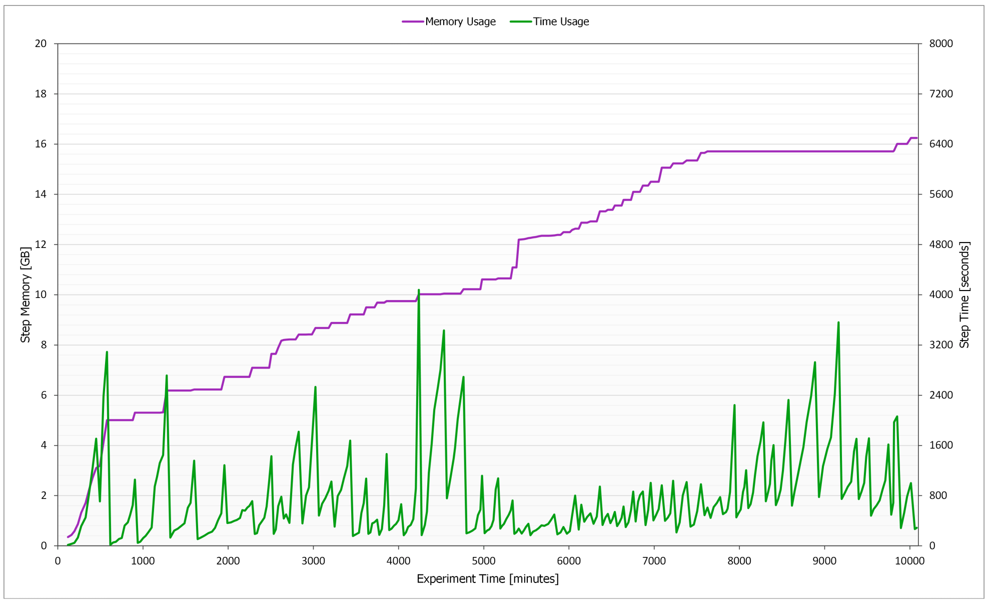


Figure 7.3: Experiment 1 – Time and memory (RAM) usage of the Analyser software.

We begin by showing in charts **7.3**, **7.4** and **7.5** the memory and time resources utilized by the Analyser software over the course of the three experiments. In all cases it can be observed how the time needed for each step rapidly increases; this is due to the logical exponential increase in related and unrelated addresses to be found, as more information is considered. The variation between experiments is given by the fact that, depending on what was logged, less or more information can be inferred about the sampled addresses, causing a chain reaction affecting the current and subsequent resources needed for the analysis of the data.

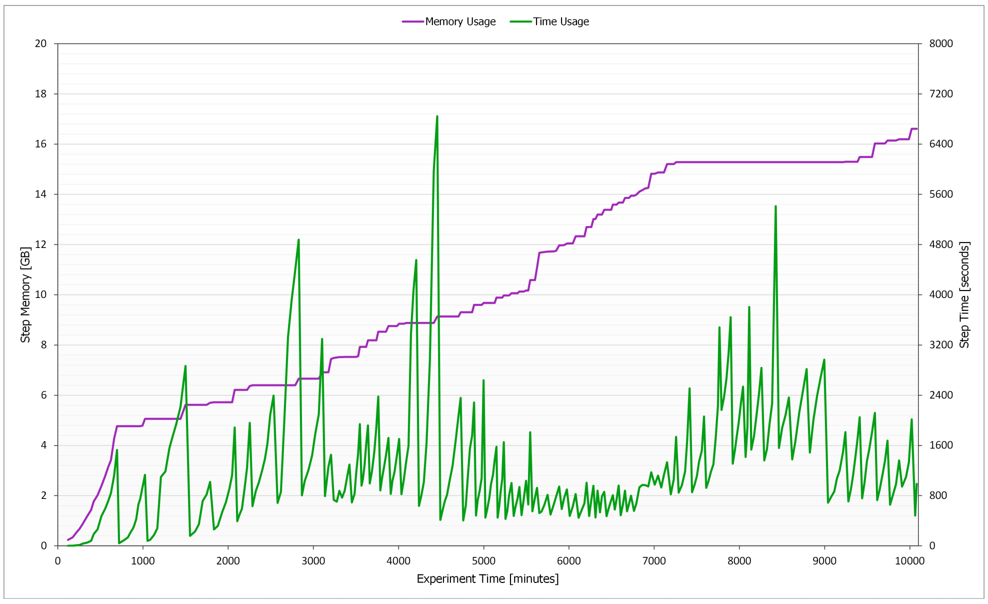


Figure 7.4: Experiment 2 – Time and memory (RAM) usage of the Analyser software.

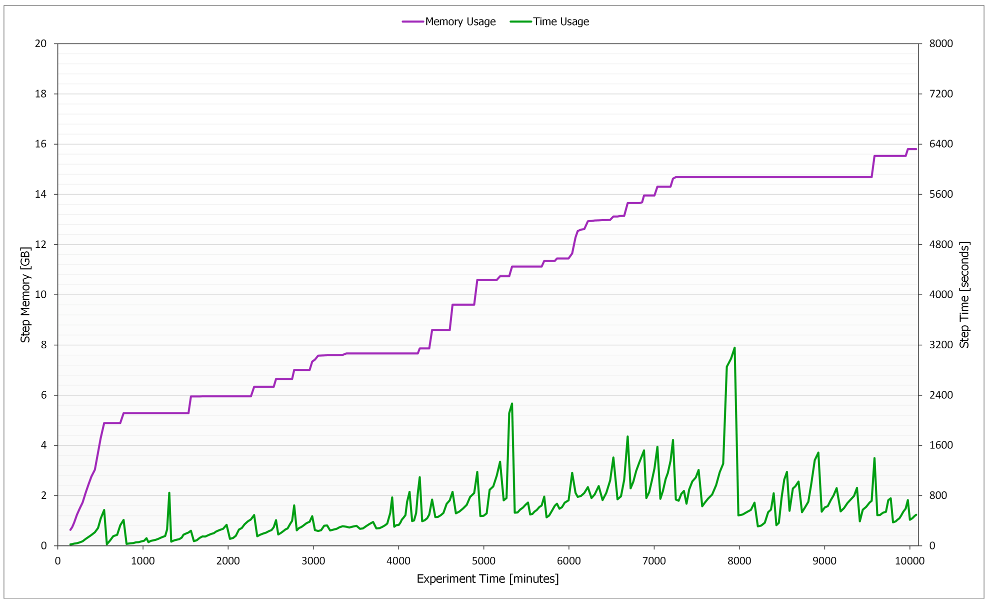


Figure 7.5: Experiment 3 – Time and memory (RAM) usage of the Analyser software.

The spikes are caused by the memory purges done by the program: the time keeps going up as we get more data until some is erased, after that the usage goes down, to then rise again in cycles. Our software has not been particularly optimized, we don't exclude that improvements could be done about its computational performance. Memory can be seen to rapidly increase, then going up following a terraced behaviour. This is again due to memory purges, which causes part of the considered data to be deleted; the OS for performance reasons leaves the memory still assigned to the program, to be then filled again over time. We believe that memory usage increases so fast due to memory fragmentation, as the actual occupied memory should theoretically be less (although still increasing over time), further measurements and optimizations should be made in the future for longer analysis periods.

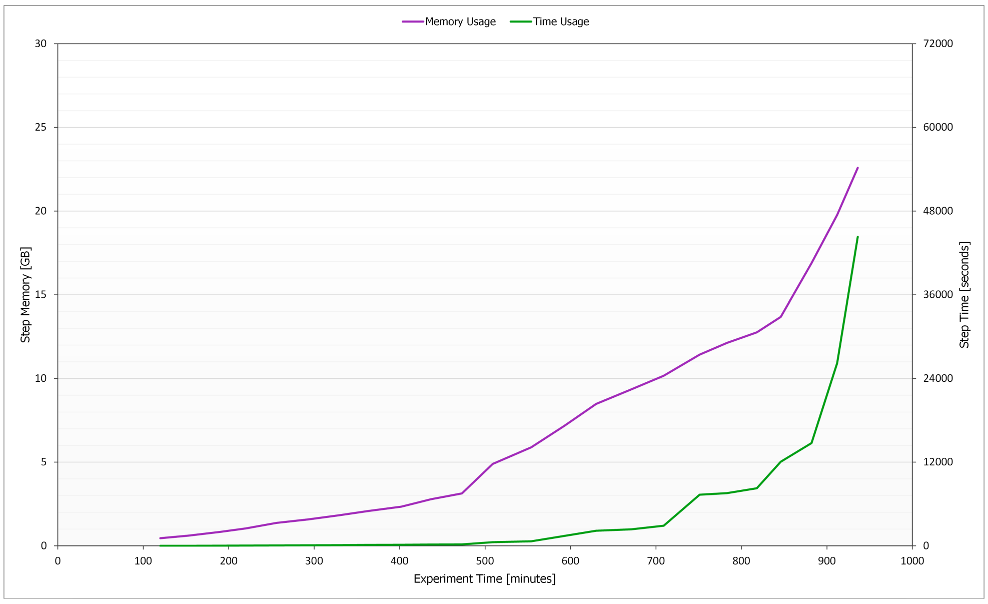


Figure 7.6: No data purging – Time and memory (RAM) usage of the Analyser software.

Initially the Analyser didn't purge any data at all; however, after the execution however of analyses considering longer and longer periods of time, it became clear that, unless a vast amount of resources and optimizations were found, compromises had to be made. Chart **7.6** shows the resource usage of the software if run without any kind of limit and never discarding any type of information; note that after less than one day worth of data memory usage surpasses the available RAM on our machine (32 GB) and causes the program to crash, while the time needed for each step also reaches high values in an exponential way. Performing data purges allows us to keep both time and memory needed under control, while obtaining results that differ less than 2% in our measurement tests performed on a time-limited amount of information where no data is erased over time.

7.2.2 Heuristics' Contribution

During the analysis process we recorded the amount of links (related or unrelated addresses) that have been affected by the different heuristics over time. It is difficult to say if one is more important than the others due to the synergy that can be found between them: even though some bring larger numbers of relations, others can be as effective by forming few but very important links. Nevertheless we think that it's important to know at least the raw numeric contribution that each of the different heuristics provides and how it helps to arrive to our final results.

