University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

A Decentralized Bitcoin Exchange with Bitsquare — Attack Scenarios and Countermeasures

Alexander Mülli
Zürich, Switzerland
Student ID: 07-603-301

Supervisor: Dr. Thomas Bocek, Daniel Dönni
Date of Submission: July 30, 2015

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

The recent demise of one of the major exchange markets for Bitcoin, a cryptographic currency, has motivated the development of alternative Bitcoin exchanges that follow the decentralized model of the Bitcoin network. This thesis analyses a proposed concept by the name of Bitsquare that operates a decentralized offer book based on a distributed hash table (DHT). After common attack vulnerabilities of the underlying DHT are identified, a possible countermeasure in form of a registry service based on the Bitcoin blockchain is proposed. The design uses the Bitcoin blockchain as an immutable registry and the block nonce, an output of a proof of work function, as a source of randomness to secure the assignment of peer identifiers in the DHT network. The concept was implemented as part of the TomP2P library and an evaluation in regards of its impact on performance was conducted. The evaluation showed that the created overhead might still lay in a range acceptable for a DHT application with moderate data alteration. In the worst case scenario where a data storage request is based on a previously unknown registration an overhead by factor 6 was measured.

Zusammenfassung

Der jüngste Vertrauensverlust in zentrale Bitcoin Börsen (engl. exchanges) durch wiederholte Insolvenzverfahren von grösseren Marktplätzen hat die Entwicklung von alternativen Bitcoin Marktplätzen angetrieben, die nach einem, vom dezentralen Design des Bitcoin Netzwerk inspirierten, Aufbau streben. Diese Masterarbeit untersucht eines solcher möglichen Modellen, das Bitsquare Projekt, welches den Ansatz hat ein dezentrales Orderbuch auf einer Verteilte Hash Tabellen (engl. Distributed Hash Tables, DHT) basiert umzusetzen. Nach einer Analyse bekannter Schwachstellen von DHTs für böswillige Angriffe und deren Bedeutung für Bitsquare, wurde eine mögliche Gegenmassnahme zur Unterbindung solcher Angriffe vorgeschlagen. Die Massnahme basiert auf einer Verwendung der Bitcoin Blockchain als unveränderbare Registrierungsdatenbank und benutzt die sogenannte Block-Nonce als Zufallsgenerator zur Absicherung der Zuteilung von PeerIDs im DHT Netzwerk. Das Konzept wurde als Teil der TomP2P Java-Library implementiert und im Bezug auf dessen Auswirkungen auf die Performanz evaluiert. Die Evaluation konnte aufzeigen, dass der Verlust an Performanz immer noch in einem Rahmen liegt, der ein mögliche DHT-Anwendung mit moderater Datenmutationsfrequenz realistisch macht. Im Worst-Case-Szenario, wo eine Speicheranfrage auf einer unbekannten Registration basiert, wurde eine Erhöhung der Antwortzeit um Faktor 6 gemessen.

Acknowledgments

I would like to thank several people for their support in the realization of this master thesis. First of all, I express my deepest gratitude to my supervisor, Dr. Thomas Bocek, for his competent assistance, patience, enthusiasm and ever cooperative interaction. Special appreciation also to Prof. Dr. Burkhard Stiller for the opportunity to write this thesis at the Communication Systems Group. Last but not least, I am grateful to my family and close friends for their ongoing encouragement and support throughout this work.

Contents

Chapter 1

Introduction

1.1 Motivation

Bitcoin has proven itself resilient now over a period of more than 5 years and might be considered a viable solution to a decentralized payment transaction system. However to enter the Bitcoin ecosystem one has still go through centralized exchanges that bear certain risks. Various projects are trying to come up with a solution to mitigate these risk by applying Bitcoin's philosophy of decentralization to marketplaces such as OpenBazaar [1] and Bitsquare [2]. Specifically Bitsquare is relaying on a distributed hash table to decentralize the storage of exchange offers. This opens up a wide array of possibilities for attacks and manipulation.

1.2 Description of Work

To fully harness the benefits of a decentralized currency like Bitcoin it is vital that there also exists a decentralized way to exchange national currencies for bitcoins. Bitsquare offers one possible solution to this. However with the application of a distributed hash table to decentralize the offer book the exchange is opened up to additional attack scenarios that have to be analyzed. After assessing what kind of attacks are most critical and can possibly be mitigated the goal of this thesis is also to propose possible countermeasures against these attacks. From the beginning the focus is set on what kind of security issues DHT face and what possibilities their are to restrict them in regards of the specific use case of Bitsquare.

1.3 Thesis Outline

Chapter 2 will first explain the background of this thesis, introduce Bitsquare and similar projects and give a short introduction into the relevant topics of bitcoin, cryptocurrency

exchanges and DHTs. Afterwards related work in regards of possible DHT attack countermeasures is discussed and applied to the DHT use cas of Bitsquare in Chapter 3. After describing the concept of the proposed countermeasure and how this was implemented as part of the TomP2P library in Chapter 4. Chapter 5 will describe an evaluation where the effects of this implementation on the DHT performance regarding response times were tested.

Chapter 2

Background & Related Work

This chapter discusses related work and gives background information on a related topics.

2.1 Bitcoin

Bitcoin is a peer-to-peer (P2P) payment system based on a paper published in 2009 by Satoshi Nakamoto [3]. The design of Bitcoin builds on previously existing technologies such as public-key cryptography and cryptographic hash functions. The combination of those building blocks brought the first decentralized consensus system that has been standing firm against any attacks over a period of more than 5 years. There doesn't exist a formal specification of the Bitcoin protocol, however Nakomoto also published the Bitcoin Core client that still defines the protocol as the reference implementation and has been under ongoing development since.

The units of account of this payment system are also called bitcoins. In this thesis the expression Bitcoin, capitalized, is used to refer to the technology and network and bitcoin, lowercase, to refer to the unit of account. Anybody can take part in this payment system by creating a Bitcoin wallet and connecting to the network. A wallet is based on a public private key-pair. From the private key a Bitcoin address can be generated where bitcoins can be received to and/or sent from. The network gives anybody access to see any payments that are occurring between any addresses. In contrast to other payment systems it isn't built on a big database of account records, where for example each address has a certain balance. Instead bitcoins are recorded as transactions. For example, some user Carol does not just hold 5 bitcoins in his address, but rather takes part in a publicly verifiable transaction where Bob transfers 5 bitcoins to Carol. Carol was able to verify that Bob is allowed to make that payment because there exists a publicly verified prior transaction where Alice transferred 5 bitcoins to Bob.

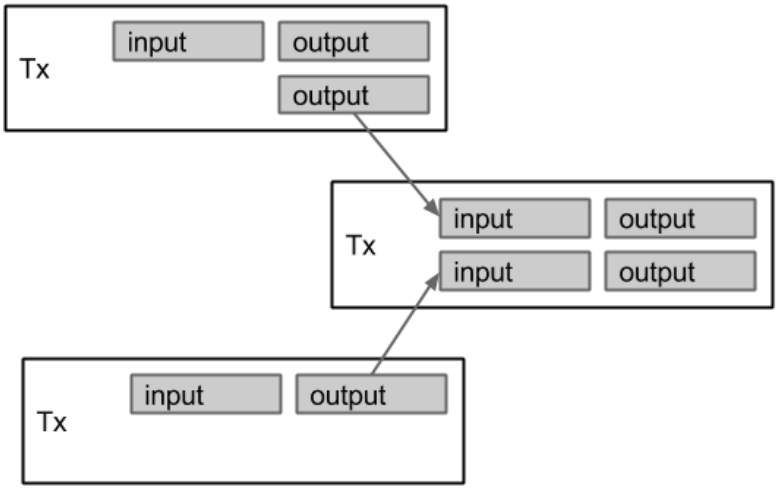

Figure 2.1: Simplified illustration of how Bitcoin transactions reference to each other.

2.1.1 Transactions

Having a closer look at the building blocks of a transaction in the Bitcoin network reveals that the Bitcoin protocol allows for more complex transactions than a simple payment from Bob to Carol. All transactions consist of multiple inputs and outputs (see Figure 2.1). Normally there is either one input from a larger previous transaction or multiple inputs combining the amount of previous small transactions. In most cases there are not more than two outputs: One with the amount of bitcoins transferred to the recipient and another one with the change of the transaction back to the sender.

A transaction output consists of the amount of bitcoins and a locking-script that specifies the conditions that must be met in order to be able to spend this output in another transaction. An input includes a reference to a specific output of a specific transaction together with an unlocking-script. A transaction can be verified by running a combination of the locking-script and unlocking-script. This scripts are based on simple Bitcoin specific scripting language that allows to introduce additional conditions for a transaction to be valid and spendable. In the following sections a selection of Bitcoin's standard transactions that are relevant for this thesis' scope will be introduced.

Pay-to-Public-Key-Hash Transaction

The Pay-to-Public-Key-Hash (P2PkH) transaction is the most common occurring transaction type in the Bitcoin network and corresponds to the previous example with Bob and Carol. It contains an output with a locking-script that binds it to a double hash of a public key, also called a Bitcoin address. The equivalent unlocking-script in the input presents a signature based on this public key's corresponding private key.

Multi-Signature Transaction

With multi-signature scripts allow to set conditions where more than one party has to give permission to unlock a transaction output. In the locking script a M-of-N-scheme can be setup that specifies how many signatures (M) of a specified set of public keys (N) have to provide there signature in the unlocking-script. For example with a 2-of-3 multi-signature transaction the transferred amount of bitcoin would be locked in a transaction output that can only be spent in another transaction when at least 2 out of 3 potential signers that were defined in the locking-script give there signatures and herewith their consent.

