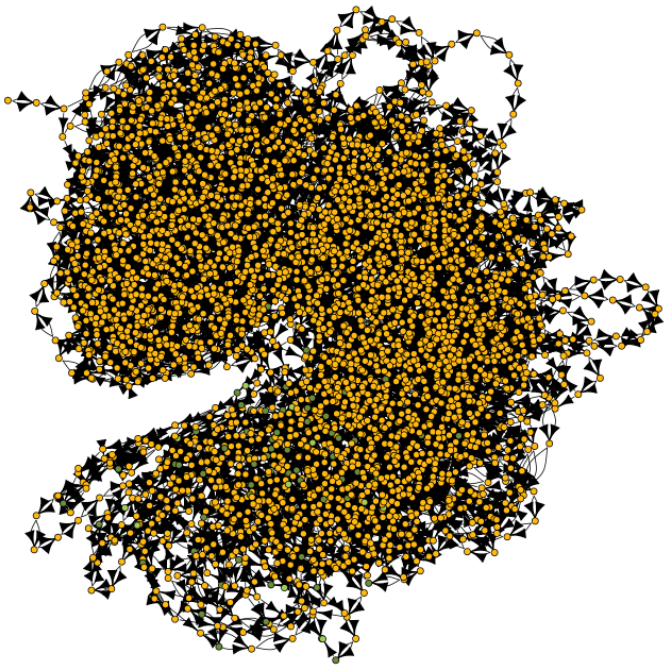
# Leveraging blockchains to establish cooperation

Pim Veldhuisen



Delft University of Technology

# Leveraging blockchains to establish cooperation

Master's Thesis in Computer Engineering

Distributed Systems group – Blockchain Lab
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Pim Veldhuisen

5th May 2017

*Cover image: A visualisation of a series of blockchains recording interaction data.*

**Author**
Pim Veldhuisen

**Title**
Leveraging blockchains to establish cooperation

**MSc presentation**
12th May 2017

**Graduation Committee**

| | |
|---|---|
| Prof. Dr. C. Witteveen | Delft University of Technology |
| Dr. ir. F. Kuipers | Delft University of Technology |
| Dr. J. Pouwelse | Delft University of Technology |

**Abstract**

When people interact with each other, it is often fruitful to cooperate. Cooperation is however not always established naturally, since in many systems it is not beneficial for rational actors to cooperate without the assurance of reciprocation. In real-world scenarios authoritarian central parties or social pressure can be used to organise cooperation by giving some assurances of reciprocation. In open access systems on the Internet however, both these mechanisms fall short. There is thus a need for other mechanisms that can realise cooperation in such systems, which remains an unsolved problem today. A recent innovation that has opened up new avenues in many applications is blockchain technology. The blockchain is a new type of datastructure that can provide reliable and tamperproof storage of information in distributed environments. In this thesis we apply the principles of blockchains to the problem of realising cooperation by creating an incentive system called the Multichain. By recording historic behaviour using blockchains, we utilise the principle of the shadow of the future to incentivise cooperation. The steps towards a system that can use this idea to establish cooperation in BitTorrent networks are identified. The first step, consisting of the creation of records, is designed and implemented, taking several issues of performance and security into consideration. Furthermore it is deployed to real-world users and evaluated. The second step consists of the discovery of records by others in the network. For this purpose several algorithms are explored, showing that a focus on high reputation peers can improve the efficiency of the discovery process. Additionally a prototype version of the discovery process is implemented and tested in a distributed manner. The successful implementation and validation of these two steps show the utility of blockchains in distributed networks, and pave the way for realising behavioural change using the Multichain.