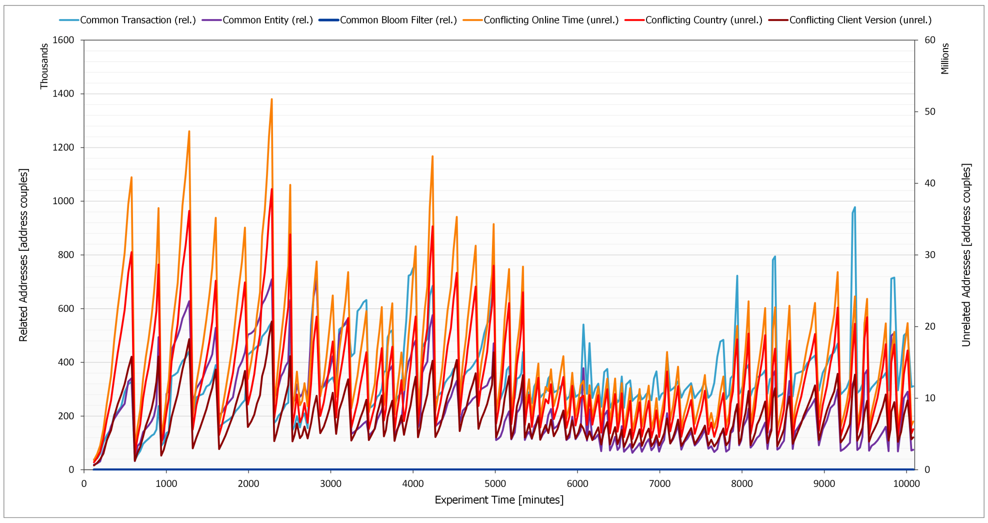


Figure 7.7: Experiment 1 – Heuristic contribution in terms of links (address couples).

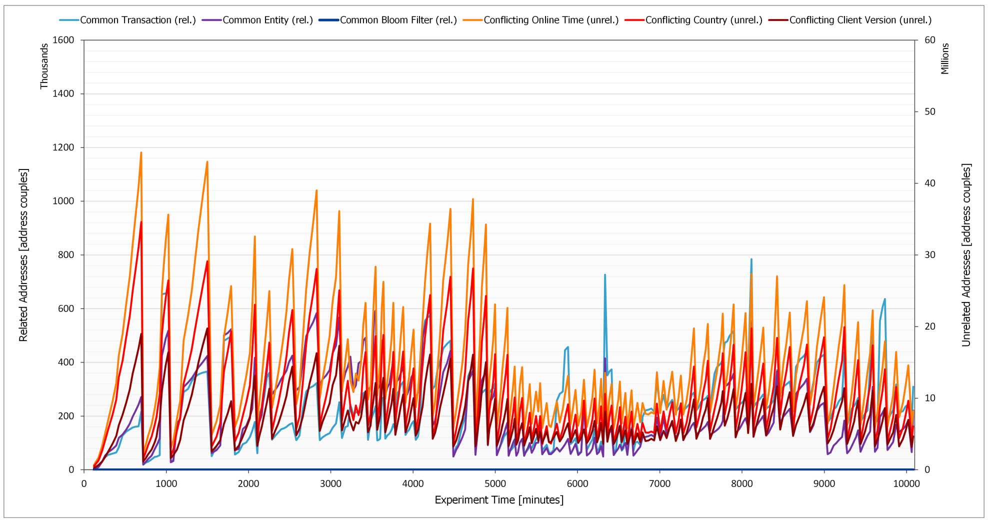


Figure 7.8: Experiment 2 – Heuristic contribution in terms of links (address couples).

Charts **7.7**, **7.8** and **7.9** show the contribution of the six utilized heuristics as number of address couples (links, related or unrelated addresses), the three heuristics for related addresses are shown in the blue hue, while the three for unrelated addresses in the red hue. In all three experiments we can again observe the spike behaviour given by the accumulations of information followed by data purges. Note that the given related and unrelated values given are only the "direct" links (**45**); many more links are then added by finding and storing all the "indirect" relations.

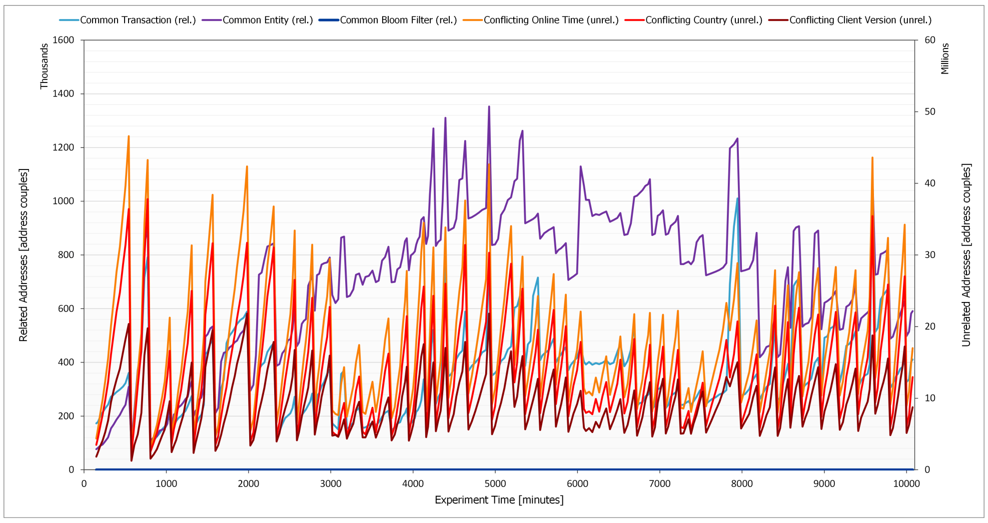


Figure 7.9: Experiment 3 – Heuristic contribution in terms of links (address couples).

The bloom filter heuristic doesn't seem to contribute many new relations to the analysed data as the chart shows; despite the very low numbers, its contribution could nonetheless prove itself important for the reasons previously explained. We can note how in the third experiment we encounter and store many related addresses thanks to the "common entity" heuristics. This is most probably due to one or more entities being (temporarily) responsible for a considerable number of transactions, or being identified as sources of fewer transactions with many inputs. As presented in the next section, this doesn't seem to have a great effect on the final results in regards to the measured hiding set size if compared to the other experiments.

To better display the actual contribution comparisons between the different heuristics, charts **7.10**, **7.11** and **7.12** show the percentages of related and unrelated addresses links brought by the two groups of heuristics, utilizing the same data as before. The percentages are calculated independently for the two groups: related and unrelated links.

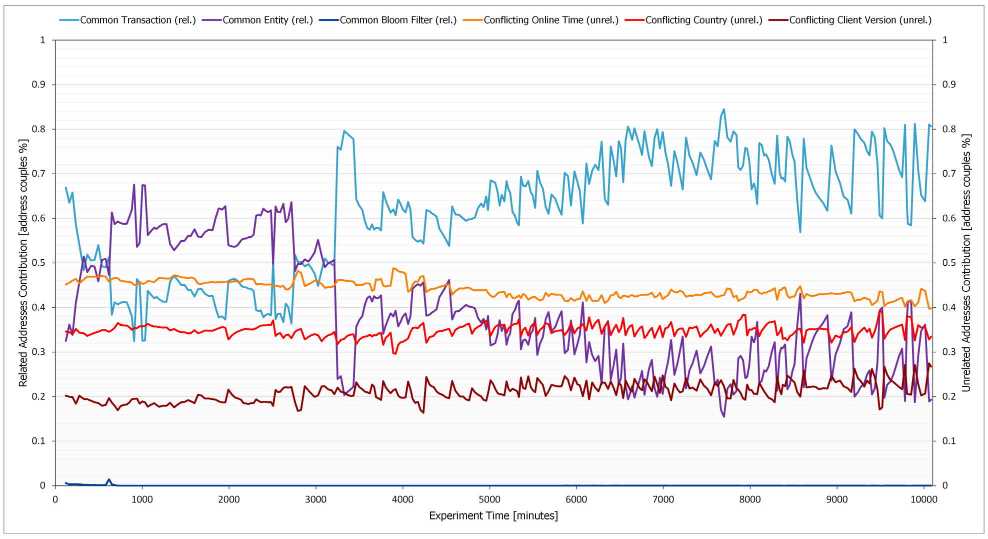


Figure 7.10: Experiment 1 – Heuristic contribution percentages.

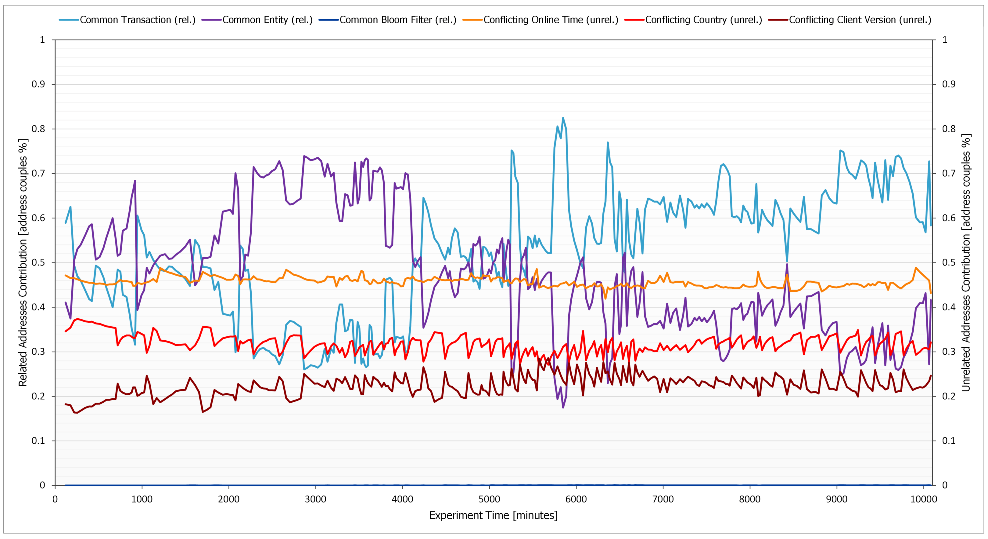


Figure 7.11: Experiment 2 – Heuristic contribution percentages.

We can observe how (especially for the first two experiments) for the related addresses we have an initial greater contribution by the "common entity" heuristic, with the rest almost completely taken by the "common transaction" heuristic; after some time the situation reverses and seems to remain stable. For the third experiment this behaviour appears to be delayed but can slowly be observed as well. The contribution of unrelated addresses is very stable; the results are similar for all our tests and show that the "conflicting online time" heuristic brings the most links, followed in order by "conflicting country" and "conflicting client version". The spikes due to data purging are obviously still visible throughout all the charts.