Data Output (OP_RETURN) Transaction

Adding a OP_RETRUN statement has also the effect that this output can not be used as an input in another transaction. Any amount of bitcoin that is associated with this output becomes therefore provably unspendable. However more importantly as the OP_RETURN statement stops the validation process and anything afterwards is ignored it is possible to add any random 80 bytes of data to a transaction output. The ability of adding data to a transaction and consequently to the blockchain can been utilized for different purposes. For example there exists a service called "proof of existence" that allows to anonymously and securely store an online distributed proof of existence for any document

Time Locked Transaction

Actually technically not a type of transaction per se but an option for any kind of transaction is a time lock. The standard structure of a bitcoin transaction includes a *Locktime* variable (see Table 2.1) that allows to specify the earliest time a transaction is valid and can be relayed on the network. The time can be either defined as a Unix timestamp or with a block number (also called block height) that references the earliest block it would be allowed to be included. A transaction is not allowed to be confirmed by miners until the lock time is reached. Since Bitcoin 0.8+ a transaction that did not end its lock period (non final) is considered to be non standard and won't be relayed or included in the memory pool either. This means that a transaction with a lock time in the future is not stored by the network to be included at a later point. One has to make sure that it is published to the network when it has reached the specified time lock. In most use cases such a transaction can be transmitted to the receiver of the bitcoin output and with this hand over the responsibility to publish it on the network at the appropriate time.

2.1.2 The Blockchain

In the previous example with Carol and Bob it was stated that Carol is able to verify that the payment of Bob was legit as it was based on a publicly verified prior transaction. It was omitted how this transaction is publicly verified. With just the transaction from Bob, Carol can not be sure that e.g. Bob is not also making a payment to Dan based

Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1–9 bytes (VarInt)	Input Counter	How many inputs are included
Variable	Inputs	One or more transaction inputs
1–9 bytes (VarInt)	Output Counter	How many outputs are included
Variable	Outputs	One or more transaction outputs
4 byte	Locktime	A Unix timestamp or block number

Table 2.1: Structure of a Bitcoin transaction. [4]

on the same prior transaction. This is called the double spending problem and is solved by the Bitcoin blockchain. Transactions that are broadcast to the bitcoin network are picked up by so called miners that verify these transactions and consolidate them into so called blocks. To create a valid block miners have to solve a computational costly problem. Whoever succeeds first at solving that problem, also called mining a block, gets rewarded with an additional transaction (also called coinbase transaction) in this block to his own address with a defined amount of newly mined bitcoins. A block always includes a reference to the previous block, therefore creating a chain of blocks and with this a verified chronological order of occurred transactions. Table 2.2 shows the structure of such a block in more detail [4]. A new block is mined roughly every 10 minutes and thereby adding the included transactions to the blockchain. Transactions that are part of the blockchain are considered ”confirmed” and the new owners of the bitcoins received in this transaction can be spent in another transaction. So, as soon as Bob sees the transaction from Carol included in the blockchain, confirmed by a sufficient amount of blocks, he can be sure that he is now the owner of those bitcoins Carol transferred to him. One block confirmation for Carol is enough to rule out that Bob is attempting a double spend, however there are other possible attacks than a double spend that could be occurring on Bobs transaction that can be ruled out after 6 confirmations.

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	**Block Header**	**Several fields form the block header**
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkel Root	A hash of the root of the merkle tree of this block's transactions
4 byte	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 byte	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 byte	Nonce	A counter used for the proof-of-work algorithm
1-9 bytes	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Table 2.2: Structure of a Bitcoin block. [4]

Blocks are difficult to forge as a high amount of computational resources have to be invested to build one. This concept is called Proof-of-work. Bitcoin's proof-of-work system is largely based on Adam Back's Hashcash [5]. Basically Miners search for a nonce that is part of the block so that a SHA256 hash of the block header lies below a certain threshold. In contrast to the amount of work that is needed to find a valid nonce, validating that the block has been mined correctly is very easy and the matter of a simple hash operation.

2.1.3 Simplified Payment Verification (SPV)

The block chain is getting bigger with every new block that is added. Currently it is of the size of 38.671 gigabytes [6]. To verify the correctness of each transaction a standard bitcoin client needs to store the whole blockchain locally. For some use cases as for example a mobile client this seems unpractical. Nakamoto already introduced in the founding paper [3] an alternative mode called simplified payment verification that allows the verification of transactions with a smaller storage footprint. Each block in the bitcoin blockchain contains a summary of all the transactions in the block, using a merkle tree. A merkle tree, also known as a binary hash tree, is a data structure used for efficiently summarizing and verifying the integrity of large sets of data. Merkle trees are binary trees containing cryptographic hashes. Merkle trees are used extensively by SPV nodes. SPV nodes don't have all transactions and do not download full blocks, just block headers. In order to verify that a transaction is included in a block, without having to download all the transactions in the block, they use an authentication path, or merkle path [4].

2.1.4 The Bitcoin Ecosystem

One of Bitcoin's distinct characteristics is its emphasis on decentralization. However with the rise of Bitcoin an ecosystem has emerged that largely relies on a broad range of third-party intermediaries such as currency exchanges, escrow services and online wallets [7].

As the mining difficulty has been continuously increasing over the last few years, bitcoin mining has become unprofitable for normal users. Therefore the conventional way to acquire bitcoins is to buy them through an exchange. Over the last few years there has been a variety of Bitcoin exchanges that had risen in popularity and volume only to disappear a short time after [8]. The most famous example of this is probably the case of the Mt.Gox which in February 2014, at that time the biggest Bitcoin exchange, filed for bankruptcy and many customers lost their funds they had with that company [9].

2.2 Bitsquare

Bitsquare is an open source software project with the goal to build a decentralize P2P Bitcoin exchange. It started in early 2014 as a proof of concept prototype by Manfred Karrer. Since July 2014 the community around Bitsquare has steadily been growing with other developers joining and also contributing to the code base. The currently released version is still in alpha and is not being used for real trading for now. Eventually it will provide a market for exchanges of national currencies such as dollars, euros or yen for bitcoin. Also exchanges between a wide range of alternative crypto currencies for bitcoin will be a possibility. The concept of Bitsquare distinguishes itself form centralized bitcoin exchanges in these major characteristics:

- Decentralized: Bitsquare follows the mantra of decentralization in every aspect. Both in a technical and conceptual sense there is no central instance needed to conduct a single exchange. All trades are person-to-person, the matching happens manually on one side of the two parties and no communication relies on central server. In case of disputes between two exchange parties arbitration is done by a third party that was agreed on before the trade was initiated.
- No Funds: No third party has to be trusted with full control over any funds.
- Permission-free Access: There are no restrictions or any prerequisite for taking part and conducting trades.
- Privacy: Personal information (e.g. bank details) are only shared with the counterparty of a trade at the the point in time where it is relevant to complete the exchange.
- Open Source: Every aspect of the project is transparent. The Software is open source and the course of development is discussed openly on mailing lists and IRC.

What Bitsquare is not designed for is day trading. The matching is always done manually. There aren't any tools like stop-loss orders available. However because the trade always happens directly with another party the settlement might often be faster than with centralized exchanges which often impose delays when cashing out or adding funds.

Bitsquare is currently alpha-quality software The project is implemented in Java and uses libraries such as bitcoinj and TomP2P.

2.2.1 The Client

Unlike other popular Bitcoin exchanges Bitsquare is not web based but requires the user to install a software client in order to take part in trading via this platform. The client implements a Bitcoin wallet that is used for all transactions.

When launching the client it will connect to two networks: The Bitcoin network and the Bitsquare network. By connecting to other peers in the Bitsquare network the client receives the latest offers for buying and selling Bitcoin for other currencies. This offers are displayed in the order book.

The user can either take up on an existing offer or create one herself (see Figure 2.2). To prevent spam in the offer book and to align the incentives of both parties to go through with a trade once it is setup every offer and take request requires a small deposit in BTC.

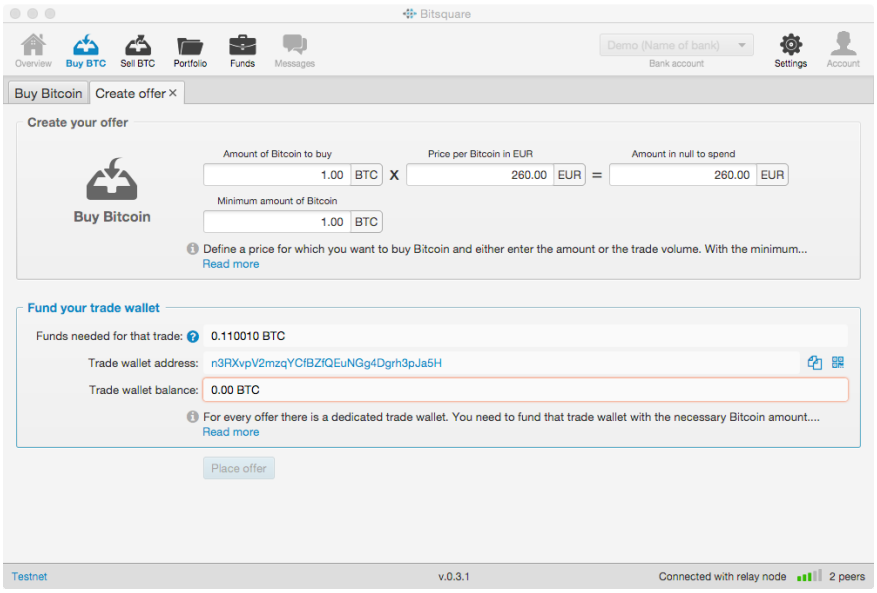

Figure 2.2: Screenshot of the Bitsquare client when creating a new offer.

2.2.2 The Trading Protocol

Bitsquares White Paper [10] demonstrates the Trading Protocol in detail. Buyer and Seller commit themselves to carry out the trade with a deposit in bitcoins that is locked together with the traded bitcoins in a multisig address until the exchange is completed. Figure 2.3 illustrates an example how a trade between Alice and Bob would take place. Alice is interested in buying bitcoins for US dollars. To find someone that is willing to sell her bitcoins she creates an offer with the relevant information in regards to amount, price and what kind of bank transfer she offers. Form Alice's Bitcoin Wallet a deposit transaction is created and sent to an address Alice has control over. The offer detail and the address of the deposit is stored in the offer book for everyone to see and verify.