# Preface

When looking back at thousands of year of human history, it is alluring to see some trends, a steady line of progress, some goal everything is moving towards. While large parts of history seem random and indiscriminate, I see two movements that are prevalent throughout the ages: The first is the rise of technology, equipping humans with the tools we need to bend our environment to our will and increase our quality of life. The second is that of an increasing scale of cooperation. From small tribes of hunter-gatherers to villages of farmers. From villages to feudal realms, and from localised realms to nation states. The final step in this process is now upon us: from nation states to a global community. By the very principles of cooperation, this increase in scale of the groups that cooperate, increases the benefits yielded to everyone within the group. On the other hand, managing cooperation becomes more difficult the larger the group becomes. A tribe is able to manage its cooperation through social mechanisms, but a country cannot be run based on good faith alone. Luckily, advancing technology provides us with the means to manage more and more complex forms of cooperation. This thesis aims to continue this process, by using one of our newest forms of technology, the blockchain, to establish cooperation in large digital networks.

I have of course not completed my thesis on cooperation entirely alone. I would like to thank my supervisor Johan Pouwelse for providing inspiration and expertise, dispensing me with the framework for this research. I am grateful to Ewout and Pim for their work on the Multichain, providing feedback and ideas on the system. I would like to thank Ernst, Jesse, Otto, Stefan and Tim and all others on the 7th floor for making my main place of work fun, supportive and intellectually challenging. I would also like to thank Elric and Martijn for the above, but as lead developers of Tribler I also specifically appreciate their help in figuring out the many enigmas in the code base. I would also like to especially thank Stefan, Martijn and Otto for taking the time to proofread my work. Your suggestions have improved the quality of this text. Any and all remaining errors are of course my responsibility. I am grateful to my parents, who have supported me financially, emotionally and intellectually throughout my academic career. Lastly I want to thank my brother Max, for believing in me more than anyone else ever could.

Pim Veldhuisen
Delft, The Netherlands
5th May 2017

# “REMQUE ORDINE PANDO”

*“And then I presented everything in order”*

– Virg. Æn. III. 179

# Contents

# Chapter 1

# Introduction

The introduction of the Internet to the world in the nineties brought unprecedented opportunities for interaction and cooperation between people. However, without a powerful organisational structure, interacting people often act selfish and attempt to greedily pursue their own goals. This often means that cooperation grinds to a halt and the benefits of synergy are lost to all. A famous example of such behaviour is the prisoners dilemma, where game theory shows that fully rational individuals would not cooperate, even if it is in their best interest to do so. The optimal situation for *both* prisoners is to not betray each other, but each *individual* prisoner has an incentive to do exactly that. If each prisoner act rationally and betrays the other, both are worse off. This situation is a Nash equilibrium, i.e. a situation where no actor can improve his own gains by changing only his own actions. In general rational actors will often move systems to a Nash equilibrium, even when this is not the optimal scenario for the system as whole.
Scenarios where the Nash equilibrium does not correspond to the optimal outcome occur in many forms in the real world. One situation where each individual's pursuit of his own goals leads to the devaluation of the community as a whole is the sharing of a common good. In a 1968 Science article Garrett Hardin showed that unregulated use of a limited common good leads to a situation where everybody is worse off [1]. This is known as the tragedy of the commons. An example of a tragedy of the commons problem is that of climate change; while each country would benefit from a reduction of global greenhouse gas emissions, the reduction in an individual country will likely affect the local economy somewhat negatively. In the global picture, the benefits of limiting climate change will likely outweigh the economic costs. However, for each individual country there is an incentive *not* to reduce emissions, as this will be better for the local economy, while the environment will still be saved by the efforts of the other countries. However, if every country follows such a strategy, this will result in a Nash equilibrium where nobody reduces emissions, and everybody suffers from the disastrous consequences of climate change. A similar problem exists within the NATO alliance; since each coun-

try vows to protect each other, defence capabilities are basically pooled together. This gives small countries incentives to cut on the defence budget, since it will only marginally affect the total strength of the alliance. However, when all countries cut the budget, the strength becomes insufficient to guarantee security.