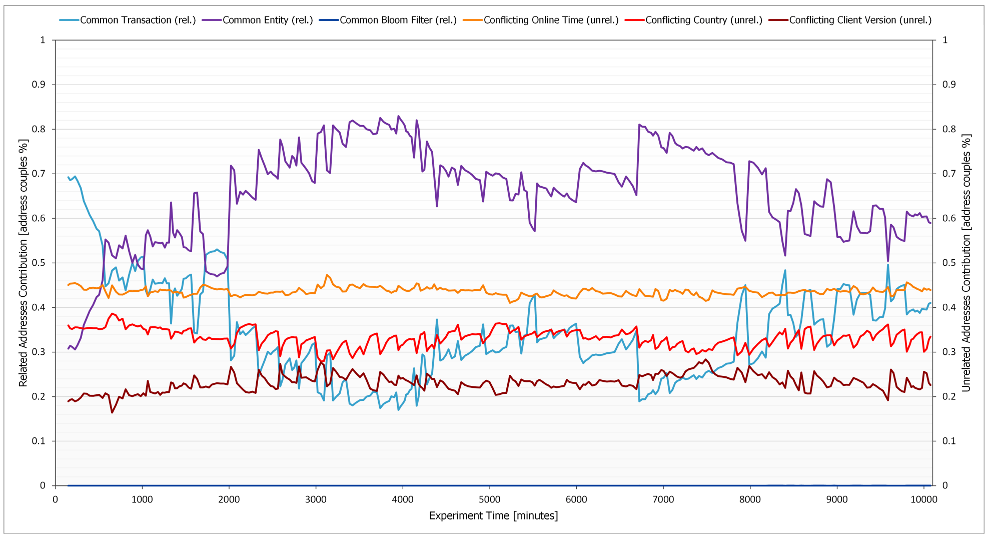


Figure 7.12: Experiment 3 – Heuristic contribution percentages.

7.2.3 Hiding Set Size

We finally present the main results of our work: for all the sampled addresses we show the measured hiding set size; the set, as previously explained, contains all the elements an address can "hide" into, while the other addresses outside the set are found to be unrelated. In order to have good pseudonymity and thus privacy it would be optimal for the set to contain all the observed addresses, this is however not the case with Bitcoin.

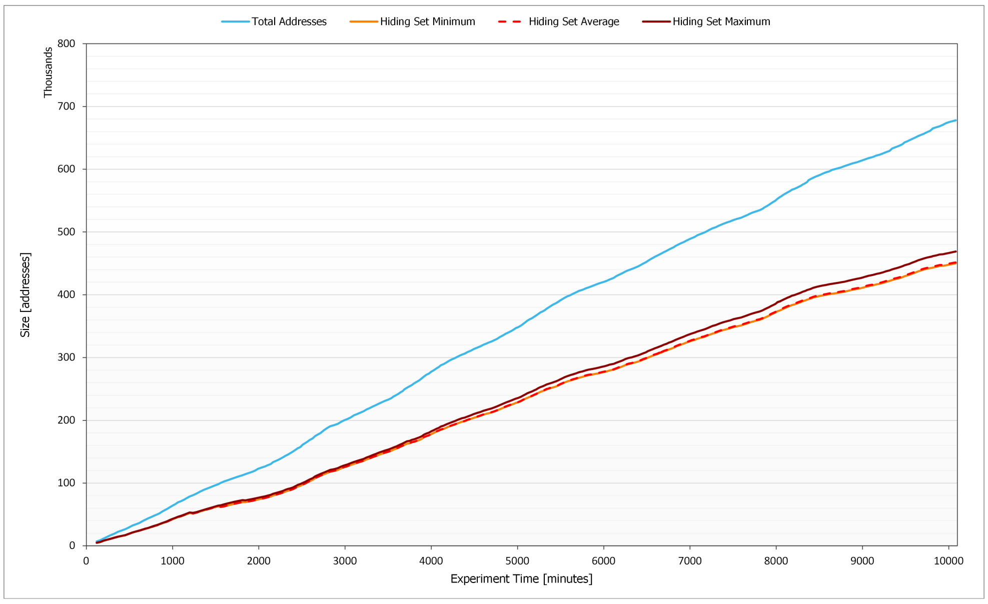


Figure 7.13: Experiment 1 – Hiding set size with observed addresses.

In charts **7.13**, **7.14** and **7.15** it is possible to observe our estimation of the hiding set size step after step for the three experiments, as more data is considered. We present the minimum, maximum and average measurements from all the sample addresses, together with the total amount of addresses that was observed up to that point in time.

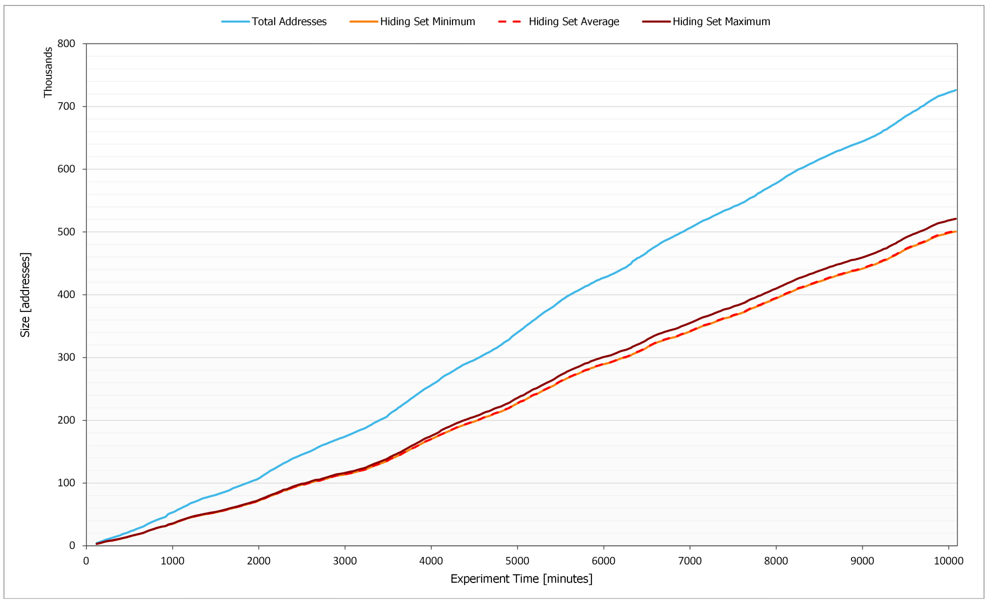


Figure 7.14: Experiment 2 – Hiding set size with observed addresses.

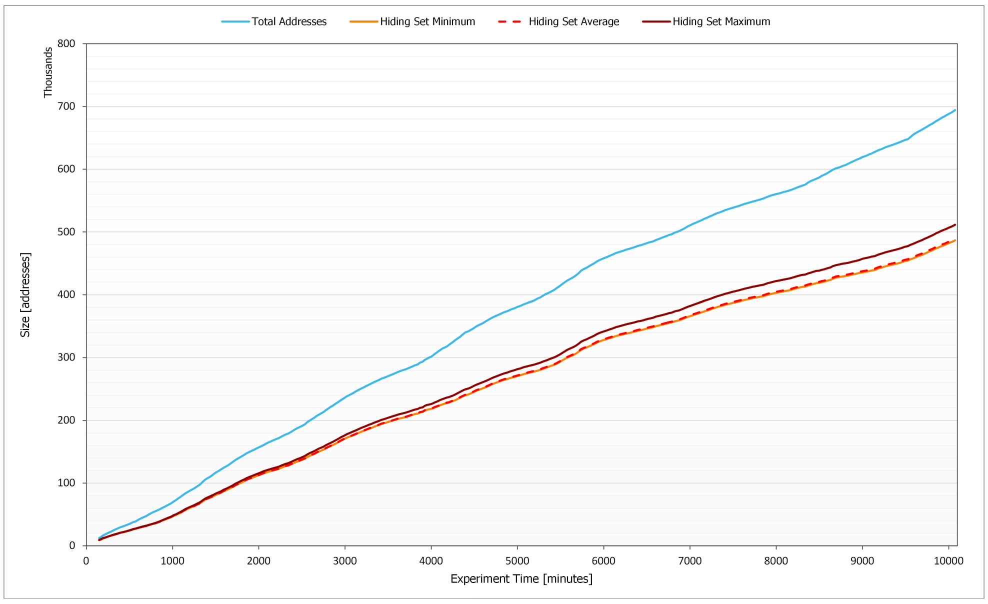


Figure 7.15: Experiment 3 – Hiding set size with observed addresses.

An almost identical behaviour can be seen in all the experiments: a growth in observed addresses together with hiding set estimations following with a similar increase. A gap between the total addresses (optimal privacy value) and the estimated hiding set can be observed and would seem to maintain proportions over time. Hiding set measurements all lie very close between each other, with a small difference between minimum and maximum and the average almost at the same level of the minimum (worst privacy value).

Charts **7.16**, **7.17** and **7.18** show the same data, but with the hiding set size estimation expressed as percentage of the addresses observed up until that point in time; we plot again the minimum and maximum values, together with the average of all values and the standard deviation.

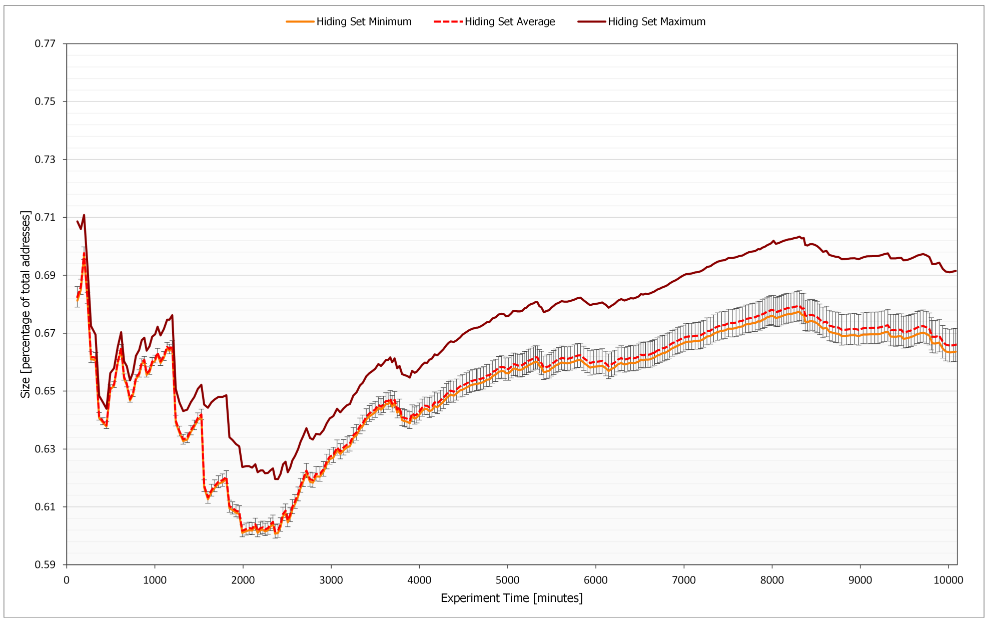


Figure 7.16: Experiment 1 – Hiding set size as percentage of observed addresses.