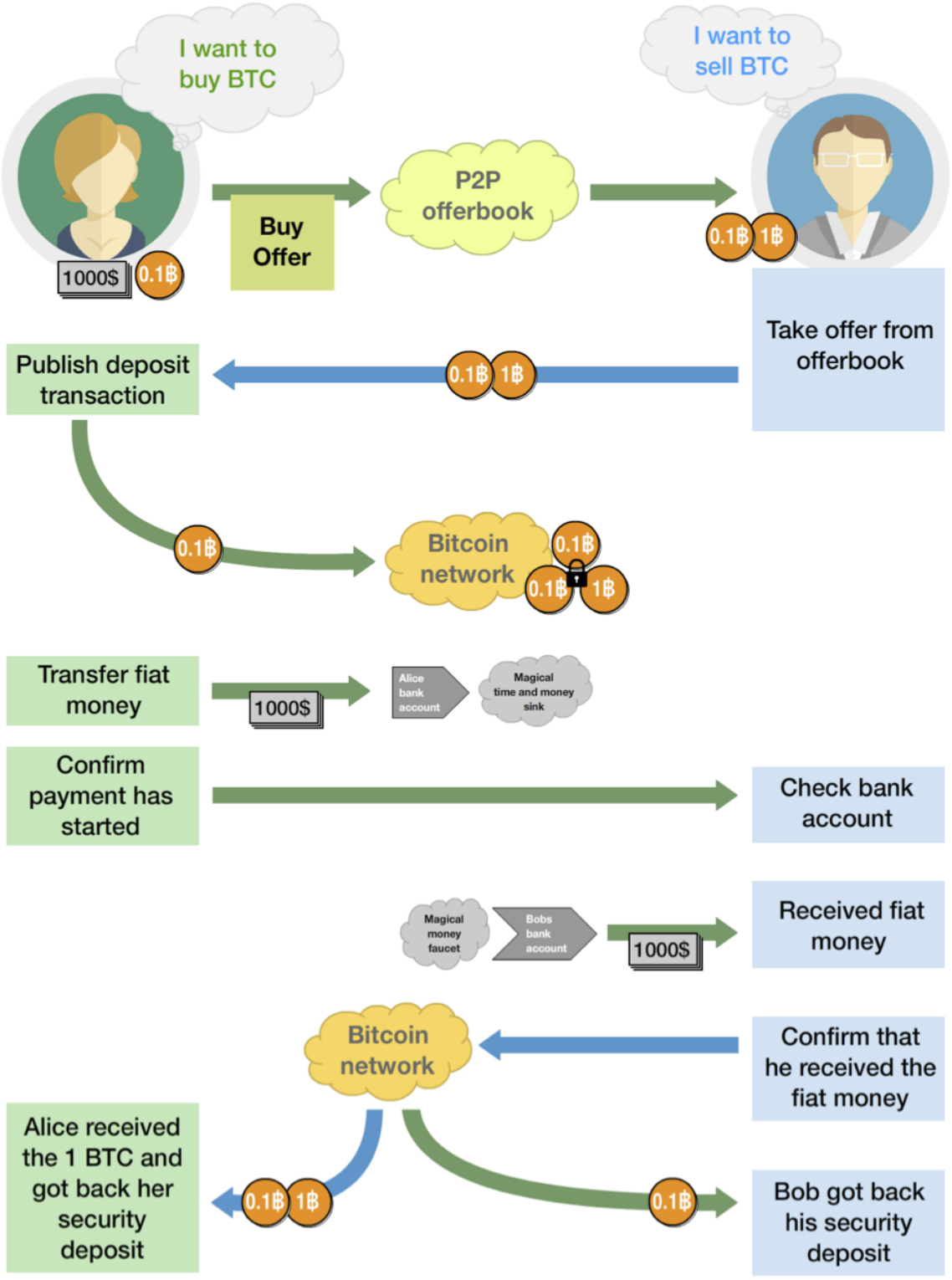

Figure 2.3: The Bitsquare trading protocol. [10]

Bob is interested in selling bitcoins. He has also created a deposit transaction from his Wallet as collateral for a trade. To take Alice's offer from the offer book a multisig-transaction is created with the two deposits from Alice and Bob and the bitcoins Bob

wants to sell as inputs. The multisig output is based on the public keys from Alice, Bob and an agreed on Arbitrator. The inputs from Bob are already being signed and the transaction is sent to Alice with a request to take her offer including Bob's bank account details. Alice then can sign her deposit input of the transaction and broadcast it to the Bitcoin network. With the inclusion of the transaction into the blockchain the funds are locked and can only be transferred when 2 out of the 3 signee agree. Alice can now wire the agreed on amount of US dollar to Bob's bank account and inform Bob when this step was completed. She prepares a transaction that will transfer the deposits back to Alice and Bob and the bitcoins she bought to an address of her choosing. Alice signs this transaction and sends it to Bob with the request to also sign it and broadcast it to the Bitcoin network. Bob will check his bank account and upon arrival of Alice's wire transfer sign and broadcast the transaction to the Bitcoin network.

2.2.3 Registration

In order to conduct such a Person-to-Person trade at some point privacy critical data such as bank account details to wire the transaction have to be shared with the other party. Bitsquare introduces an interesting concept for linking bank account information with a trading account that supports a high level of privacy where bank account details are only revealed to the buyer at the stage in the trade where they become relevant. An OP_RETURN transaction stores the blinded payments account data in the blockchain. The owner can prove ownership as the transaction is veritably linked with his Bitcoin address but cannot change or remove the associated data once stored in the block chain. For every additional bank account the user has to make a new registration [10].

2.3 Other decentralized exchanges

There are many other Bitcoin exchanges that label themselves as peer-to-peer and/or decentralized. In this section we describe a selection of exchanges that are similar to Bitsquare and highlight the shared characteristics and distinctions.

2.3.1 LocalBitcoins.com

Localbitcoins.com is an online platform that facilitates person-to-person currency exchange [11]. Users can post the amount and exchange rate of BTC they are willing to trade for various national currencies (see Figure 2.4). The transaction can be conducted face to face in cash or by any other means that allows remote payment such as online banking. As there is no third party or market maker that guarantees that users don't take advantage of each other there needs to be a certain trust between the exchanging parties. To mitigate the risk of being defrauded Localbitcoins.com offers a escrow service where the traded bitcoins are stored in a wallet that is controlled by Localbitcoins.com. This makes sure that the funds are only released to the buyer when the bank wire (or

whatever means of payment is used) has gone through successfully and is confirmed by the seller. In addition to the escrow service there is also a rating and feedback system that gives information about the trustworthiness of users based on their previous trades with other users. The escrow service might mitigate the risk of fraud by the counter-party but leaves the seller vulnerable towards Localbitcoins.com. By transferring bitcoins into the escrow wallet he looses any control over these funds. The platform is also not fully decentralized as it is run by a single company and the online platform can be assessed as a single point of failure.

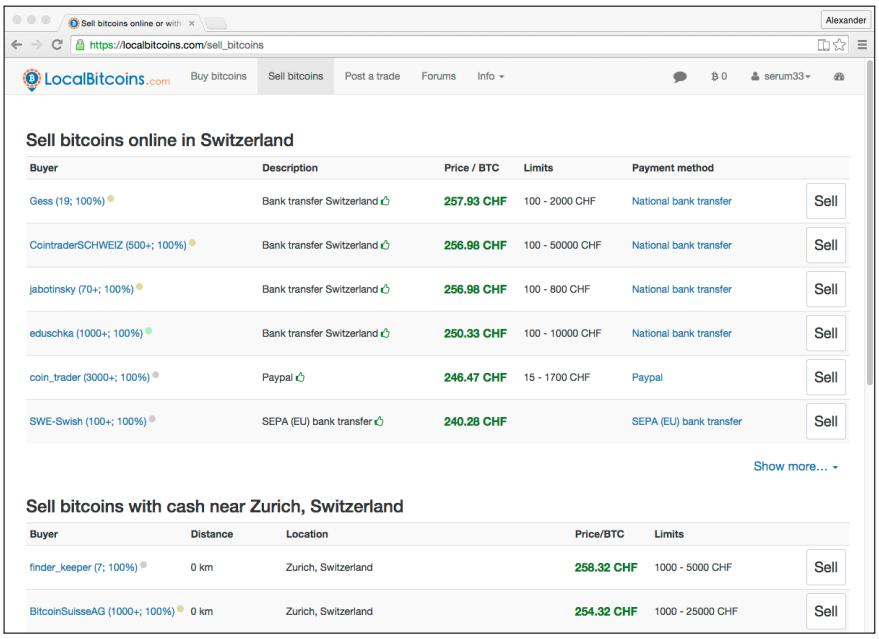

Figure 2.4: LocalBitcoins.com website with various, location specific bitcoin sell offers.

2.3.2 OpenBazar

OpenBazaar is an open source project to create a decentralized network for peer to peer commerce online using Bitcoin.[1] From its philosophy it is probably the project that is most similar to Bitsquare. The main differences is their focus on a general market place for any kind of offers and good and that isn't just limited to currency exchange. As from a technical view it is also built on an Kademlia DHT implementation, however developed with a different technology stack, mostly in Python.

At the time of writing the OpenBazar's implementation of the Network layer was in the midst of a major overhaul and there are some ongoing efforts to introduce some countermeasures against sybil attacks. One possible suggestion is that the peerID is a proof-of-work mechanism that set computational costs for generating a valid peerID: "we generate the peerID using sha512(signed public key) where the first 20 bytes are the peerID and the last 32 bytes are a 'proof of work'. The PoW must be below a certain t[h]reshold for the peerID to be valid. The target threshold is set to require about 30 seconds of brute forcing, on average, to find a vaild peerID." [12]

2.3.3 Coinffeine

Coinffeine is another open source, peer-to-peer bitcoin exchange platform that has begun in July 2015 to officially operate [13]. Similar to Bitsquare it is based on a software client (see Figure 2.5) with an integrated bitcoin wallet that connects to other peers in order to setup a peer-to-peer trades. In contrast to Bitsquare the underlying architecture however is not completely decentralized as it relies on a brokerage server that manages bid and ask orders and sets up the matching of two parties. An interesting aspect of this project is the implemented exchange algorithm that uses a variant of bitcoin contract known as Micropayment Channel to create deposits which force users to continue to cooperate in the transaction if they want to recover the deposit money [14]. By setting up deposits and executing the trade in small steps of minimal amounts (e.g. 0.01 BTC against 1€) the incentives to complete the trade for both parties aligns and promises no need for an arbitrator system as in Bitsquare. The perquisite for this to work is however that the payment method for the exchanged currency supports this kind of micro transactions with moderate fees. Coinffeine currently only supports OKPay as a payment method.