Early Internet applications that attempted to stimulate cooperation between Internet users were also affected by such problems. This is strongly apparent in peer-to-peer file sharing networks. A study from 2000 showed that almost 70% of users of the Gnutella network were free riders; meaning that they used the network to access files from other users, but did not contribute files themselves [2]. A similar experiment performed on the eDonkey network in 2004 identified 68% of the users as free riders [3]. These large percentages of peers that do not contribute to the network reduce the availability of scarce files, and the bandwidth with which popular files can be downloaded, thus reducing the utility of the network. The performance of the network could potentially be much higher when all peers contribute to the full extent of their capability.

One way to solve the problems with uncooperative users is to introduce a central party that regulates the users, enforcing certain behaviours. While this is an effective manner to create a more optimal mode of cooperation, it has its downsides. The main drawback of this concept is that it centralises power. While a benevolent dictator can often achieve great results, dictators can also create great injustice. There is always the question of who can be trusted with the power to regulate the users, and whether this entity will not abuse this power. Other disadvantages of centralisation are more practical; the central party can be a performance bottleneck and it forms a single point of failure. Considering all this, it is desirable to realise fruitful behaviour not through binding rules imposed by a centralised master node, but through a communal sense of duties and obligations towards peers. Unfortunately anyone who has used the Internet will know that one cannot simply rely on the gallant nature of all Internet users. Many users need the proper motivation to behave in a way that is beneficial for the system as a whole. This motivation can be created by setting up a system of rules that provides benefits for users that contribute to the operation of the network, and disadvantages to those who do not contribute. Such a set of rules is called an incentive scheme. Complicated incentive schemes are often hindered by the fact that people do not always behave fully rationally [4] [5]. In Internet applications however, people interact through the use of software programs. Software is generally excellent at behaving completely rational, while it often inhibits other mechanisms for cooperation, such as altruism. This makes the use of incentives schemes especially suitable for software applications.

BitTorrent is one of the most used systems with a successful decentralised incentive scheme [6]. It is based on a memory-less tit for tat principle, meaning peers allocate upload bandwidth to the peers it currently receives the most download bandwidth from. By this mechanism peers are incentivised to contribute upload

bandwidth, because this will likely also lead to receiving more download bandwidth. This protocol has shown to result in a functional network where many peers choose to cooperate. The system is however not watertight, and free riders are still able to realise significant download speeds [7]. One of the problems is that seeders, i.e. peers that already have the complete file, cannot use tit-for-tat to distinguish between contributors and free riders, since they are no longer in need of download bandwidth. Distributed global networks such as the BitTorrent system could still perform much better when all users cooperate. The key to realising this behaviour is a better distributed incentive scheme. A critical innovation that is needed is the incorporation of the history of peers within the decision making process. The Maze project attempted to do this by using a persistent score for each user [8]. This was found to stimulate the desired behaviour among peers playing by the rules, but encountered problems with whitewashing; behaviour where a user can clear a negative reputation by creating a new identity. More problematically, in this and many other systems, the reputation is self-reported, meaning peers communicate about their own reputation towards other peers. This makes it fairly trivial for peers to lie about their reputation, presenting an unfairly aggrandised reputation to others. As a result, a secure decentralised incentive system that incorporates history is still an open problem. This thesis investigates this problem using the Tribler platform.

The Tribler project is a research project a Delft University of Technology which aims to create new methods and algorithms for distributed networks. To investigate the properties of such networks in the real world, a BitTorrent client with advanced features is available to the general public. The project is developed in Python, and is open source, meaning all code is publicly available on GitHub [1]. Additionally packaged versions are available to anyone interested in using the software [2]. A recently added feature of the Tribler software is to provide anonymous routing, in order to enable privacy and prevent censorship. This feature is currently being utilised by Tribler users, and is functional, but the performance is often lacking in comparison to open traffic. One of the root causes of the performance degradation is the increased bandwidth requirements. Since the traffic is routed through multiple hops, the total bandwidth requirements are proportional to the number of hops. This means that a network that enables anonymous routing has even more need for a good incentive scheme.