Again we can observe similar behaviours: in all the experiments we can notice how the hiding set size starts "high" at around 70-75%, we then have a rapid descent during the first twelve hours of data, some fluctuations and then a phase of relative stability that seems to continue in the future with minimums of 65-70% and maximums of 70-75%.

We can see that that a bottom "worse" value is obtained in the initial phases of the week, and that the value ranges start very closely together and then slowly separate over time, reaching a separation of few percentage points by the end of the week. The average always stays really close to the minimum, with the standard deviation slowly rising as the experiment progresses.

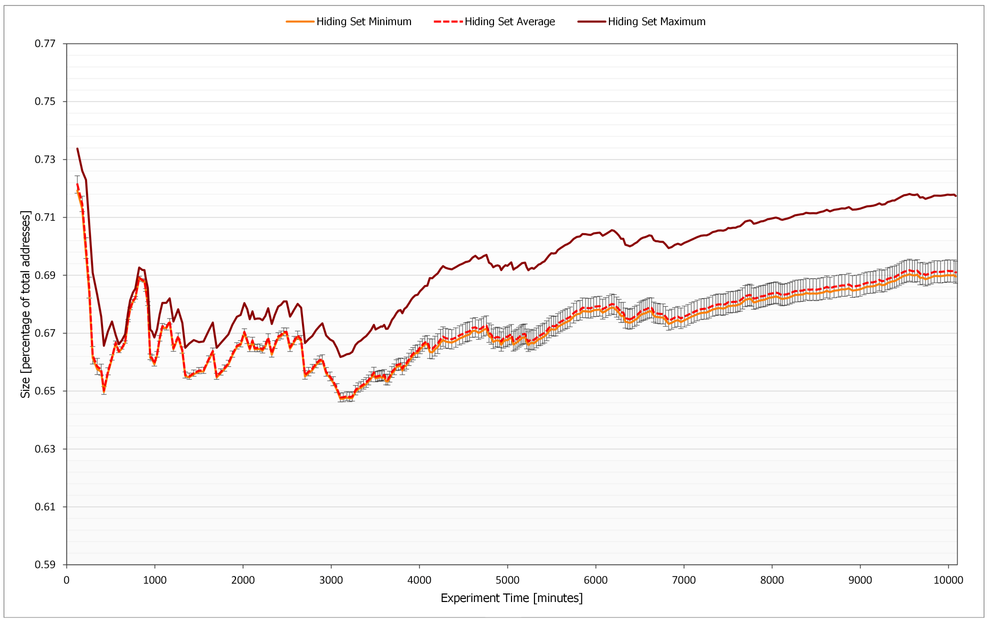


Figure 7.17: Experiment 2 – Hiding set size as percentage of observed addresses.

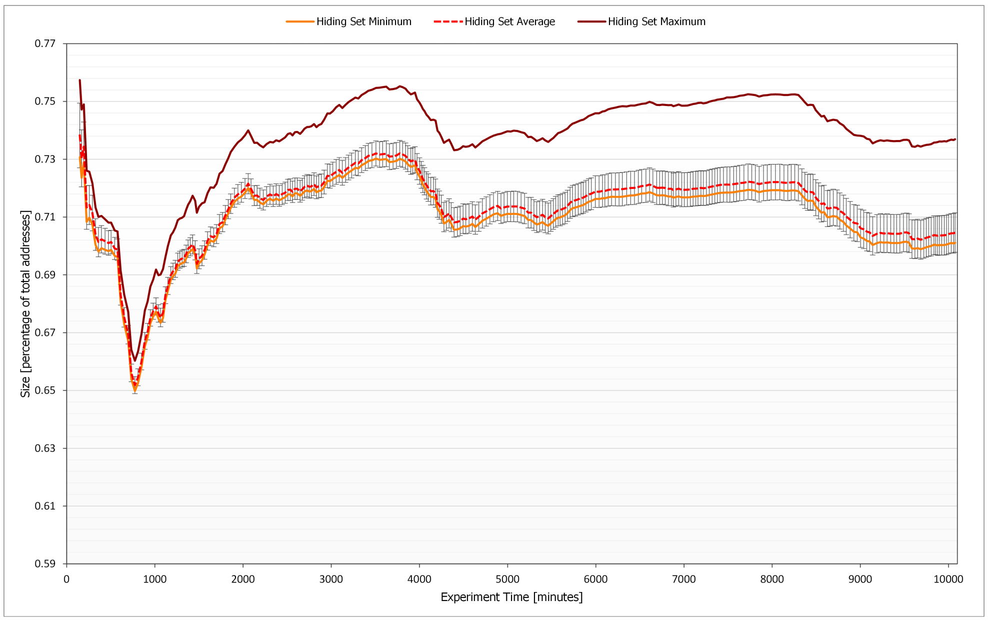


Figure 7.18: Experiment 3 – Hiding set size as percentage of observed addresses.

For testing purposes we tried running the Analyser not with the custom weights for the heuristics explained in section **6.3.3**, but with constant values of 1 for the first three (related addresses) and -1 for the others (unrelated addresses); to see what differences (if any) it would bring to our results. The outcome for the hiding set size can be seen in charts **7.19** and **7.20**, to be compared with the previously shown third experiment results.

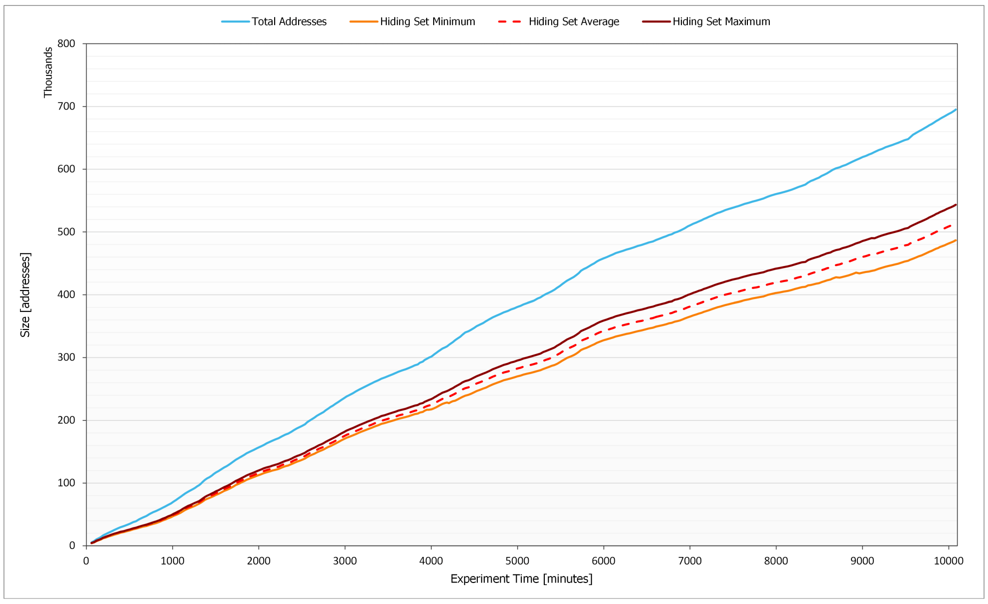


Figure 7.19: Experiment *"same weights"* – Hiding set size with observed addresses.

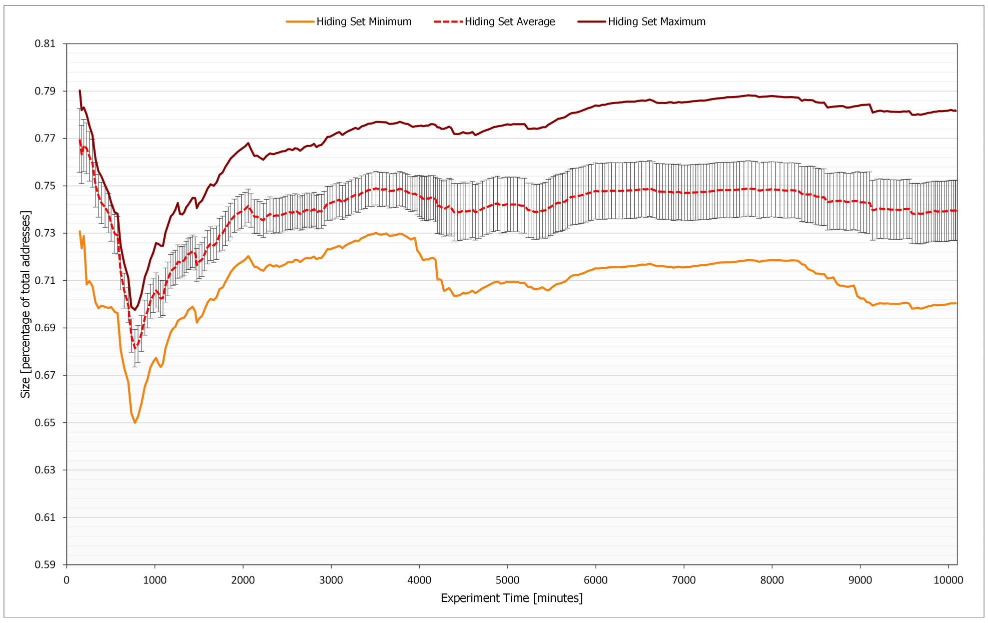


Figure 7.20: Experiment *"same weights"* – Hiding set size as percentage of observed addresses.

We can observe how doing this results in minimum values that are identical to the previous experiment, the difference from the maximum values is however large since the beginning and grows over time, the maximum values also show larger hiding set sizes. The average is now in the middle of the two limits, with a larger standard deviation, showing a different distribution of the hiding set values.

We believe these simple values cause conflicts between related and unrelated addresses, resulting in unknown links; using values for the heuristics that approximate their accuracy is an important part of the analysis process. We can't assert that our chosen values are the best or correct ones, as they were simply picked following general ideas about the capabilities and properties of the heuristics, and not found through calculations or extensive testing. We believe that it will be important in the future to find a way to optimize the weights for any old and new heuristic used in the Analyser.

Returning back to our original experiment results, what do they tell us? On average during all the experiments we were able to assign to an IP slightly over 22% of all the observed transactions, which translates to at least the same percentage of addresses for which our hiding set sizes and conclusions apply, without even considering their related addresses. For these addresses the privacy provided by Bitcoin via the usage of pseudonyms (the addresses themselves) is lacking: by passively listening to the network with limited resources we were able in addition to finding their relations, to discard from 30% to 40% of all observed addresses as "unrelated" depending on the considered time period, thus drastically reducing the amount of addresses they could hide into, diminishing their privacy and making possible a more detailed analysis of their transfers and behaviour.