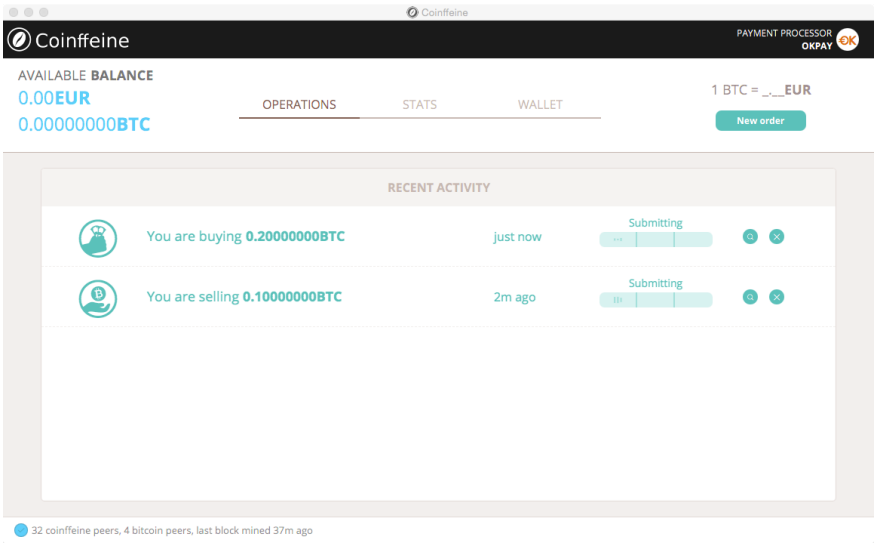

Figure 2.5: Screenshot the Coinffein client [13]

2.3.4 Mercury

Mercury is a open source project of decentralized cryptocurrency exchange [15]. Similar to Bitsquare the software client is only available in a alpha version (see Figure 2.6) and it is also implemented in Java, based on a fork of the bitcoinJ library. Its main difference to Bitsquare is that it is limited to exchange between cryptocurrencies: Bitcoin and other alternative cryptocurrencies like Litecoin. Users can submit their bid and ask orders that are then automatically matched by a order matching service. The actual exchange is guided by a atomic swap protocol where both parties create a deposit transaction of the cryptocurrency coin they want to trade with. A cryptographic mechanism where unlocking information is shared as part of the first transaction to redeem one of the

deposits guarantees that either none or both deposits can be redeemed by the two counter parties. This eliminates the need for any third party as both parties can be sure that the agreed exchange will go through entirely or not at all. Unfortunately this still leaves open the question how a similar exchange against national currencies could be implemented. The concept also leaves a centralized entity which is the matching service that matches bid and asks orders. Although no real trust is placed into that service as the funds always reside with the traders it still embodies a single-point-of-failure. Bitsquare evades this issue by not providing any order-matching service at all and leaves the selection of a matching offer as a manual task to the user. This of course has also its downsides as it might lead to a more time intensive setup process for each trade and might restrain the trading volume.

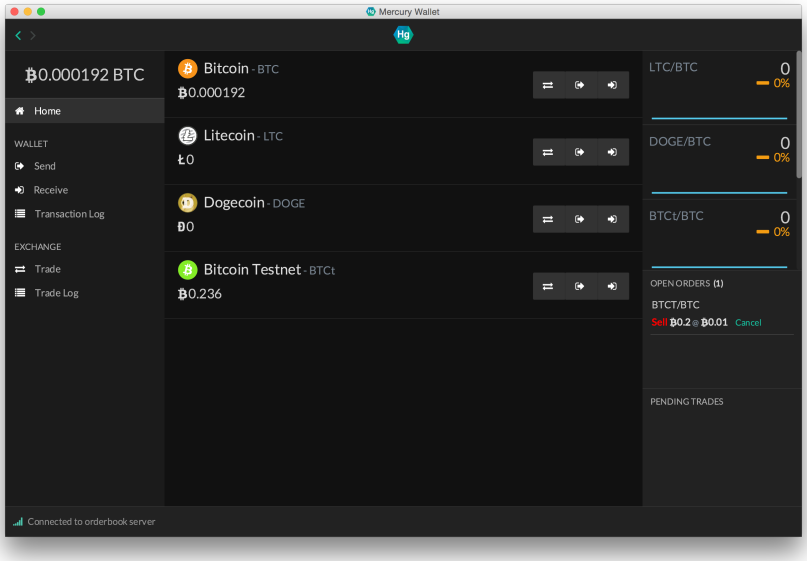

Figure 2.6: Mercury client in its current alpha version. [15]

2.4 Distributed Hash Table

A distributed hash table is a peer-to-peer network that manages the storage and retrieval of data in a distributed manner. All data is mapped to a key which is the data's hash and storing it on the peer whose identifier (peerID) is nearest to this key. So a DHT has a common identifier space for both peers and keys for data. To find the nearest identifier the distance between two identifiers is calculated. To attain an even distribution over the identifier space it is considered continuous where the first identifier is again the neighbour of the last one. Therefore a DHT's identifier space is conceptually illustrated as a circle of possible identifiers with distributed peers and the corresponding data they are responsible for (see Figure 2.7).

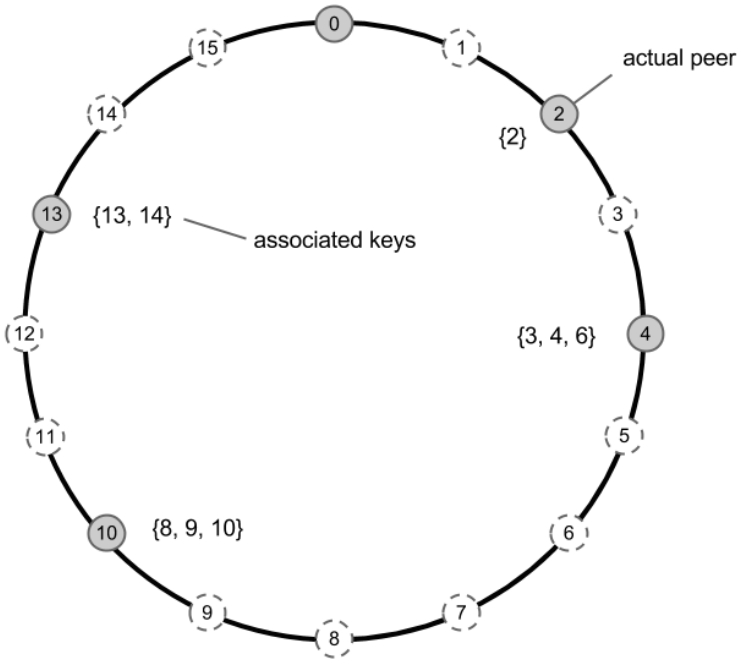

Figure 2.7: Simplified illustration of a DHT peerID/key storage concept

A routing mechanism creates a overlay network between the DHT peers. With a *lookup*(k) operation a peer can find the peer responsible for data associated with a key k. In order to locate the peer responsible for key k, a peer forwards the lookup request to another peer whose identifier is closer to k according to a distance function. All peers maintain links to a subset of the other peers, thus forming an overlay network. A lookup request is forwarded by peers until no peer is found with an identifier closer to the requested key [16].

There are many different concepts of DHTs. Some of the most populars are Kademlia [17], Chord [18] and Pastry [19]. All of them have a wide range of derivatives and implementations in several programming languages. In the case of Bitsquare the Java library TomP2P is used which is an implementation of the Kademlia concept.

A fundamental component of Bitsquare's decentralized design is the offer book that is based on a distributed hash table (DHT).

2.4.1 TomP2P

TomP2P is an open-source, Java library that implements an advanced DHT, with an iterative routing protocol that is based on Kademlia. As in Kademlia the peerIDs and data keys are unique 160-bit identifiers and for routing the distance between ids is defined as a bit-wise exclusive or (XOR).

Chapter 3

Attack Scenarios & Countermeasures

In the Bitsquare Risk analysis document [20] various attack scenarios are discussed. Most of them are focused on how the trade protocol holds up against various external factors that come into play when it is combined with current payment methods. However, the vulnerability of the underlying distributed hash table is not explored in depth. This chapter gives an overview of possible attack scenarios, how they could be applied to Bitsquare and discusses possible approaches to mitigate them. Based on these findings an assessment is made on what possible countermeasure could be interesting to implement.

3.1 Trade Protocol related Attacks

Bitsquare Some of the described bitcoin exchanges in Section 2.3 rely on trusted third party to maintain funds for exchanges. This is not the case with Bitsquare. Instead, the implemented trade protocol allows the two trading parties to stay in control of the traded Bitcoins the whole time. This section examines what other possible weaknesses might get introduced with this trade protocol.

3.1.1 Bank Transfer Reversal

Most cases where buyer or seller deviate from the standard trade process (e.g. buyer doesn't respond or transfer the money) can be resolved through an arbitrator system. However there are few scenarios that might happen outside of a completed trade process that could be used to defraud the seller. A fundamental difference between bitcoin transactions and common electronic money transfers lies in its irreversibility. Whereas a transaction on the blockchain with a few confirming blocks will be valid forever, in most banking payment systems there exists the possibility to revoke a payment (e.g. due to indications of fraudulent behaviour).

Use-case

In a exchange where bitcoins are traded for a electronic money transfer this fact might be exploited by the buyer of bitcoin. For instance by requesting a reversal of the bank transaction with a convincing argument for fraudulent behaviour the buyer might be able to regain the money he initially paid to the seller after he has already received the bitcoins from the payout transaction. The bitcoin seller on the other hand doesn't have any possible response at his disposal to regain the transferred bitcoins without the buyers cooperation. Also the arbitrator has no control over the transferred bitcoins as the multisig-lock was already released with the payout transaction when the initial arrival of payment was confirmed by the seller.

Severity and Impact

The severity of this scenario highly depends on the payment method that is used and how request for transfer reversals are handled by the respective payment processor are handled.