Previous attempts in Tribler to create incentives for contributions culminated in the Bartercast protocol [9]. This is a fully distributed system that associates a reputation to each peer in the network based on a maxflow algorithm. While this protocol proved to be a resilient way to motivate most users to cooperate, the protocol was not tamper-proof, meaning code-savvy peers could cheat the protocol by providing false information to the system. While in most networks there is only

[1] `https://github.com/Tribler/`
[2] `https://tribler.org`

a small fraction of users that is willing to actively cheat, this minority can nevertheless undermine trust in the system and disrupt its functionality. This calls for a system that cannot be manipulated and is tamper-proof.

The solution pursued by the Tribler team is the Multichain; a distributed database based on blockchain technology. The blockchain has shown to be a great way of creating tamper-proof records in a distributed environment [10]. The concept also shows great promise in recording cooperative behavior. The idea of using this concept to stimulate cooperation in peer-to-peer networks was first conceived as an extension to the BarterCast protocol [3], but later matured into an self-contained system; the Multichain. In the Multichain, a record is created for each interaction between the participating nodes. Both parties then store their record, and can show them to other peers to prove their past actions. This allows peers in the network to treat people differently based on their historic behaviour. People who have contributed to the network can be rewarded by reciprocation, and freeriders can be punished by exclusion. These reactions to behaviour are not the goal of the system, but the prospects of them should motivate peers to change their behaviour. In game theory, this is known as *the shadow of the future*; the reckoning of future consequences of behaviour incentivises peers to behave differently. If all participants consistently react to historic behaviour in a suitable manner, rewarding cooperation and punishing defection, the shadow of the future leads to the emergence of cooperation by rational actors. Research shows that in repeated prisoner's dilemma games the shadow of the future does indeed drive people to cooperate [11]. The Multichain aims to use this principle to achieve cooperation in BitTorrent and other distributed networks.

A preliminary version of the Multichain has been implemented in code by S. D. Norberhuis [12]. Conceptual and practical issues still remain in this version of the code, and it has therefore not yet been integrated in a release of Tribler. This version is rather limited in scope, and does not implemented checks and measures to prevent cheating.

This thesis develops the Multichain further, outlining the road to the creation of an all-round blockchain based incentive system that can influence behaviour in actual systems. It also takes some steps forward on this road by implementing features and evaluates various forks on the path by evaluating different algorithms and strategies. In its fundamental form the Multichain system can be applied to a wide variety of scenario's where cooperation of individuals is desirable but does not come naturally. The theoretical properties of the system are mostly evaluated in such a generalised context, but to see the actual effects of different policies the system is applied to the file-sharing use case. Peer-to-peer file-sharing provides a useful test case for the Multichain because it is representative of other cases, has

[3] `https://github.com/Tribler/tribler/issues/67`

been the subject of scientific research in the past, and the Tribler platform provides a way to study the behaviour of actual users.

The desirable properties of the Multichain system and the challenges involved in realising these are explored in Chapter 2. The design of the historic record creation part of the system is described in Chapter 3, including several conceptual and implementation improvements upon the existing record creation protocol. This record creation part is then evaluated in Chapter 4, using a combination of specific tests, wider experiments and deployment to actual Tribler users. Chapter 5 explores another part of the Multichain system; that of record discovery, where created records are shared among the network. Various algorithms are evaluated in their effectiveness and their impact on resources using a trace-driven simulation. Based on this evaluation, a simple version of such a record sharing system is then implemented and tested using Tribler in Chapter 6. Finally, Chapter 7 summarises the results of the research in a conclusion, and surveys future steps for the Multichain system.

# Chapter 2

# Problem Description

Encouraging cooperation has been a problem throughout the ages. On the Internet, this mostly translates to attempts to create reputation systems, but this has proven to be a hard problem. An interesting test case for this research is that of BitTorrent networks, since there are plenty of opportunities for cooperation, results are measurable, and the stakes are not so high that accidents are unacceptable. This chapter explores some theoretical models for cooperation, and establishes the requirements for a system that can stimulate cooperation.