The analysis that we do on the logged data could be extended in the future by new or improved heuristics and additional ideas to link addresses, thus further reducing pseudonymity and privacy. The technique we use to assign transaction to connected peers can also be used for other purposes, targeting specific clients and revealing precisely the transactions and addresses they are responsible for. We also know that a malicious entity would be free to set up multiple connections with its targets, allowing a much more precise identification of the victims' relations and usage of bitcoins.

Even though Bitcoin doesn't promise anonymity, we believe that work still needs to be done to avoid these analyses and attacks on the privacy of its users. We have shown how, with the current system, it's possible to reduce the pseudonymity level provided by the different addresses for an important part of them, which thus in turn results in worse privacy guarantees.

Chapter 8

Conclusion

We will now conclude this thesis, starting with a series of possible improvements on the Bitcoin protocol regarding privacy (**8.1**), and finishing with considerations about the work that still needs to be done on the subject and the future of research in this field (**8.2**).

Through the logging of data via the Bitshark system we passively registered information about data being exchanged in the Bitcoin network. Via theoretical calculations and practical measurements we have shown how it's possible to assign Bitcoin transactions to real-world entities in different scenarios, affecting the privacy of the system users. We developed and presented a new analysis tool, capable of merging data coming from different heuristics and building detailed information on the relations between Bitcoin addresses (pseudonyms). We used known and new heuristics to find related addresses and we proposed innovative heuristics that bring info about unrelated addresses. By arriving at a series of estimations of the hiding set size, it has been shown how the level of pseudonymity (and thus privacy) provided at the moment by Bitcoin is not very good.

The presented results were obtained through a well-behaving software, Bitshark, which respected common behaviour in the Bitcoin network; from the given calculations it has been shown how theoretically malicious entities could obtain more information from the network, thus further reducing the privacy provided by Bitcoin addresses to its users.

8.1 Bitcoin Improvements Proposals

We have shown through theoretical and practical results how it is possible to assign transactions to connected peers in a decent percentage of cases; thanks to this information it is possible to obtain data about unrelated addresses, thus undermining Bitcoin's potentially full pseudonymity obtained by being able to hide in all the observed addresses.

We believe the current Bitcoin protocol could be improved, in order to decrease the chance of evil entities being able to perform transaction assignments like we did, thus increasing the users' pseudonymity level. In the next paragraphs we will describe two possible modifications, all diminishing assignment probability while at the same time not decreasing the speed at which transactions are eventually transmitted through the network.

The node responsible for the creation of a transaction must of course send an *inv* message to announce and broadcast it: sending it quickly to all connected peers doesn't solve anything and makes the transaction easier to assign, while sending it to a very restricted number of neighbour clients makes the assignment difficult, but at the cost of a slow broadcast of the transaction over the network. Due to the core concept that the Bitcoin network is peer-to-peer and fully decentralized, selecting only "trusted" nodes for the broadcast could certainly not be considered as an improvement.

Our first proposal is related to the connections that a client in the Bitcoin network has with its peers. As it is for now, nodes accept any incoming connection and set up a minimum number of outgoing connections if needed, without any particular selection criteria; this results in links with frequently high delay as seen in previous measurements.

We believe that it would at the same time benefit the network and decrease assignment probability if clients reserve a part of their outgoing connection slots to peers with measured low link delays. All nodes should be connected to a minimum amount of other nodes with low delay; these connections can be selected over time by comparing times of the outgoing connections via a series of Bitcoin pings. This method shouldn't penalize too much nodes with very slow internet connections, due to its application only to (part of) the outgoing connections, and would bring numerous advantages.

The second proposal is aimed at modifying how the current protocol for transaction relay works, in a way related to what has been already proposed for blocks in **[Dec13]**. In the current protocol, before a node *B*, receiving a transaction from another node *A*, can retransmit it to its other neighbours, the series of messages *inv* $\rightarrow$ *getdata* $\rightarrow$ *tx* must happen between the two, causing an important amount of time to pass before information can be retransmitted, depending on the link delay. This procedure is followed due to the fact that *A* can't know if *B* already knows about the transaction, and *B* must verify the validity of the latter by receiving the complete data before retransmitting it to any of its peers.

We propose to modify the protocol by "pipelining" how transactions are propagated through the network: a node *B* always begins broadcasting the *inv* for a transaction immediately after the receipt of the relative *inv* from *A* (in case it's a new and unknown transaction for *B*), the receipt and verification of the full transaction happens after it has already been announced to the peers. As transactions with the new method are not verified any more before being announced to the next peer during the relay, it is possible that attackers could use this to spam the network with *inv* messages for transactions that will never be sent. Note however that a similar attack is already possible with the current system, where an evil entity could simply create an arbitrary number of (conflicting) transactions that get sent throughout the entire network utilizing bandwidth; the limited size of an *inv* message also shouldn't cause much trouble. This proposal would result in much faster propagation throughout the network for all transactions, and a lowered probability of assignment of transactions to nodes.

The presented ideas for improvements on the Bitcoin protocol shouldn't entail any drawback and, in addition to improve the situation regarding transaction assignments to IP addresses, would also help the performance of the Bitcoin network. We feel that the implementation of each proposal, which could be taken alone or combined, would benefit the current system.

8.2 Future Work

Our work has shown that the pseudonymity level provided by Bitcoin addresses is not ideal, we used a newly built analysis tool, looking at data logged in a limited period period of time and building relation information through a series of heuristics. We will not list a series of possible improvements, additions and research that can be done in the future on the subject.

Our analyses and presented results all are done for what can be considered "short" periods of times; this was done so due to time constraints on the work and the necessity for multiple experiments. It would be interesting to observe what would happen to the relations and the hiding set sizes by logging and analysing data for much longer periods.

The developed Analyser software has not been extensively optimized for neither memory nor CPU usage; working on a faster and less memory intensive program would bring better results for longer periods of time, requiring less data purges. For the same objectives it would be desirable to also be able to test the software on more advanced hardware.

The rules for the assignments of transactions at the moment don't "automatically" support SPV nodes: to obtain less (already rare) false positives and be able to assign more transactions, the software could be modified to automatically assign transaction to a certain SPV client if it appeared anywhere in the originator list of a transaction, as these clients never relay transactions.

The Analyser program contains several different parameters related to the used heuristics and how logged data is interpreted for the discovery of related/unrelated links between addresses. We believe that thorough testing for their correctness and optimality, if possible, would lead to more accurate analyses and results that are more reliable than the current estimations.

New heuristics can (easily) be added to the Analyser software, leading to better and more accurate results, further decreasing the obtained hiding set sizes and showing more accurately the actual pseudonymity provided by Bitcoin. It is also possible to find new ways to assign transactions to IP addresses. Two ideas are already known and could help to extend and improve the current system: the "shadow" (change) address heuristic from **[And12]** identifies and relates the change address of a transaction with the latter's inputs, while through the concepts presented in **[Kos14]** it would be possible to create an heuristic to assign transactions to IP addresses when some rare non-standard relay patterns are observed.

The research remains open: new ways to infer information about related and unrelated addresses can always be found and used together with previously known data to reduce the privacy (via pseudonymity) of Bitcoin's users.

Bitcoin has quickly risen to become the most successful cryptocurrency in the world, it's innovative and still in continuous development, it isn't perfect by all means, but it can be improved and built upon **[Bar12]** to obtain a better system. We don't know yet how the economy of the future will develop, what we know is that Bitcoin offers new promising ideas, and work should be focused on its development both in useful features and security; working in this field now means thinking about the (crypto)currency of the future.

Glossary

51% attack – Situation where an entity (group) controls more than 50% of the mining hashing power of the whole network, when this happens this entity effectively controls all the blocks being added to the main blockchain and can then control which transactions are accepted, roll-back transfers of money and allow double spending attacks.

Address – Representation of the public part of an asymmetric key pair, often displayed as an alphanumeric or QR code, it is used in transactions to indicate sources and destinations of bitcoins.

Altcoin – An alternative cryptocurrency to Bitcoin, Altcoins often are a fork of the latter's code base, they offer similar functionalities or additional features and can work on the same or a different network.

Application-Specific Integrated Circuit – An hardware chip developed and built to exclusively execute a limited set of tasks quickly and efficiently.

ASIC – *See "Application-Specific Integrated Circuit".*

Base58 – An human-readable string representing binary data as a sequence of letters and numbers, with the exclusion of certain symbols to ensure visual uniqueness; it is used to represent Bitcoin addresses.

Bitcoin – The original digital currency as described by Satoshi Nakamoto and then worked on by the community, "Bitcoin" (capitalized) can be used to refer to the technology and code, while "bitcoin" (lowercase) is used to indicate a unit of the currency itself.

Bitcoin Days Destroyed – Measure of the Bitcoin transaction volume, calculated for all transactions by multiplying the number of bitcoins transferred by the number of days that passes since they were last spent.

Bitcoin Network – The peer-to-peer network composed of nodes running the Bitcoin code and thus communicating through the known protocol.

Block – Data structure consisting of a header containing the reference to the previous block and a proof of work (plus other information), and a list of the transactions included in the block. Newly mined blocks are broadcasted to the network, appended if valid to the current longest blockchain; they must be referenced to by the next block.

Blockchain – Shared public ledger of all transactions ever done in the network; it consists of a series of valid blocks, each referring to the previous one, containing a list and merkle tree hash of the transactions done in that period of time. Miners and clients always look at the longest block chain and ignore shorter chains (forks).

Blockchain Length – The length (or depth) of the blockchain is calculated as the sum of the difficulties of all the linked blocks in the chain, the main (valid) chain is the one with the highest length.

BTC – *See "Bitcoin".*

Change – Part of the output of a transaction (optional) used to return bitcoins to the sender; since all inputs are completely utilized in a transaction, change is used to handle the frequent cases where what is the intended final payment doesn't exactly correspond to the input sum.

Client – *See "Node".*

Coinbase – Special type of input of a transaction which generates new bitcoins and uses them as input in a transaction; the output(s) of the latter are owned by the miner who discovered the block and who will thus receive them as a reward for their work.

Deflation – Reduction of prices of goods or services over time due to a supply of a product or service growing faster than the supply of a currency, or due to the money amount being finite and decreasing; people will spend less and hoard more money.

Depth – *See "Blockchain Length".*

Difficulty – Number representing the difficulty of finding a new block; it is calculated as the maximum possible target divided by the current target. The target (and thus difficulty) is continuously adjusted after every block to ensure regular mining of blocks over time.

Digital Signature Algorithm – Standard for digital signatures; an entity can create related private/public key pairs: with the private key the entity can sign any desired data, with the public key (available for anyone) any other entity can check the signature and thus prove the authenticity of the data in relation to the signing entity.