Difficulty to mitigate

As the risk for transfer reversal or charge backs mostly depends on the used payment method, Bitsquare plans to only support a limited choice of payment methods that are known to be less vulnerable to this kind of fraud. A technical possibility to decrease the risk of charge back that has already partly been implemented is the introduction of time locked payout transactions. As described in Section 2.1.1 transactions can be prevented to be seen as valid for a certain amount of time. In this uses case the payout transaction that unlocks the bitcoins from the multisig adddress can be time locked to a point of time in the future where a transaction reversal of the payment is unfeasible or the risk thereof much reduced. This assumes for instance that a reversal of a SEPA transaction is very difficult to accomplish for a malicious buyer. In case the payment is revoked before the time lock it would still be possible to generate another payout transaction in cooperation with the arbitrator to transfer the bitcoins back to the seller. As this payout transaction without time lock would be included in the blockchain within a narrow time frame it would invalidate the initial one. This precaution obviously has the downside that it prolongs the processing time for the buyer. The length of such a time lock would have to be determined by the seller depending on the payment mechanism.

3.2 DHT related Attacks

Urdaneta et al. published an extensive survey on various DHT attack scenarios and security techniques [16]. The most common discussed attacks are the sybil and eclipse attacks which exploit the fact that the assignment of the peerID in most DHT implementation

can be controlled haphazardly. In regard of Bitsquare's use of a Kademlia-like DHT implementation for the offer book especially Kademlia specific attacks are relevant.

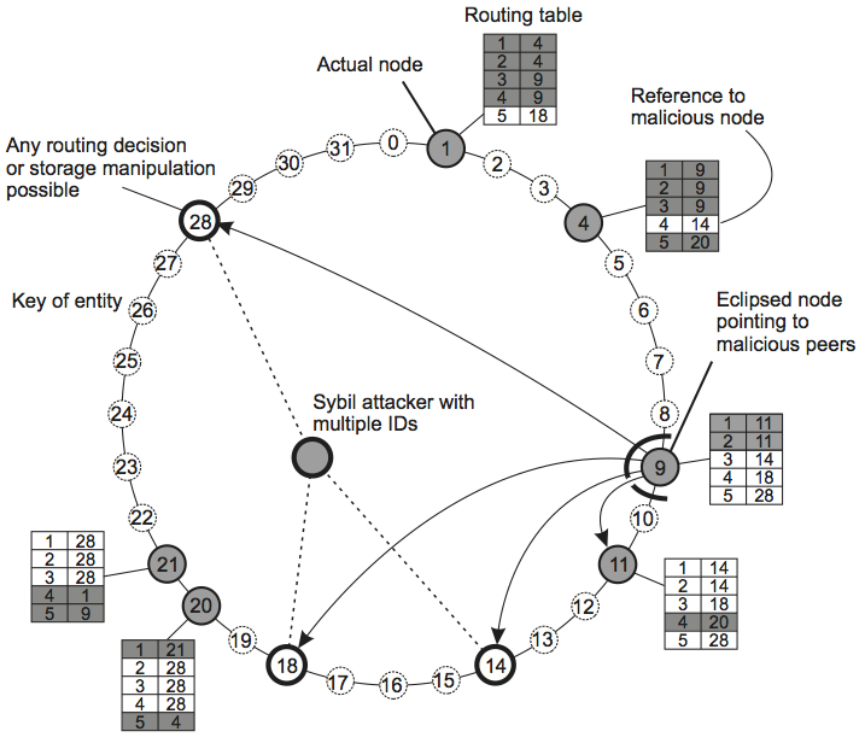

Figure 3.1: The organization of a typical DHT, illustrating attacks on the core functionality. [16]

3.2.1 The Sybil Attack

Due to the decentralized nature of a Kademlia-like DHT there exist no verifiable link between a participating entity (a user or machine) and its identity (peerID) there is nothing holding a malicious user back from joining the network with multiple identities [21]. This characteristic is shared by many P2P systems and the attack procedure of introducing a lager number of bogus peers into a network was first described by Douceur and named the Sybil attack [22].

Use-case

In the context of the Bitsquare offer book a sybil attack could be used to fill the DHT with bogus offers. Such a behaviour could be sourced in various motivations. For instance a seller might want to fabricate higher buy offers to put his selling offer in a better position. Another thinkable motivation for a competitor would be to obstruct the trading process on this platform in favour of competing markets.

Severity and Impact

The offer book is a central part of the Bitsquare concept as any buy or sell decision is based on information in the offer book. Any attack that undermines the integrity of this information has negative implications on the effective workings of the Bitsquare exchange.

Difficulty to mitigate

Bitsquare counters the use case of bogus offers or spam in the offer book with the deposit transaction. Clients can verify if a data entry in the DHT that represents an offer is a earnest offer where the offerer has already locked a specific amount of bitcoins. This however still doesn't prevent the storage of other irrelevant data on the DHT which could lead to performance issues when clients receive a large amount of bogus offer data that they have to filter.

Since the Kademlia paper has been published in 2002, there has been a wide range of proposals on how to make it more secure against sybil attacks. Castro et al. [23] propose a trusted third party (TTP) certified identifier assignment that prevents the random assignment of peerIDs. Also Maccari et al. [24] in their paper analyse possible counter measure for sybil attacks in a DHT network and in their conclusion propose a centralized certifications service. Caubet et al. [25] and [26] present an enhanced approach based on implicit certificates that speeds up the registration and verification process. All this approaches however rely on a central entity. Wang et al. [27] propose a peerID assignment based on network coordinates and characteristics so to restrict the random assignment of peerIDs. This however binds a peerID to a specific network location and e.g. is not applicable to use cases with mobile clients. Yu et al. [28] in his proposed solution instead uses social networks to bind peerIDs to users as real entities. Alternative solution that follow the distributed design of the DHT are proposed by Borisov [29] who introduced a proof of work (computational puzzle) mechanism to the generation of peerIDs. Similar to the implementation of OpenBazaar this increases the costs for launching attacks with many or specific peerIDs. If the underlying proof-of-work mechanism stays the same over a long period specific peerIDs can still be generated with a brute-force attack. The difficulty of the computational puzzle might also not be strong enough for attackers with large resources and at the same time still manageable for honest users on a light client.

The Kademlia implementation of TomP2P currently doesn't offer any kind of direct measures to counter sybil attacks.

3.2.2 The Eclipse Attack

With an eclipse attack the attacker tries to corrupt the routing tables of honest peers by filling them with references to malicious peers [24].

This attack tries to place adversarial peers in the network in a way that one or more peers are cut off from it, i.e. all messages are routed over at least one adversarial peer. This

gives the attacker the control over a part of the overlay network. Thus the Eclipse attack can "hide" some peers from the overlay network

When the eclipse attack is targeted against the stored contents on DHT, making them inaccessible to lookups, then it is known as peer insertion attack: a vast number of peers marked with identifiers numerically close to the key k of the target content are initiated, receiving the most lookup requests for k and answering with bogus contents or ignoring them completely, effectively hiding the content.

Use-case

A malicious user could try to prevent competing offers to be communicated to other users in the offer book. He would have to identify the peer where a specific, competing offer is stored. Executing an eclipse attack by inserting a few malicious peers enclosing this target requests for offers stored in this key space could be ignored. Thus it would appear to other users as this offer would not exist and might persuade them to accept his offer with a higher asking price.

Severity and Impact

If the data the offer book consist of can be withheld or tampered with it violates one of the core requirements of an offer book: data integrity. This vulnerability could be used for market manipulation and when detected by other users lead to a loss of confidence in the offer book and the Bitsquare market in general.

Difficulty to mitigate

Singh et al. propose an anonymous auditing technique [30] that would prevent bogus entries in the routing table. It however does not protect from attacks where the goal is to block access to data stored on certain peers. Similar to the Sybil attack the eclipse attacks can effectively take place only when attacker peers are able to assign their own peerIDs without restrictions. Therefore the discussed countermeasures for the Sybil attack that focus on restrictions for the peerID assignment would also be valid for the eclipse attack.

3.3 Assessment

Protocol related weaknesses might become apparent when the Bitsquare network will be actually used and certain payment options might have to be prohibited as there is a high risk of transfer reversal.

The two given use cases demonstrate that the offer book that relies on a DHT is very open to sybil and eclipse attacks. The offer book signifies a vital part of Bitsquare and any manipulation of the availability of offer entries could lead to distortions of the market price. A total breakdown of the DHT would render the Bitsquare client useless.

Chapter 4

Design & Implementation

As the previous analysis showed that DHT vulnerabilities could harm the vital integrity of the Bitsquare offer book, the focus on possible countermeasures was laid on the underlying DHT implementation. The fundamental problem, as various studies seem to agree on [16] [31] [32], lies in the insecure assignment of peer identifiers (peerIDs). Proposed countermeasures entailing the introduction of a central registration services or a new trusted third party don't seem to be appropriate options for the use case of Bitsquare as they introduce another trusted third party and hence invalidate its other efforts towards decentralisation. A possible solution is to implement a decentralized peerID registration service by reusing key mechanisms the Bitsquare project already relies on: the Bitcoin blockchain.

The first part of this chapter introduces an alternative mechanism for a secure peerID assignment that relies on a registration services based on the Bitcoin blockchain. While the second part documents an implementation of this peerID registration and verification mechanism as an additional package of the TomP2P library.

4.1 Design

Inspired by the bank account registration process (see Section 2.2.3) where the Bitsquare project made use of the blockchain as a public registry where everyone an can publish small data entries, it was standing to reason to try applying this idea also to a registration service for the underlying DHT network. The concept for this registration system has been described in the Bitsquare white paper. Parts of it have been implemented in the current version of Bitsquare.

For the purpose of registering a peerID in the blockchain, additional mechanisms were needed. Previous analysis on how to prevent sybil and eclipse attacks showed that a viable solution for a decentralized peerID registration not only had to be reliant in the sense of data integrity but a more critical factor is that it should not allow a user to influence the resulting peerID that would be registered in any way. In other words the resulting peerID would have to be random.

It is convenient that the blockchain can also be used as a source or randomness. Approximately every 10 minutes a new block is generated and with it also a block nonce. The mining process that generates this nonce was made unpredictable by design so that a huge amount of computational resources have to put into this process to find a correct nonce according to the set restrictions. It can therefore be argued that the resulting nonce inside this restriction is random. If it weren't, there would be a shortcut for mining a block that would be exploited by miners immediately.