## 2.1 Cooperation in BitTorrent Networks

Theoretical models of cooperation describe an opportunity for cooperation as a situation where an individual can choose to perform some action which results in some benefit $b$ for another party, but also incurs a cost $c$ on the individual that could potentially cooperate [13]. If the cooperator wishes to avoid this cost, it can choose not to perform the action; this is known as defecting. For cooperation to be viable at all, the cost of the action must be smaller than the benefit. This ensures that the action is beneficial for the system as a whole. This condition is necessary but not sufficient for cooperation to occur, since the costs and benefits of cooperation apply to different people. The action in itself will, all other thing being equal, not benefit the cooperator. Cooperation becomes beneficial when parties choose to reciprocate; i.e. parties on the receiving end of the cooperative action will choose to also initiate the action themselves. If two parties choose to both apply the same action to each other, both will receive a net gain of $b - c$, which is positive under the assumption that $b > c$. However, in this scenario each individual party still has an incentive to defect, if this decision will not affect the decision of the other party.

This problem is poignantly shown in the prisoners dilemma, a well known example from the field of game theory. In this example, two arrested criminals A and B are separately offered the option to betray the other. If neither prisoner betrays the

other, both will be sentenced to a year in prison. If one of the prisoners betrays the other while the other does not, the betrayer will walk free, while the other will be sentenced to three years in prison. If both prisoners betray the other, both will serve 2 years in prison. The resulting payoff matrix is shown in Figure 2.1, with each tuple indicating the net gain of party A and B respectively.

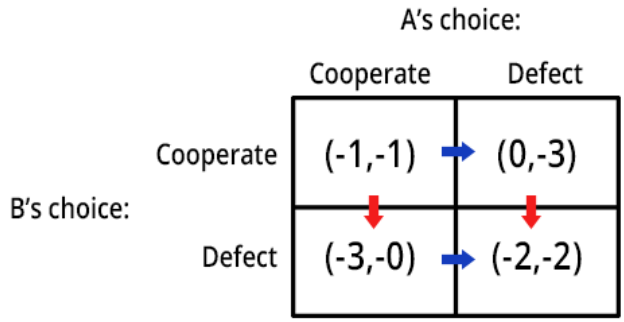



Figure 2.1: The payoff for different outcomes of the prisoners dilemma, denoted as (gain of A, gain of B).

We can see the prisoners dilemma as a cooperation problem. From this point of view the base scenario is the case where both prisoners defect and betray each other. Each prisoner has the opportunity to cooperate, inflicting upon themselves a cost of 1 year in prison, while rewarding the other with the benefit of a reduction of 2 years. We see that when both prisoners cooperate, they are collectively better off. However, when they cannot communicate, and thus not influence the decision of the other party, each party makes individual gains by defecting, reducing their cost without impacting their benefits. This is indicated by the arrows in the figure; blue arrows indicate prisoner A choosing to defect, red arrows indicate prisoner B doing the same. We can see that for all four arrows, the prisoner making the choice is better off, increasing his own gain. The combined effect of the blue and red arrow is however negative for both prisoners. This means that fully rational individuals will always defect, reaching a Nash equilibrium that is not globally optimal.

A similar situation occurs in BitTorrent networks without any incentive scheme. Uploading is a form of cooperation that yields benefits to the one being uploaded to, but also incurs some cost on the uploader using his bandwidth for this purpose. The cost of uploading is usually quite small because sufficient bandwidth is available anyway, but the benefits can be quite large, as downloading at a higher speed results in quicker availability of a desired file. The requirement $b > c$ thus holds for file-sharing in BitTorrent. However, without an incentive scheme there is no

direct reward for uploading, and the Nash Equilibrium is again one where everybody defects.

BitTorrent currently implements an incentive scheme to improve the situation. This scheme makes use of the fact that file-sharing does not consists of a single one-shot interaction, but a continuous series of repeated interactions. At any point during the interaction, nodes can decide to cooperate or defect. In this sense the process is more like the iterated prisoners dilemma: the hypothetical case where the prisoners get arrested on a regular basis, and thus have an opportunity to come up with a cooperative strategy over time. For the iterated prisoners dilemma multiple theoretical strategies are possible that result in cooperation. A famously successful strategy is called *tit for tat* [14]. This strategy attempts to cooperate, but answers defections with defections in the next iteration, retaliating for the last defection. This strategy works especially well if it is optimistic; occasionally retrying to establish cooperation. The tit for tat scheme is implemented in the BitTorrent protocol by peers preferring to upload to peers that also upload to them. This system is effective at stimulating cooperation among peers [6].