Double spending – Family of attacks aimed at spending bitcoins from certain inputs multiple times; this can happen if an evil user is in some way able to create two transactions containing the same subset of inputs, and making the victim accept one of them, while the other one is the one really being included in the final accepted blockchain.

DSA – *See "Digital Signature Algorithm".*

ECDSA – *See "Elliptic Curve DSA".*

Elliptic Curve DSA – Alternative implementation of DSA, based on the algebraic structure of elliptic curves over finite fields instead of discrete logarithms; it brings shorter keys for the same security level and generally faster computations for signatures and verifications.

EWallet – Web service allowing a (trusting) user to store and manage his/her private/public keys, currency amount, and transaction log on the web, thus providing online Wallet functionalities.

Exchange – Web service providing users with the possibility of trading bitcoins for other altcoins or even real currencies.

Fork – Split of the blockchain where different network parts see (and follow) a different sequence of blocks from the fork; this usually happens when blocks with the same difficulty are mined and broadcasted around the same time. The situation usually resolves by itself automatically as more blocks are found and one fork becomes the longest and is then the accepted main blockchain.

Genesis Block – The first (original) block in the blockchain, it contains hardcoded data and serves as base for the creation of the next blocks in the chain which recursively refer to the previous one.

Hash – Number obtained from a hash function for the given input data, the generated number is deterministic and of fixed-length; a (secure) hashing function is not invertible (one-way function), gives a completely different result for any small change in the input, and is built to minimize the probability of collisions.

Hiding Set – Set of Bitcoin addresses assigned to a specific address where the latter can "hide" into, since we don't know what relation exists between them. The rest of the existing addresses that are outside the hiding set are by definition known to be unrelated to said address.

Inflation – Increase of prices of goods/services over time due to a decreasing value of money, causing people to spend more and hoard less.

Mempool – List of transactions kept by nodes which are not yet included in the blockchain (unconfirmed), they are retained until they expire or are included into a newly mined block.

Merged Mining – Procedure through which a miner can, by selecting several different cryptocurrencies that support it (plus Bitcoin), use the hashes calculated for finding the solution to a Proof of Work for multiple currencies at the same time; thus improving personal gains and security (due to increased total hashing power) for all the selected coins.

Merkle Tree – Tree structure where every non-leaf node contains the hash of all its leaves; this type of structure allows for fast and secure verification of the contents inserted in it.

Mining – The process of trying to find a valid hash (under the target) for a new always updated node containing the latest transactions. Hashing is done on the block header by continuously modifying various variables; as it is a random (lucky) process, the probability of finding a new block is given by the speed at which the used machine performs hashes. The reward for mining a new block comes in the form of the included transactions' fees and an initial single block reward.

Mining Pool – Service allowing multiple entities to collaborate and work together on mining new blocks, as the hashing rate is summed and the rewards are combined and distributed proportionally to the work done, this allows for a steadier income for the miners.

Node – A node (client) is a device connected to the Bitcoin network running some version of the Bitcoin program and thus able to receive, understand and send messages following the protocol. Nodes usually perform the important operations of relaying or broadcasting valid data and exchanging other information through the peer-to-peer network.

Nonce – Number or sequence of bytes (often chosen randomly) meant to be used only once in a cryptographic communication or algorithm.

Orphan Block – Valid block which is however not linked to the current blockchain, due to blocks referenced as "previous" being missing and not currently known; is ignored until the unknown blocks are received and validated. Blocks in the shorter chains are also called orphans.

Orphan Transaction – Valid transaction which contains at least an input referring to a previous transaction which is not currently known because it was not yet received or invalid; is ignored until all the missing transactions are received and validated.

PoS – *See "Proof of Stake".*

PoW – *See "Proof of Work".*

Proof of Stake – Alternative method to Proof of Work to protect the cryptocurrency, for now it is used in conjunction with it in the Altcoins utilizing it. With Proof of Stake the amount of currency you already have proportionally affects the amount of new blocks you can mine.

Proof of Work – A proof (usually a number) which is difficult to compute and thus, when shown, demonstrates spent time and power on the previously decided problem. In the Bitcoin world the Proof of Work is the creation of a hash whose number is below a certain target.

Reward – Bitcoins that are awarded to the discoverer of the block (miner); they are given in a special transaction at the beginning of the block.

Satoshi – Pseudonym of Bitcoin's creator, and name of the smallest unit (arbitrarily set in the code) in which bitcoins can be divided to execute transactions: one bitcoin is equal to 100 million satoshis.

Scrypt – Password-based key derivation function; an algorithm, used in cryptocurrencies as a proof of work, designed to be expensive in both CPU cycles and memory needed, with the objective of making custom hardware devices for its execution difficult and expensive to obtain.

SHA-2 – Set of cryptographic hash functions; currently they are the most used functions in various applications and are seen as secure by the community, with no known practical attack on their security.

Simplified Payment Verification – Client feature allowing nodes to verify transactions without having to keep the whole blockchain updated, by using block headers and bloom filters. It brings space and network savings but makes the node more vulnerable to certain attacks.

SPV – *See "Simplified Payment Verification".*

Target – Number (mask) representing the upper limit for a new valid block hash; to higher targets corresponds a higher probability of finding an accepted hash and vice versa. The target is continuously adjusted after every block to ensure regular mining of blocks over time.

Transaction – Signed data describing the movement of bitcoins from a set of input addresses to a set of output addresses; usually refers to previous transactions indicating the balance of inputs. Transactions are broadcasted over the network and, if valid, are included in the next mined block and confirmed; additional future blocks increase confirmations.

Transaction Fee – Amount of bitcoins that the originator of a transaction decides to pay to the miner that will include the transaction in a new block (as incentive). It corresponds to the sum remaining after subtracting the output amounts from the total inputs of the transaction.

TX – *See "Transaction".*

Wallet – Application providing an interface to handle all the owned private and public key pairs and manage lists of transactions; since bitcoins are simply data recorded as part of a transaction in the blockchain, the wallet's purpose is to store and manage one's addresses.

Appendix

A.1 MongoDB Document Examples

Transaction

```
{
    "_id" : "4a4a38ea2d5d56b497a3278564fc0c91  \
             b0e079b552e67537366fec423dd7de07",
    "originators" : [
        {
            "id" : "1ce9c95f-3dd8-4b87-99fb-585613fd3df2",
            "timestamp" : NumberLong("1398432432722")
        },
        {
            "id" : "e62c8b71-876e-41fd-8168-bbaa1c4d933f",
            "timestamp" : NumberLong("1398432433198")
        },
        {
            "id" : "5d2cb1a9-7b20-49c8-90c7-7490f40dcd53",
            "timestamp" : NumberLong("1398432433786")
        },
        {
            "id" : "9eef8b2b-9bef-4b61-8dd8-817a6da21636",
            "timestamp" : NumberLong("1398432433804")
        }, (...)
    ],
    "inputs" : [
        "1FHbbjtLrqvMqHzYAecrsPRQbGKrejMNta",
        "1AAhQXYLGDvaQxE72bQRpNioPY7xtfEDhP",
        "1MPC7NHWNKLrnv4gxh9DJn38sAzucnwv6M"
    ]
}
```

Entity (incomplete)

```
{
    "_id" : "e5f7e870-dfd5-49c5-ba61-c2f4f5cd711c",
    "ip" : "***.***.***.***",
    "version_name" : "",
    "version_number" : "",
    "time_online_from" : NumberLong("1398633910312"),
    "time_online_to" : 0,
    "link_delay_minimum" : 1000
}
```

Entity (complete)

```
{
    "_id" : "5118569c-a8f6-4928-8766-5a20ae6d5fc8",
    "ip" : "***.***.***.***",
    "version_name" : "Satoshi",
    "version_number" : "0.9.1",
    "time_online_from" : NumberLong("1398433397397"),
    "time_online_to" : NumberLong("1398433828276"),
    "link_delay_minimum" : 61
}
```

Bloom Filter

```
{
    "_id" : "001000002801104480400110000800000101  \
             100A1140A800004000000002020208000002  \
             000000400000220062440004401040000030  \
             20000184A1004040400101C40041809020A0  \
             10000000600A80000000831A108000008000  \
             010180200024000042041008001001002008  \
             410048000001040002080000000008000608  \
             A49800000009081000100040044608280002  \
             081280000268480600120000000A80400201  \
             001101000000000000880000000404000441  \
             060020000A90100080004092080000001000  \
             010000-0000000A-890ED400-02"
}
```

A.2 Bitshark Protocol Functions

getIPs

Call:

```
{
   "method" : "getIPs",
   "nips" : 1000,
   "workerid" : 1 // Zero here would mean we are a new worker
}
```

Answer:

```
{
   "result" :
      {
         "timestamp" : 1404086400000,
         "workerid" : 1,
         "ips" : ["***.***.***.***", "***.***.***.***", (...)]
      },
   "error" : null,
   "id" : 1
}
```

newIPs

Call:

```
{
   "method" : "newIPs",
   "ips" : ["***.***.***.***", "***.***.***.***", (...)]
}
```

Answer:

```
{
   "result" :
      {
         "timestamp" : 1404086400000
      },
   "error" : null
}
```

newLogs

Call:

```
{
   "method" : "newLogs",
   // Each call will have either one of the possible arrays:
   // transactions/entities/bloomfilters
   "transactions" : [
      {
         "_id" : "4a4a38ea2d5d56b497a3278564fc0c91  \
                  b0e079b552e67537366fec423dd7de07",
         "originators" : [
            {
               "id" : "1ce9c95f-3dd8-4b87-99fb-585613fd3df2",
               "ip" : "***.***.***.***",
               "timestamp" : NumberLong("1398432432722")
            },
            {
               "id" : "e62c8b71-876e-41fd-8168-bbaa1c4d933f",
               "ip" : "***.***.***.***",
               "timestamp" : NumberLong("1398432433198")
            }, (...)
         ]
      }, (...)
   ],
   "entities" : [
      {
         "_id" : "5118569c-a8f6-4928-8766-5a20ae6d5fc8",
         "ip" : "***.***.***.***",
         "version_name" : "Satoshi",
         "version_number" : "0.9.1",
         "time_online_from" : NumberLong("1398433397397"),
         "time_online_to" : NumberLong("1398433828276"),
         "link_delay_minimum" : 61
      }, (...)
   ],
   "bloomfilters" : [
      {
         "_id" : "0010000028011044804001100008000000101  \
                  100A1140A8000040000000002020208000002  \
                  0000004000002200624400044010400000030  \
                  20000184A1004040400101C40041809020A0  \
                  10000000600A80000000831A108000008000  \
                  0101802000240000420410080010010002008  \
                  4100480000010400020800000000008000608  \
                  A4980000000908100010004004460828002  \
                  08128000026848060012000000A80400201  \
                  0011010000000000000880000000404000441  \
                  060020000A901000800040920800000001000  \
                  010000-0000000A-890ED400-02"
      }, (...)
   ],
}
```

Answer:

```
{
   "result" :
      {
         "timestamp" : 1404086400000
      },
   "error" : null
}
```

newIncoming

Call:

```
{
   "method" : "newIncoming",
   "ip" : "***.***.***.***"
}
```

Answer:

```
{
   "result" :
      {
         "timestamp" : 1404086400000
      },
   "error" : null
}
```

removeIP

Call:

```
{
   "method" : "removeIP",
   "ip" : "***.***.***.***"
}
```

Answer:

```
{
   "error" : null
}
```

A.3 Bitcoin Client Code Extracts

ThreadMessageHandler

Thread receiving and sending all messages from and to all connected nodes.