4.1.1 General Process

The resulting registration and request verification process look as follows:

- For a joining peer a private/public keypair is generated locally. This keypair is the basis of a peerID and also reference to a registration. Among other things it will be used to sign outgoing request as a way to proof the peer's identity.
- to register the public key a valid bitcoin transaction with an output that contains the public key with a OP_RETURN statement is created and published to the Bitcoin network.
- After the transaction is included in a block the peerID can be generated as a hash, based on the public key and as a seed for randomness the nonce of the block the transaction was included in.
- When connecting to other peers with a signed request they also are informed of the block and transaction id that the peerID is based on so they can verify that the peerID is valid by looking up the respective nonce and the transaction containing the public key in the blockchain. (They wouldn't have to download the whole blockchain. Asking for the specific block and verifying it as a SPV client should be enough.)

4.1.2 Limitations

As a consequence of introducing the bitcoin blockchain as a dependency we might also introduce security vulnerabilities thereof. This solution bypasses the necessity of trusting a third party as a registration service but instead trusts in the correctness of the bitcoin network. Theoretically miners could be bribed to hold back blocks with undesirable nonce. This would however mean that they loose out on the mining rewards of currently 25 BTC and additional fees. A bribe that would compensate these losses would therefore be costly.

Another concern might be that it is possible that several registrations are based on the same block nonce when their transactions end up in the same block. This can increase the probability of generating a specific peerID when a large amount of registration is created that are included in the same block. The Block size however is currently limited to 1MB which would make such an attack infeasible. A general disadvantage of the registration

process is that it takes some time for the registration transaction to be included in the blockchain. The initial joining of the DHT network is therefore put off by at least 10 to 20 minutes. Compared to solutions where an application request a certificate from a central authority, this might still be acceptable. This waiting period will only be noticeable to the user before first usage. Identical to an implementation with a central registry the registration process has to be completed only once and might even be reused for several applications.

As the validity of a registration has to be checked for every incoming message a certain overhead to the current DHT implementation is created. The effect this will have on the performance of certain requests has to be determined.

Another interesting aspect of this mechanism is that the costs could even be increased additionally by setting other requirements on the bitcoin transaction for being the basis of a valid registration. E.g. one could ask for another output with a minimal amount of bitcoins to a certain address. This could be used as a registration fee that for example would fund the further development of a project such as Bitsquare.

4.2 Implementation

As a proof-of-concept this mechanism was implemented as part of the TomP2P library. This section describes the architecture of the integration in the TomP2P and explains the implementation of the registration and verification process in detail.

4.2.1 Architecture

The overall goal was to keep the implementation of a registration service independent of the core package without having to make too many adaptations in the core and other packages. To encapsulate the dependency on other libraries a new package *bitcoin* was introduced. Figure 4.1 gives an overview of its contained classes. The processes of registration and verification is abstracted with a Registration Interface, which allowed to keep the modifications in the core package general and support also other implementation of registration service (e.g. central registration server). To make it possible for peers to verify that incoming requests are from a registered peer, the Message class had to be extended with an optional header extension. This extension allows to add additional registration reference data to any message. Not only messages should be able to be verified but also the list of neighbour peers should only consist of registered peers. For this purpose the *PeerAddress* class was also adapted to hold extra information about the registration of that peer.

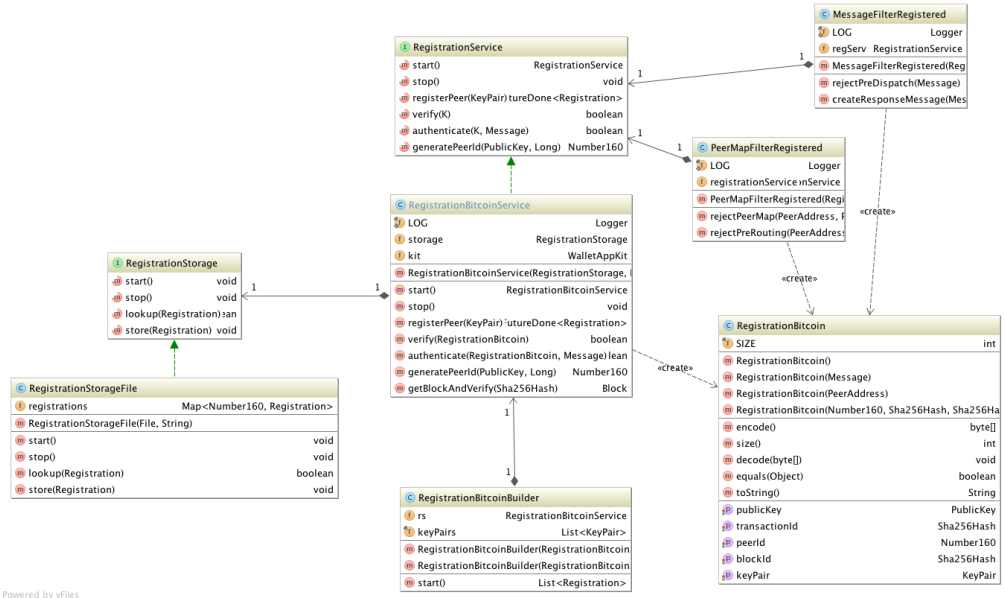

Figure 4.1: UML class diagram of the TomP2P bitcon package

To implement the behaviour of only accepting incoming request that originate from a registered peer a *MessageFilter* interface was introduced to the core package. The class *MessageFilteredRegistered* implements this interface and only allows request from peers that were checked successfully for a valid registration. The same concept was also applied to filter instances of *PeerAddresse* that are added to the *PeerMap*. In this case the interface *PeerMapFlter* already existed in the core package.

4.2.2 Tools

To communicate with the Bitcoin network and to provide an easy way to create and manage a bitcoin wallet the bitcoinJ library is included as a dependency in the TomP2P bitcoin package. The bitcoinJ library offers a easy solution for setting up a SPV wallet and connect it to the bitcoin network. It implements all of the protocol commands that were relevant for the purpose of this project. For instance it simplified the process of setting up a lean SPV wallet based only on the blocks headers (see Section 2.1.3). At the same time it also supported download requests to the Bitcoin network for specific blocks in full detail. During the development a variety of JUnit tests were used to test and simulate the registration and verification process of a set of peers. A simple OP_RETURN transaction usually doesn't inculde a high amount of bitcoins. However when testing in a larger scale the transaction fees needed to generate hundreds of peerIDs can accumulate to a some amount. For this reason a test version of the bitcoin network was used, the TestNet. The bitcoinJ library also made this easy. A simple setting would change the mode it operated in from MainNet to TestNet. RegTest, a mode where the network is run locally and block mining can be simulated, also helped speeding up the development process by allowing faster testing as it doesn't depend on the standard block confirmation time.

4.2.3 Registration

The registration process is initiated with the *RegistrationBitcoinBuilder*. The main part is however implemented in the *registerPeer* method of the RegistrationService (see Listing 4.1 at the end of this Chapter).

The builder abstracts the process for generating the registration object that then can be used to setup a Peer with a slightly adapted implementation of the *PeerBuilder*. This builder pattern is also used in many other occasions in the TomP2P library. The *registerPeer* method takes a generated private/public keypair and then creates a data output (OP_RETURN) Bitcoin transaction with the public key as the payload. The transaction is then sent to the bitcoin network and a listener is set for the successful confirmation in the blockchain. Once the transaction has been included in a block the bitcoin client request the download of this full block to learn the block nonce. The actual generation of the peerID is implemented in a separate method as it is also used in the verification process. The current implementation bases the peerID on the public key and the block nonce from the block the registration transaction was included in. As a first step the public key is hashed, identical to the previous standard implementation in TomP2P. As a second step, to prevent the generation of arbitrary peerIDs, the block nonce is appended and the whole expression hashed again to 160 bits.

```java
/**
 * Generates peerId by hashing the public key first, then appending the block nonce and hash it again.
 *
 * @return generated peerId as Number160
 */
@Override
public Number160 generatePeerId(PublicKey publicKey, Long blockNonce) {
    // initialize peerId with SHAHash of public key
    Number160 peerId = Utils.makeSHAHash(publicKey.getEncoded());
    byte[] peerIdBytes = peerId.toByteArray();
    byte[] blockNonceBytes = ByteBuffer.allocate(8).putLong(blockNonce).array();
    // for the seed create a new array that is the size of the two arrays
    byte[] seed = new byte[peerIdBytes.length + blockNonceBytes.length];
    // copy pub key into the seed array (from pos 0, copy pubKeyEnc.length bytes)
    System.arraycopy(peerIdBytes, 0, seed, 0, peerIdBytes.length);
    // copy block nonce into end of seed array (from pos pubKeyEnc.length, copy blockNonceBytes.length bytes)
    System.arraycopy(blockNonceBytes, 0, seed, peerIdBytes.length, blockNonceBytes.length);
    // generate final peerId by hashing the combined byte array
    peerId = Utils.makeSHAHash(seed);
    return peerId;
}
```

Listing 4.2: *generatePeerId* method in *RegistrationBitcoinService*

When the peerID has been generated the registration object is completed with the reference to the block and the transaction (see Listing 4.2). By adding this information in a header extension of every outgoing message, other peers are able to verify that this message originates from a registered peer. The same information is also added to the *PeerAddress* object that captures how to contact a specific neighbouring peer.