The incentive scheme however only works amongst peers that are in the process of downloading a torrent. In peer-to-peer networks however, *seeders* can also play an important role. Seeders cannot make use of the tit-for-tat scheme, since they are not downloading. They can thus not distinguish between cooperating and freeriding peers in the network, and will provide upload bandwidth to both equally. This reduces the incentive for peers to cooperate and enables freeriding. Furthermore, seeders themselves are not rewarded for their service to the network. This means there is no incentive to upload files, which could greatly improve the availability of rare files.

There is thus a need for a incentive system that takes historic behaviour into account, to stimulate cooperation where tit-for-tat cannot. The Multichain system is an attempt at such a system.

## 2.2 A tamper-proof accounting system

The goal of the Multichain system is to establish cooperation among peers in a decentralised manner. In order to achieve this, the peers should be incentivised in such a way that their goals are aligned with the goal of the network. Behaviour that is beneficial to the individual peer should also be beneficial for the network as a whole. Since the application domain is that of file-sharing, this means that making files available to the network by uploading them should be rewarded appropriately. To reward certain behaviour in the underlying network, the Multichain system is influenced by the events that occur in the underlying network. These events then result in some sort of judgement that distinguishes between desirable and undesir-

able behaviour, which peers then punish or reward in the original network. This creates a feedback loop, which, when designed properly, improves the behaviour in the underlying network. This concept is shown in Figure 2.2. In the figure, the red interactions represent the underlying network in which we want to change behaviour. In the Tribler use case, these are the interactions in the file-sharing network, but this could potentially consist of any type of peer-to-peer interactions. The blue Multichain comprises of the system we use to observe the network, and to adjust our own behaviour in the network.

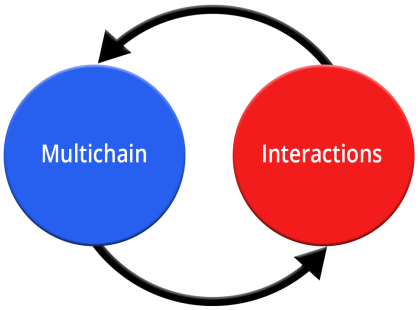



Figure 2.2: The feedback loop between the Multichain system and the underlying network.

### 2.2.1 The steps to influence behaviour

For certain behaviour to be rewarded in a peer-to-peer network, other peers must be made aware of this behaviour. In the network, behaviour consists of interactions between peers. We therefore need some sort of record-keeping system that keeps track of these interactions. This step is titled *record creation*. Peers must then discover the records of other peers, to be able to consider their behaviour; *record discovery*. When a sufficient amount of information is available, a peer can analyse the behaviour of other peers, and make some kind of judgement on it. A concise way of judging behaviour is assigning a numerical score to each known peer. The score is assigned based on an *Accounting Mechanism*. The final step then attaches consequences to the perceived behaviour by initiating (or avoiding) new interactions, and by allocating various amount of resources to it according to the perceived priority based on the reputation score. This is called the *Allocation Policy*. These steps are shown in Figure 2.3 for two peers. Both peers take the different steps involved in the system. The first steps require communication with other peers; the interactions are by definition between two peers. The record creation process also involves communication, since both peers must agree on the content of the record. This is also true for the record discovery process, since it consists of peers sharing records of themselves and third parties. The final steps,

consisting of the judging of behaviour and the attachment of consequences to this judgement are however autonomous; each peer makes its own decisions. This approach makes it harder to manipulate reputations and allows different peers to have different policies. This enables peers to use a policy that is appropriate to their situation; a peer that is strapped for resources may use a very conservative policy, only working with the very best peers in the network, while a peer that has plenty of resources available can use a more tolerant policy, serving any peer that behaves somewhat decent.