Code source: Bitcoin 0.9.1 - net.cpp

```
void ThreadMessageHandler()
{
   (...)
   while (true)
   {
      (...)

      // Poll the connected nodes for messages
      CNode* pnodeTrickle = NULL;
      if (!vNodesCopy.empty())
         pnodeTrickle = vNodesCopy[GetRand(vNodesCopy.size())];

      bool fSleep = true;

      BOOST_FOREACH(CNode* pnode, vNodesCopy)
      {
         if (pnode->fDisconnect) continue;

         // Receive messages
         (...)
         if (!g_signals.ProcessMessages(pnode))
            pnode->CloseSocketDisconnect();

         if (pnode->nSendSize < SendBufferSize())
         {
            if (!pnode->vRecvGetData.empty() ||
                (!pnode->vRecvMsg.empty() &&
                 pnode->vRecvMsg[0].complete()))
            {
               fSleep = false;
            }
         }
         (...)

         // Send messages
         {
            (...)
            g_signals.SendMessages(pnode, pnode == pnodeTrickle);
         }
         (...)
      }

      (...)

      if (fSleep) MilliSleep(100);
   }
}
```

SendMessages

Function called for all the connected peers (`CNode* pto`), responsible for sending most of the network messages.

Code source: Bitcoin 0.9.1 - main.cpp

```
bool SendMessages(CNode* pto, bool fSendTrickle)
{
    {
        (...)
        vector<CInv> vInv;
        vector<CInv> vInvWait;
        {
            (...)
            BOOST_FOREACH(const CInv& inv, pto->vInventoryToSend)
            {
                if (pto->setInventoryKnown.count(inv)) continue;

                // trickle out tx inv to protect privacy
                if (inv.type == MSG_TX && !fSendTrickle)
                {
                    // 1/4 of tx invs blast to all immediately
                    static uint256 hashSalt;
                    if (hashSalt == 0) hashSalt = GetRandHash();
                    uint256 hashRand = inv.hash ^ hashSalt;
                    hashRand = Hash(BEGIN(hashRand), END(hashRand));
                    bool fTrickleWait = ((hashRand & 3) != 0);

                    if (fTrickleWait)
                    {
                        vInvWait.push_back(inv);
                        continue;
                    }
                }

                // returns true if wasn't already contained in the
                    set
                if (pto->setInventoryKnown.insert(inv).second)
                {
                    vInv.push_back(inv);
                    if (vInv.size() >= 1000)
                    {
                        pto->PushMessage("inv", vInv);
                        vInv.clear();
                    }
                }
            }
            pto->vInventoryToSend = vInvWait;
        }
        if (!vInv.empty()) pto->PushMessage("inv", vInv);
        (...)
    }
    return true;
}
```

ProcessMessages

Function called from the client whenever a new message is received from a peer node (`CNode* pfrom`), responsible for reacting to messages.

Code source: Bitcoin 0.9.1 - main.cpp

```
bool static ProcessMessage(CNode* pfrom, string strCommand,
                           CDataStream& vRecv)
{
   (...)

   else if (strCommand == "inv")
   {
      vector<CInv> vInv;
      vRecv >> vInv;
      (...)

      for (unsigned int nInv = 0; nInv < vInv.size(); nInv++)
      {
         const CInv &inv = vInv[nInv];

         boost::this_thread::interruption_point();
         pfrom->AddInventoryKnown(inv);

         bool fAlreadyHave = AlreadyHave(inv);
         (...)

         if (!fAlreadyHave)
         {
            /* AskFor queues the "getdata" request, which
               is later processes by the SendMessages function */
            if (!fImporting && !fReindex) pfrom->AskFor(inv);
         }
         (...)
      }
   }

   else if (strCommand == "getdata")
   {
      vector<CInv> vInv;
      vRecv >> vInv;
      (...)

      pfrom->vRecvGetData.insert(pfrom->vRecvGetData.end(),
                                 vInv.begin(), vInv.end());
      /* ProcessGetData immediately sends a "tx"
         answer to the peer if the request is valid */
      ProcessGetData(pfrom);
   }

   (...)
```

```
    else if (strCommand == "tx")
    {
        (...)
        CTransaction tx;
        vRecv >> tx;

        CInv inv(MSG_TX, tx.GetHash());
        pfrom->AddInventoryKnown(inv);

        (...)
        if (AcceptToMemoryPool(mempool, state, tx, true,
                               &fMissingInputs))
        {
            (...)
            /* TODO: comment RelayTransaction
               Bla */
            RelayTransaction(tx, inv.hash);
            (...)

            // Recursively process any orphan transactions
            // that depended on this one
            for (unsigned int i = 0; i < vWorkQueue.size(); i++)
            {
                uint256 hPrev = vWorkQueue[i];
                for (set<uint256>::iterator mi =
                      mapOrphanTransactionsByPrev[hPrev].begin();
                      mi != mapOrphanTransactionsByPrev[hPrev].end();
                      ++mi)
                {
                    const uint256& orphanHash = *mi;
                    const CTransaction& orphanTx =
                        mapOrphanTransactions[orphanHash];
                    (...)

                    if (AcceptToMemoryPool(mempool, stateDummy,
                        orphanTx, true, &fMissingInputs2))
                    {
                        (...)
                        /* TODO: comment RelayTransaction
                           Bla */
                        RelayTransaction(orphanTx, orphanHash);
                        (...)
                    }
                    (...)
                }
            }
            (...)
        }
        (...)
    }

    (...)
}
```

Bibliography

[And12] Elli Androulaki, Ghassan Karame, Marc Roeschlin, Tobias Scherer, Srdjan Capkun: “Evaluating User Privacy in Bitcoin”. *Financial Cryptography and Data Security*, pp. 34–51; 2012.

[And14] Elli Androulaki, Ghassan Karame: “Hiding Transaction Amounts and Balances in Bitcoin”. *Trust and Trustworthy Computing*, pp. 161–178; 2014.

[Bab12] Moshe Babaioff, Shahar Dobzinski, Sigal Oren, Aviv Zohar: “On Bitcoin and Red Balloons”. *ACM Electronic Commerce*, pp. 56–73; 2012.

[Bam13] Tobias Bamert, Christian Decker, Lennart Elsen, Roger Wattenhofer, Samuel Welten: “Have a Snack, Pay with Bitcoins”. *IEEE Peer-to-Peer Computing*, pp. 1–5; 2013.

[Bar12] Simon Barber, Xavier Boyen, Elaine Shi, Ersin Uzun: “Bitter to Better – How to Make Bitcoin a Better Currency”. *Financial Cryptography and Data Security*, pp. 399–414; 2012.

[Bec13] Jörg Becker, Dominic Breuker, Tobias Heide, Justus Holler, Hans Peter Rauer, Rainer Böhme: “Can We Afford Integrity by Proof-of-Work? Scenarios Inspired by the Bitcoin Currency”. *The Economics of Information Security and Privacy*, pp. 135–156; 2013.

[Ben14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, Madars Virza: “Zerocash: Decentralized Anonymous Payments from Bitcoin”. *IEEE Security and Privacy*; 2014.

[Dec13] Christian Decker, Roger Wattenhofery: “Information Propagation in the Bitcoin Network”. *IEEE Peer-to-Peer Computing*, pp. 1–10; 2013.

[Dob96] Hans Dobbertin, Antoon Bosselaers, Bart Preneel: "RIPEMD-160: A Strengthened Version of RIPEMD". *Fast Software Encryption*, pp. 71–82; 1996.

[Don14] Joan Antoni Donet Donet, Cristina Pérez-Solà, Jordi Herrera-Joancomartí: "The Bitcoin P2P Network". **http://fc14.ifca.ai/bitcoin/papers/bitcoin14_submission_3.pdf**; 2014.

[Eya13] Ittay Eyal, Emin Gün Sirer: "Majority Is Not Enough: Bitcoin Mining Is Vulnerable". **http://arxiv.org/pdf/1311.0243v5.pdf**; 2013.

[FIP13] FIPS: "Digital Signature Standard (DSS) *(revision 4)*". **http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf**; 2013.

[Ger13] Arthur Gervais, Ghassan Karame, Srdjan Capkun, Vedran Capkun: "Is Bitcoin a Decentralized Currency?". *International Association for Cryptologic Research*; 2013.

[Gri12] Reuben Grinberg: "Bitcoin: An Innovative Alternative Digital Currency". *HeinOnline*; 2012.

[Joh01] Don Johnson, Alfred Menezes, Scott Vanstone: "The Elliptic Curve Digital Signature Algorithm (ECDSA)". *International Journal of Information Security*, vol. 1, no. 1, pp. 36–63; 2001.

[Kar12] Ghassan Karame, Elli Androulaki, Srdjan Capkun: "Two Bitcoins at the Price of One? Double-Spending Attacks on Fast Payments in Bitcoin". *ACM Computer and Communications Security*; 2012.

[Kin12] Sunny King, Scott Nadal: "PPCoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake". **http://peercoin.net/bin/peercoin-paper.pdf**; 2012.

[Kon13] Dániel Kondor, Márton Pósfai, István Csabai, Gábor Vattay: "Do the Rich Get Richer? An Empirical Analysis of the Bitcoin Transaction Network". **http://arxiv.org/pdf/1308.3892v3.pdf**; 2013.

[Kos14] Philip Koshy, Diana Koshy, Patrick McDaniel: "An Analysis of Anonymity in Bitcoin Using P2P Network Traffic". **http://www.bitcoinsecurity.org/wp-content/uploads/2014/01/Koshy-FC141.pdf**; 2014.

[Kro13] Joshua A. Kroll, Ian C. Davey, Edward W. Felten: "The Economics of Bitcoin Mining, or Bitcoin in the Presence of Adversaries". *Workshop on the Economics of Information Security*; 2013.

[Mei13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, Stefan Savage: "A Fistful of Bitcoins: Characterizing Payments Among Men With No Names". *ACM Internet Measurement*, pp. 127–140; 2013.