4.2.4 Verification & Authentication

In order to enforce that all peers generate their peerID randomly based on a block nonce all incoming messages have to be checked that they originate from a complying peer. Messages that don't correspond to an existing registration are ignored.

```
 /**
 * Validates if the supplied peerId is based on public key in the transaction and the block nonce
 * @return true if verification was successful
 */
@Override
public boolean verify(RegistrationBitcoin registration) {
    //check local registration storage if registration was already verified
    if(storage.lookup(registration)) {
        return true;
    }
    //else verify registration on blockchain
    Number160 peerId = registration.getPeerId();
    Sha256Hash blockHash = registration.getBlockId();
    Sha256Hash transactionHash = registration.getTransactionId();
    Block b = null;
    // asking bitcoin peer for block
    Peer peer = kit.peerGroup().getConnectedPeers().get(0);
    b = getBlockAndVerify(blockHash);

    for(Transaction tx : b.getTransactions()) {
        if (tx.getHash().equals(transactionHash)) {
            for(TransactionOutput output : tx.getOutputs()) {
                if (output.getScriptPubKey().isOpReturn()) {
                    byte[] pubKeyEncoded = output.getScriptPubKey().getChunks().get(1).data;
                    X509EncodedKeySpec pubKeySpec = new X509EncodedKeySpec(pubKeyEncoded);
                    KeyFactory keyFactory = null;
                    try {
                        keyFactory = KeyFactory.getInstance("DSA");
                        PublicKey pubKey = keyFactory.generatePublic(pubKeySpec);
                        //check if peerId is based on block nonce and public key
                        if (generatePeerId(pubKey, b.getNonce()).equals(peerId)) {
                            //set public key from transaction data and store registration for later reference
                            registration.setPublicKey(pubKey);
                            storage.store(registration);
                            return true;
                        }
                    } catch (Exception e) {
                        e.printStackTrace();
                        return false;
                    }
                }
            }
            break;
        }
    }
    return false;
}
```

Listing 4.3: *verify* method in *RegistrationBitcoinService*

This behaviour had to be integrated in the core implementation of TomP2P. To do this a new interface for message filtering was introduced. These filters can be added to the message dispatcher when setting up the TomP2P configuration. For the application of only allowing messages from registered peers the *MessageFilterRegistered* class relies on the *verify* method of the RegistrationService (see Listing 4.3).

With the current implementation all incoming messages except those of the type Ping and Neighbor are checked for a verified registration. The reason for this exception is that those two types of messages do currently not support signatures and could therefore be spoofed. As they do not pose a critical risk for manipulating data entries this checks can be disregarded. The *verify* method will first check if the referenced registration has been verified before and is still stored locally. If this is not the case it will download the referenced block from the blockchain and check if there exists a transaction with a registered public key. Based on the information from the blockchain the peerID generation will be executed again and the full registration information is stored locally for later reference.

After the registration object has been retrieved successfully, either from the blockchain or in the local storage form previous lookups, the next step is to authenticate the message with that registration. A simple method (see Listing 4.4) check if the message was signed correctly and if the signature corresponds to the public key of the registration.

This step is important, leaving it out would allow malicious peers to hijack the registration and the peerID from other users for their own messages. The signature of the message assures that only the user who had access to the registration's corresponding private key was able to create this message.

```
/**
 * authenticates if registration corresponds to public key in message
 * @return true if authentic
 */
@Override
public boolean authenticate(RegistrationBitcoin registration, Message message) {
    PublicKey publicKey = registration.getPublicKey();
    //check if public key exists, message signature was verified and public key corresponds to registration
    return publicKey != null && message.verified() && publicKey.equals(message.publicKey(0));
}
```

Listing 4.4: *authenticate* method in *RegistrationBitcoinService*

```java
/**
 * Registers public key with a transaction in the blockchain.
 * @param keyPair keyPair with public key for peer registration
 */
@Override
public FutureDone<Registration> registerPeer(final KeyPair keyPair) {
    final FutureDone<Registration> registrationFuture = new FutureDone<Registration>();
    Number160 peerId = null;
    final RegistrationBitcoin registration = new RegistrationBitcoin();
    registration.setKeyPair(keyPair);
    Coin value = Coin.parseCoin("0.00001");

    //check for sufficient funds in wallet
    if(kit.wallet().getBalance().isLessThan(value)) {
        ListenableFuture<Coin> balanceFuture = kit.wallet().getBalanceFuture(value, Wallet.BalanceType.AVAILABLE);
        FutureCallback<Coin> callback = new FutureCallback<Coin>() {
            public void onSuccess(Coin balance) {
            }
            public void onFailure(Throwable t) {
            }
        };
        Futures.addCallback(balanceFuture, callback);
        try {
            balanceFuture.get(); // blocks until funds are sufficient
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }

    //create transaction with public key in output script
    Transaction tx = new Transaction(TestNet3Params.get());
    Number160 pubKeyHash = net.tomp2p.utils.Utils.makeSHAHash(keyPair.getPublic().getEncoded());
    Script script = new ScriptBuilder().op(OP_RETURN).data(keyPair.getPublic().getEncoded()).build();
    tx.addOutput(Coin.ZERO, script);

    //broadcast transaction
    Wallet.SendRequest sendRequest = Wallet.SendRequest.forTx(tx);
    try {
        kit.wallet().sendCoins(sendRequest);
    } catch (InsufficientMoneyException e) {
    }
    registration.setTransactionId(tx.getHash());

    //add Listener for when transaction is included in blockchain
    tx.getConfidence().addEventListener(new TransactionConfidence.Listener() {
        @Override
        public void onConfidenceChanged(Transaction tx, ChangeReason reason) {
            TransactionConfidence confidence = tx.getConfidence();
            if (reason.equals(ChangeReason.TYPE) && confidence.getConfidenceType().equals(TransactionConfidence.ConfidenceType.BUILDING)) {
                Peer peer = kit.peerGroup().getConnectedPeers().get(0);
                List<Peer> peers = kit.peerGroup().getPendingPeers();
                Sha256Hash blockHash = null;
                for (Sha256Hash hash : tx.getAppearsInHashes().keySet()) blockHash = hash;
                registration.setBlockId(blockHash);
                try {
                    //download the full block where transaction was included

                    Block block  = getBlockAndVerify(blockHash);
                    //generate peerId
                    Number160 peerId = generatePeerId(keyPair.getPublic(), block.getNonce());
                    registration.setPeerId(peerId);
                    registrationFuture.done(registration);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
    });
    return registrationFuture;
}
```

Listing 4.1: *registerPeer* method in *RegistrationBitcoinService*

Chapter 5

Evaluation

This chapter describes the evaluation of the implemented peerID registration mechanism.

5.1 Goal

The presented proof-of-concept introduces an additional communication layer with the Bitcoin network for generating and verifying peerID registrations. As one of the shortcomings of a DHT compared to centralized solutions can be inferior performance, the repercussions of this introduced overhead are of importance. The goal of this evaluation therefore is to analyze the performance impact of verifying every incoming message for the validity of the originating peer's registration.

5.2 Method

To measure the performance of various operations the Java Microbenchmark Harness (JMH) [33], a benchmarking tool that is part of the OpenJDK code tools project, was used. It allowed to define several use cases and partial procedures of the verification process as methods that then were executed repeatedly to estimate its average execution time.

Starting with a test case of a simple put operation, a DHT with 100 nodes was simulated locally. A put operation in this case consists of a random node that requests to store some data in the DHT. Based on the hash of the data through a routing process the responsible node for storing the data is identified and the formal message with the request to store the data is sent to this node.

Two implementations were compared: a standard TomP2P implementation (*testPutStandard*) and an extended one where the registration and verification mechanism from the bitcoin package is used (*testPutRegistered*). For the extended implementation this means that

the node that is responsible for the storage of this data will also have to verify the registration of the requesting node.

To setup a DHT with 100 registered nodes, registrations were generated on the Bitcoin TestNet in advanced and stored in a file locally. The caching mechanism that would prevent downloading blocks again for previously verified registrations was disabled for this test to simulated the worst case where for every request a new registration has to be verified.

To visualize how the total performance overhead for a put operation is distributed among the different steps, the average time for the verification ($testVerify$) and also the download of a block ($testBlockDownload$) were also measured separately.

This scenarios where defined as methods which then were repeatedly executed by the JMH framework in an uninterrupted sequence of 1000 milliseconds. The average was calculated based on a sample of 100 testing iterations with a previous warmup iteration.

5.3 Results

The results make apparent that the verification of the registration creates an overhead on a put operation by factor 6. Table 5.3 lists the average measured operation times of the four tested execution scenarios. Figure 5.1 visualizes that the major performance decrease is due to the block downloading process. The $testPutRegistered$ method can be seen as a combination of $testBlockDownload$ and $testVerify$.

Benchmark	Mode	Samples	Score	Error (99.9%)	Unit
testPutNormal	avgt	100	48.792974	19.756872	ms/op
testPutRegistered	avgt	100	317.000443	96.398636	ms/op
testBlockDownload	avgt	100	210.654112	6.555298	ms/op
testVerify	avgt	100	106.256647	8.550205	ms/op

Table 5.1: Benchmark results

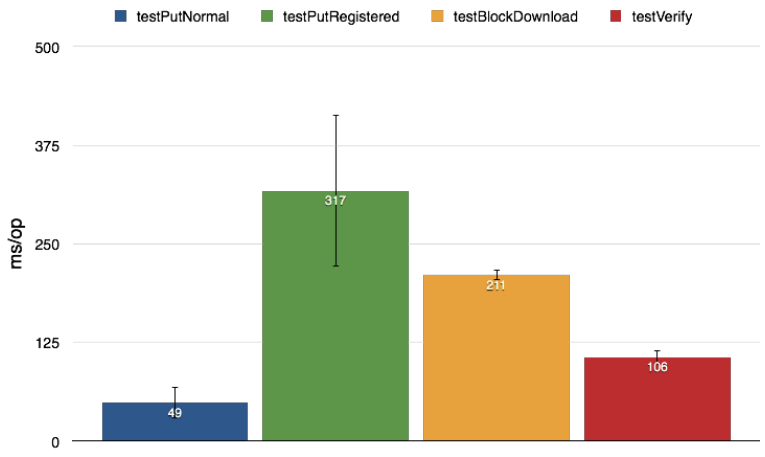

Figure 5.1: Benchmark results

The distributed nature of the verification process attributes to some factors that could not be controlled completely when measuring the performance: For one the download process can be affected by possible latency of the Internet connection or the current workload on the bitcoin network.

The uninterrupted continuous execution of the put operation in some cases seemed to have caused some ”Channel timeout for channel Sender” warnings and premature termination of the execution. This might partly explain the noticeable variance in the results for the two variants of the put operation.

The results show that a realistic application of the mechanism highly depends on the performance of the block download. The simulated scenario represents the worst-case scenario. In an usual setting with a repeated request from a previously verified peer, the response time would shrink to the equivalent of the $testVerify$ method.