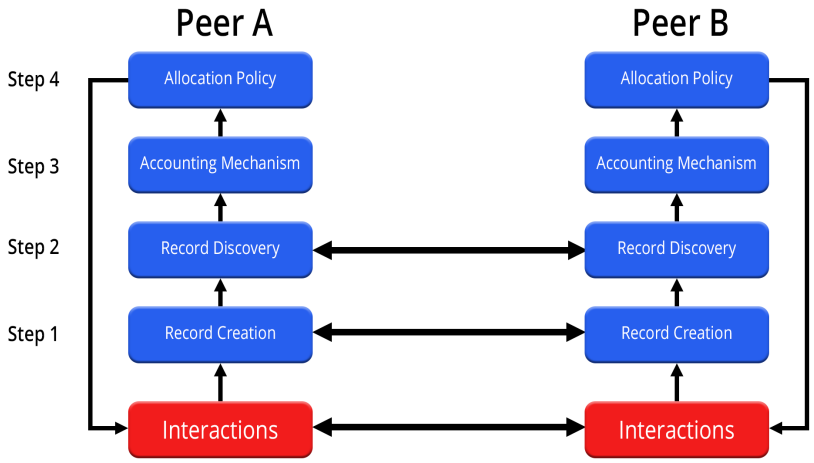



Figure 2.3: The different steps taken by the Multichain system to establish cooperation among peers.

Since the different steps have distinct functions, it is beneficial to implement them separately. This compartmentalisation makes it more manageable to implement the different aspects of the protocol in software and to test and evaluate parts of the system before the entire system is completed. This is in line with the principles of agile software development [15], allowing for improvement of the software and inclusion of new insights on the go. It also enables the evaluation of different combinations of accounting mechanisms and allocation policies to see which is most effective in realising beneficial behaviour in the network. Even in the long term implementations of different peers do not have to be identical; it is possible for different peers in the network to have different policies for judging and rewarding behaviour while interacting with each other. This means different peers can make different decisions based on the same Multichain. The steps are however not completely independent, as some policies in the higher steps might affect the feasibility of certain ways of cheating. Previous research on the third step [16] has mathem-

atically proven that some possible attacks on the record creation step, specifically Sybil attacks, can be mitigated by an accounting mechanism that is built to deal with them.

As mentioned in the previous chapter this thesis will focus on the first two steps. Previous work exists on the first step [12], but some issues remain with this work. On a fundamental level, the protocol as envisioned there is not sufficiently scalable, leading to problems for busy nodes. Additionally there exist several problems with the implementation. The second step has until now remained unexplored in the context of Multichain, and this thesis evaluates different protocols that could perform this function.

### 2.2.2 Requirements for the record creation step

We first consider the requirements for the record creation step. The first, most obvious requirement is that the record creation process is sufficiently accurate to characterise different types of behaviour. In order to reward certain behaviour in other peers, we must first have records faithfully describing their behaviour. A simple record-keeping system would consist of each peer maintaining a list of his own interactions. Other peers could then receive the list, and decide based on these interactions whether the peer has shown good behaviour in the past and should be rewarded or not. While this approach does in theory provides some incentives to be cooperative, it is obviously very tempting to cheat by providing a modified list, that shows fake good behaviour. The goal of the Multichain is to prevent this kind of cheating by making it infeasible, thus providing a tamper-proof record keeping system.

Networks generally become more useful when it includes more users. Meltcalfe's law even states that the utility of a network is proportional to the square of the number of users, and this has been empirically validated for some Internet use cases [17]. In a file sharing network, the availability of files is increased when more users are connected, greatly increasing the chance of a useful interaction being realised. To benefit from these effects we aim for a large network with a lot of users. This requires all components, including the Multichain, to be scalable. This means the performance of the software should not degrade when more users join the network. For the Multichain this means that each user cannot have complete information of all users in the network. However, the partial information must still be useful for judging the behaviour of other peers. In order to make meaningful judgements, each user will have to store significant amounts of data on a large number of peers, and hence the database used for this storage should be light-weight and efficient.