[Mer79] Ralph C. Merkle: "Method of Providing Digital Signatures". US Patent 4,309,569. **http://www.google.com/patents/US4309569**; 1979.

[Mie13] Ian Miers, Christina Garman, Matthew Green, Aviel D. Rubin: "Zerocoin: Anonymous Distributed E-Cash from Bitcoin". *IEEE Security and Privacy*, pp. 397–411; 2013.

[Moo13] Tyler Moore, Nicolas Christin: "Beware the Middleman: Empirical Analysis of Bitcoin-Exchange Risk". *Financial Cryptography and Data Security*, vol. 7859, pp. 25–33; 2013.

[Mös13] Malte Möser: "Anonymity of Bitcoin Transactions – An Analysis of Mixing Services". *Münster Bitcoin Conference*; 2013.

[Nak08] Satoshi Nakamoto: "Bitcoin: A Peer-to-Peer Electronic Cash System". **http://www.bitcoin.org/bitcoin.pdf**; 2008.

[NIS01] NIST: "Descriptions of SHA-256, SHA-384, and SHA-512". **http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf**; 2001.

[Obe13] Micha Ober, Stefan Katzenbeisser, Kay Hamacher: "Structure and Anonymity of the Bitcoin Transaction Graph". *Future Internet*, vol. 5, no. 2, pp. 237–250; 2013.

[Per12] Colin Percival, Simon Josefsson: "The Scrypt Password-Based Key Derivation Function". **http://tools.ietf.org/html/draft-josefsson-scrypt-kdf-01**; 2012.

[Rei11] Fergal Reid, Martin Harrigan: "An Analysis of Anonymity in the Bitcoin System". *IEEE Social Computing*, pp. 1318–1326; 2011.

[Ron13] Dorit Ron, Adi Shamir: "Quantitative Analysis of the Full Bitcoin Transaction Graph". *Financial Cryptography and Data Security*, pp. 6–24; 2013.

[Sax14] Amitabh Saxena, Janardan Misra, Aritra Dhar: "Increasing Anonymity in Bitcoin". **http://www.ifca.ai/fc14/bitcoin/papers/bitcoin14_submission_19.pdf**; 2014.

[Swa07] S. Joshua Swamidass, Pierre Baldi: "Mathematical Correction for Fingerprint Similarity Measures to Improve Chemical Retrieval". *Journal of Chemical Information and Modeling*, vol. 47, pp. 952–964; 2007.

[Sye11] Omar Syed, Aamir Syed: "Bitcoin Gateway – A Peer-to-Peer Bitcoin Vault and Payment Network". **http://arimaa.com/bitcoin/BitcoinGateway20110726.pdf**; 2011.

[Woo14] Christopher A. Wood, Chris H. Vu: "An Exploration of Bitcoin Anonymity". **http://christopher-wood.com/docs/WoodVu_BitcoinPrivacy_Survey.pdf**; 2014.

Additional References

[1] **http://en.wikipedia.org/wiki/Money**
Money – Wikipedia.

[2] **http://www.slideshare.net/slideshow/embed_code/954371**
“Uses and Characteristics of Money”.

[3] **http://www.coindesk.com/why-regulating-bitcoin-will-not-work**
“Why Regulating Bitcoin Won’t Work”.

[4] **http://www.coindesk.com/credit-cards-not-evolved-enter-bitcoin**
“Credit Cards Have Not Evolved With the Internet. Enter Bitcoin.”

[5] **http://news.cnet.com/8301-31322_3-57581952-256**
“Here’s Why Bitcoin Is the Future of Money”.

[6] **http://wired.com/wiredenterprise/2013/11/bitcoin-and-deflation**
“Bitcoin Is Flawed, But It Will Still Take Over the World”.

[7] **http://en.wikipedia.org/wiki/Discrete_logarithm_problem**
Discrete logarithm – Wikipedia.

[8] **http://en.wikipedia.org/wiki/Elliptic_curves**
Elliptic curve – Wikipedia.

[9] **http://en.wikipedia.org/wiki/Elliptic_curve_cryptography**
Elliptic curve cryptography – Wikipedia.

[10] **https://en.bitcoin.it/wiki/Trade**
List of entities trading and offering Bitcoin services – Bitcoin Wiki.

[11] **http://slideshare.net/CoinDesk/coindesk-state-of-bitcoin-2014**
“CoinDesk State of Bitcoin 2014”.

[12] **https://github.com/bitcoin/bitcoin**
The Bitcoin source code repository.

[13] **https://bitcointalk.org**
The official Bitcoin forum.

[14] **https://en.bitcoin.it/wiki/Main_Page**
The Bitcoin Wiki.

[15] **http://bitcoin.stackexchange.com**
Bitcoin Stack Exchange.

[16] **http://irc.lc/freenode/bitcoin**
Bitcoin IRC channel.

[17] **http://blockexplorer.com**
Realtime information about blocks, addresses and transactions.

[18] **http://blockr.io**
Blockr – Bitcoin and altcoin block, transaction and exchange statistics.

[19] **https://blockchain.info**
Blockchain – Bitcoin statistics and block/transaction explorer.

[20] **http://getaddr.bitnodes.io**
Bitnodes – Analysis and estimation of the Bitcoin node network.

[21] **http://bitcoincharts.com**
Bitcoincharts – Financial and technical Bitcoin statistics.

[22] **http://coinmarketcap.com**
Cryptocurrency market capitalizations.

[23] **https://en.bitcoin.it/wiki/Script**
Script – Bitcoin Wiki.

[24] **https://en.bitcoin.it/wiki/Proof_of_Stake**
Proof of Stake – Bitcoin Wiki.

[25] **https://en.bitcoin.it/wiki/Proof_of_burn**
Proof of Burn – Bitcoin Wiki.

[26] **https://en.bitcoin.it/wiki/Network**
Bitcoin network – Bitcoin Wiki.

[27] **https://en.bitcoin.it/wiki/Protocol_rules**
Bitcoin protocol rules – Bitcoin Wiki.

[28] **https://en.bitcoin.it/wiki/Protocol_specification**
Bitcoin protocol specification – Bitcoin Wiki.

[29] **https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery**
Bitcoin network node discovery – Bitcoin Wiki.

[30] **http://www.jsonrpc.org/specification**
JSON-RPC 2.0 specifications.

[31] **http://qt-project.org**
Qt Project – Cross-platform application and UI framework.

[32] **http://en.wikipedia.org/wiki/Bloom_filter**
Bloom filter – Wikipedia.

[33] **http://www.coindesk.com/51-attacks-real-threat-bitcoin**
"Are 51% Attacks a Real Threat to Bitcoin?"

[34] **https://en.bitcoin.it/wiki/Double-spending**
Double-spending – Bitcoin Wiki.

[35] **https://bitcointalk.org/index.php?topic=576337**
"List of Major Bitcoin Heists, Thefts, Hacks, Scams, and Losses".

[36] **http://www.coindesk.com/nearly-150-strains-malware-bitcoins**
"Nearly 150 Strains of Malware Are After Your Bitcoins".

[37] **http://www.coindesk.com/non-experts-guide-mt-gox-fiasco**
"The Non-Expert's Guide to the Mt. Gox Fiasco".

[38] **http://wired.com/wiredenterprise/2014/03/bitcoin-exchange**
"The Inside Story of Mt. Gox, Bitcoin's $460 Million Disaster".

[39] **http://www.coindesk.com/bitcoin-bug-guide**
"What the 'Bitcoin Bug' Means: A Guide to Transaction Malleability".

[40] **http://wired.com/wiredenterprise/2013/08/bitocoin_anonymity**
"Sure, You Can Steal Bitcoins. But Good Luck Laundering Them".

[41] **https://www.torproject.org**
TOR – Anonymity network through onion routing.

[42] **https://bitcointalk.org/index.php?topic=279249.0**
CoinJoin – Increased Bitcoin anonymity through coin mixing.

[43] **http://app.bitlaundry.com/**
BitLaundry – Anonymous mixing of Bitcoin transactions over time.

[44] **https://sharedcoin.com**
Shared Coin – Increased Bitcoin anonymity by transaction mixing.

[45] **http://fogcore5n3ov3tui.onion** *(TOR network)*
Bitcoin Fog – Anonymous Bitcoin mixing service via TOR.

[46] **http://6fgd4togcynxyclb.onion** *(TOR network)*
OnionBC – Anonymous TOR eWallet with escrow, mixing services.

[47] **http://nci2szjrwjqw2zbi.onion/** *(TOR network)*
TORwallet – Anonymous TOR eWallet with mixing services.

[48] **http://www.coindesk.com/bitcoin-is-as-anonymous-as-you-want-it**
"Why Bitcoin Is as Anonymous as You Want It to Be".

[49] **http://www.proofofexistence.com**
Document existence certification through the Bitcoin global log.

[50] **http://virtual-notary.org**
Electronic notary service for factoids using Bitcoin transactions.

[51] **https://www.ethereum.org**
Ethereum – Development platform for decentralized applications.

[52] **https://en.bitcoin.it/wiki/List_of_alternative_cryptocurrencies**
List of alternative cryptocurrencies – Bitcoin Wiki.

[53] **https://bitcointalk.org/index.php?topic=134179.0**
List of all cryptocoins.

[54] **http://coinchoose.com**
Real time statistics about Bitcoin and selected altcoins.

[55] **http://namecoin.info**
The Namecoin official website.

[56] **https://litecoin.org**
The Litecoin official website.

[57] **http://www.mastercoin.org**
The Mastercoin official website.

[58] **https://github.com/mastercoin-MSC/spec**
The Master protocol / Mastercoin complete specification.

[59] **https://github.com/ppcoin/ppcoin**
The PPCoin source code repository.

[60] **http://dogecoin.com**
The Dogecoin official website.
"Wow. Many Coin. Much profit. Very scrypt. Such random."

[61] **http://zerocoin.org**
The Zerocoin official website.

[62] **http://zerocash-project.org**
The Zerocash official website.

[63] **https://anoncoin.net**
The Anoncoin official website.

[64] **http://www.vmware.com/products/vsphere**
VMware vSphere – Server virtualization.

[65] **http://www.vmware.com/products/esxi-and-esx**
VMware ESXi – Virtual machine hypervisor.

[66] **http://www.ubuntu.com/server**
Ubuntu Server – Server edition of the Ubuntu Linux OS.

[67] **http://www.mongodb.org**
MongoDB – NoSQL document-oriented database system.

[68] **http://dev.maxmind.com/geoip**
MaxMind GeoIP – IP geolocation database.

[69] **http://www.citrix.com/products/xenserver/overview.html**
Citrix XenServer – Native hypervisor for server virtualization.