Chapter 6

Summary

This final chapter summarizes the approach and findings of this thesis. Finally an outlook on future developments in the bitcoin system and its ramifications for the proposed solution are given.

6.1 Summary & Conclusion

This thesis introduced the decentralized bitcoin exchange project Bitsquare. Understanding its inner workings, the Bitcoin protocol and the DHT it depends on, allowed to analyze potential vulnerabilities to malicious actors. After recognizing the vital role of the DHT based offer book and the potential attack scenarios this allows the focus was laid on proposing a possible countermeasure in the underlying DHT implementation. It was explained how a peerID registration service would mitigate the identified sybil and eclipse attack on the DHT. With the design of a decentralized registry service based on the Bitcoin blockchain a viable solution was proposed that doesn't contravene Bitsquare's efforts at decentralization. An evaluation of the effects on performance when introducing registration verification on every received message in the DHT network was made. The benchmarking results showed that the created overhead might still lay in a range acceptable for a DHT application with moderate data alteration. In the worst case scenario where a data storage request is based on an previously unknown registration an overhead by factor 6 was measured.

6.2 Future Work

The proposed countermeasure implementation relies on the Bitcoin protocol. The directions the development of the Bitcoin protocol will take are unpredictable and with that also its application for this purpose. The performance of the registration verification mostly depends on the response time of the Bitcoin network for block download requests.

With a growing adaption of the Bitcoin network its current limited block size has been questioned for the gain of higher transaction bandwidth [34]. A bigger block size would intensify the performance bottle neck of downloading blocks to verify registrations additionally. A possible remedy could be a more efficient handling of download requests that ask for filtered blocks, only consisting of the most relevant transactions. The Bitcoin Improvement Protocol 037 (BIP 0037) [35] introduced connection Bloom filtering and is partly implemented in the bitcoinJ library. The abstraction in the architecture of the library however favours the use case for supporting fast synchronisation of SPV wallets. For the implementation of an accelerated lookup of a single transaction's content the implementation would need some profound modifications to adapt it to this use case at hand.

The current implementation also doesn't take in account all special cases connected to the blockchain behaviour. The nature of the blockchain allows of certain situations where an initial confirmation in one block is invalidated by an alternative branch of the blockchain that outgrows the initial version. This would also invalidate an initial registration as the relevant block nonce would be changed. The current implementation for simplicity only waits for one block confirmation. A more robust implementation would have to wait for more block confirmation before determining the peerID or track the blockchain development and change the assigned peerID if necessary.

The future development of the Bitcoin protocol might not continue supporting its use for registry lookups. There are many alternative modifications of the Bitcoin protocol that might be better suited for this purpose. One of the firs forks of the Bitcoin project, called NameCoin follows the idea of a decentralized registry that could potentially substitute the DNS system. Projects like NameCoin might with their focus on the registry application might be more efficient for a use case such as Bitsquare. As the Bitsquare project already made use of the bitcoinJ library and currently no equivalent Java library for NameCoin is available, an implementation based on NameCoin was not pursued. Depending on the evolution of the Bitcoin protocol this decisions might have to be reevaluated in the future.

The combination of a P2P system such as a DHT with a frictionless payment network like Bitcoin also could potentially open up new models of funding independent, non-hierarchical development of P2P projects. As mentioned in Section 4.1.2 additional checks on the registration transaction could be implemented easily to enforce a registration fee as a source of funding.

Bibliography

[1] *OpenBazaar Website*. URL: https://openbazaar.org (visited on 10/07/2015).

[2] *Bitsquare Website*. URL: http://bitsquare.io (visited on 10/07/2015).

[3] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2009. URL: https://bitcoin.org/bitcoin.pdf.

[4] Andreas M. Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. English. Cambridge, MA: O'Reilly Media, 2014.

[5] Adam Back. *Hashcash - A Denial of Service Counter-Measure*. 2002. URL: http://hashcash.org/papers/hashcash.pdf.

[6] *blockchain.info: Blockchain Size*. URL: https://blockchain.info/charts/blocks-size (visited on 29/07/2015).

[7] Rainer Böhme et al. 'Bitcoin Design Principles Enabling Technologies and Processes'. In: *Journal of Economic Perspectives* 29.2 (2015), pp. 213–238.

[8] Tyler Moore and Nicolas Christin. 'Beware the middleman: Empirical analysis of Bitcoin-exchange risk'. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7859 LNCS.June 2011 (2013), pp. 25–33.

[9] Rachel Abrams, Matthew Goldstein and Hiroko Tabuchi. *Erosion of Faith Was Death Knell for Mt. Gox*. Feb. 2014. URL: http://dealbook.nytimes.com/2014/02/28/mt-gox-files-for-bankruptcy/.

[10] *Bitsquare White Paper Version 1.0*. URL: https://bitsquare.io/bitsquare.pdf.

[11] *localBitcoins.com Website*. URL: https://localbitcoins.com (visited on 10/07/2015).

[12] Chris Pacia. *OpenBazar GitHub repository commit comment*. URL: https://github.com/OpenBazaar/Network/commit/ece23e2853c1ddb771b6afe9d54ef9d1a1086609 (visited on 10/07/2015).

[13] *Coinffeine Website*. URL: http://coinffeine.com/ (visited on 10/07/2015).

[14] *Coinffeine Exchange Algorithm*. URL: https://github.com/Coinffeine/coinffeine/wiki/Exchange-algorithm\#protocol-overview (visited on 10/07/2015).

[15] *Mercury, the decentralized cryptocurrency exchange.* URL: http://mercuryex.com/ (visited on 10/07/2015).

[16] Guido Urdaneta, Guillaume Pierre and Maarten Van Steen. 'A survey of DHT security techniques'. In: *ACM Computing Surveys* 43.2 (2011), pp. 1–49.

[17] Petar Maymounkov and D Mazieres. 'Kademlia: A peer-to-peer information system based on the xor metric'. In: *First International Workshop on Peer-to-Peer Systems* (2002), pp. 53–65. URL: http://link.springer.com/chapter/10.1007/3-540-45748-8_5.

[18] Ion Stoica et al. 'Chord: A scalable peer-to-peer lookup service for internet applications'. In: *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications - SIGCOMM '01* (2001), pp. 149–160. URL: http://portal.acm.org/citation.cfm?doid=383059.383071.

[19] Antony Rowstron and Peter Druschel. 'Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems'. In: *Middleware 2001*. Ed. by Rachid Guerraoui. Vol. 2218. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2001, pp. 329–350. URL: http://www.springerlink.com/index/10.1007/3-540-45518-3.

[20] *Bitsquare Risk analysis.* URL: https://bitsquare.io/risk_analysis.pdf (visited on 10/07/2015).

[21] Luca Maria Aiello et al. 'Tempering kademlia with a robust identity based system'. In: *Proceedings - P2P'08, 8th International Conference on Peer-to-Peer Computing* (2008), pp. 30–39.

[22] JR Douceur. 'The sybil attack'. In: *Peer-to-peer Systems* (2002), pp. 1–6. URL: http://link.springer.com/chapter/10.1007/3-540-45748-8_24.

[23] M Castro et al. 'Secure routing for structured peer-to-peer overlay networks'. In: *Proc. OSDI 2002, Boston, MA, December* (2002), pp. 299–314.

[24] Leonardo Maccari et al. 'Avoiding Eclipse attacks on Kad / Kademlia : an identity based approach'. In: *ICC '09: Proceedings of Communication and Information Systems Security Symposium* (2006).

[25] Juan Caubet et al. 'Securing identity assignment using implicit certificates in P2P overlays'. In: *Trust Management VII*. 2013, pp. 151–165.

[26] Juan Caubet et al. 'RIAPPA: a Robust Identity Assignment Protocol for P2P overlays'. In: *Security and Communication Networks* 7.12 (2014), pp. 2743–2760. URL: http://dx.doi.org/10.1002/sec.956.

[27] Peng Wang et al. 'Attacking the Kad network'. In: *Proceedings of the 4th international conference on Security and privacy in communication netowrks - SecureComm '08* (2008), p. 1. URL: http://portal.acm.org/citation.cfm?id=1460877.1460907.

[28] Haifeng Yu and Michael Kaminsky. 'SybilGuard: Defending against Sybil Attack via Social Networks'. In: *IEEE/CM Transactions on Networking* 16 (2008), pp. 576–589. URL: http://dl.acm.org/citation.cfm?id=1159945.

[29] N. Borisov. 'Computational Puzzles as Sybil Defenses'. In: *Sixth IEEE International Conference on Peer-to-Peer Computing (P2P'06)* (2006).

[30] Atui Singh et al. 'Eclipse attacks on overlay networks: Threats and defenses'. In: *Proceedings - IEEE INFOCOM*. 2006.

[31] Ingmar Baumgart and Sebastian Mies. 'S/Kademlia: A practicable approach towards secure key-based routing'. In: *Proceedings of the International Conference on Parallel and Distributed Systems - ICPADS* 2 (2007).

[32] Davide Cerri et al. 'ID mapping attacks in P2P networks'. In: *GLOBECOM - IEEE Global Telecommunications Conference* 3 (2005), pp. 1785–1790.

[33] *OpenJDK JMH Website*. URL: http://openjdk.java.net/projects/code-tools/jmh/ (visited on 10/07/2015).

[34] *Implementation of BIP 101 : maximum block size increase*. URL: https://github.com/bitcoin/bitcoin/pull/6341 (visited on 29/07/2015).

[35] *Bitcoin Wiki: BIP 0037*. URL: https://en.bitcoin.it/wiki/BIP_0037 (visited on 29/07/2015).

Abbreviations

BTC	Denomination of bitcoin, the unit of account of the Bitcoin p2p payment network
DHT	Distributed Hash Table
P2P	Peer-to-peer
TTP	Trusted Third Party
SEPA	Single Euro Payments Area
SPV	Simplified Payment Verification

List of Figures

List of Tables

List of Source Codes

Appendix A

Contents of the DVD

- MasterThesis.pdf
- Abstract.txt
- Zusfsg.txt
- Data of evaluation in *evaluation* directory
- Source Code of implementation in *code* directory
- Latex source of thesis in *latex* directory