In summary, the record creation step should create across the network an accurate record of all interactions that have occurred. In order to fulfill this role, it should

be:

- *Accurate* in registering the interactions that occur.
- *Tamper-proof* in the face of malicious users.
- *Scalable* across millions of users

### 2.2.3 Requirements for the record discovery step

Record creation is not enough for rewarding good behaviour; this behaviour must first be known to other people. This means nodes must be able to discover records of other peers in the network. The discovery process must be informative; meaning each peer should have sufficient information regarding the behaviour of its relevant neighbours. Furthermore, the process must be efficient. Since a lot of interactions occur in the total network over time, discovering all records would be costly in terms of bandwidth and storage space. It is thus important to communicate as few records as possible, while still realising the required informativeness.

An emergent problem in many possible solutions for record discovery is that of load balancing. Nodes with a high amount of records or nodes that show good behaviour might be considered important nodes by certain algorithms. If these nodes are expected to have more informative records, these nodes may be polled for records very often. In some ways this is desirable behaviour, since having a lot of records shows a high capacity for interactions, which can be an indicator of a high capacity for handling requests for records. However, when specific nodes are polled too frequently, this leads to capacity problems, disturbing the functioning of the network.

In summary, the record discovery step should spread the records to nodes for which they are relevant. In order to fulfil this role, it should be:

- *Informative* by providing relevant information describing the behaviour of relevant peers.
- *Efficient* in its usage of bandwidth and storage space.
- *Load-balancing* with regards to differing nodes.

### 2.2.4 Constraints

For both steps there are some boundary conditions to the design space, designated by the research context, ethical and practical issues. An important design principle is the use of a distributed architecture. This means there can be no reliance on central servers, and all nodes should oFperate autonomously. This constraint provides a lot of challenges regarding the availability and trustworthiness of information.

Another premise is the principle of open enrolment; any Internet user should be able to join the network. Furthermore, the system should be churn resilient; as new peers come online and old ones go offline, the system should continue to function. Furthermore, it is not guaranteed that all peers in the network are directly connectable with all other peers. These constraints have to be taken into account when designing the Multichain system.

### 2.2.5 Research questions

The stated goals, requirements, and constraints can be translated into the research questions this thesis aims to answer. The overarching question is:

> *How can we use records of historic behaviour to encourage cooperation in distributed networks?*

This is a wide question that contains a lot of different aspects and is open to a multitude of approaches. To make the work manageable this thesis focuses on two sub-questions that can be approached directly:

1. *How can we create records of historic behaviour in distributed networks in a manner that is accurate, tamper-proof, and scalable?*

2. *How can we distribute records of historic behaviour in distributed networks in a manner that is informative, efficient and load-balancing?*

# Chapter 3

# Design of the Record Creation Functionality

The first step in the Multichain system consists of creating records of interactions between peers that have occurred in the network. In this chapter, we show how blockchains can be used to store these records in a reliable and tamper-proof manner. An effective protocol to create new records in these blockchains is layed out, and possible attacks on this process are discussed.

## 3.1 A tamper-proof data structure

The interactions between peers are stored in blocks. The blocks contain the transaction data that describes the behaviour of peers in the network. In our file sharing use case this is the amount of bytes that have been uploaded, and the amount of bytes that have been downloaded. This transaction data will be signed by both peers using public-key cryptography, and hence the public key of each peer is also included. When the transaction is signed by both peers neither peer can deny that the interaction has occurred while the block is available for checking. The signed block forms irrefutable proof of the interaction.

**Data hiding**
While individual blocks can be used to prove that certain interactions have occurred, this does not ensure an accurate representation of the behaviour of a peer is shown by a subset of the blocks. It might be tempting for peers to hide certain interactions that reflect poorly upon them, by not providing the blocks that correspond to these interactions. To prevent people from hiding some of their blocks, and thereby their historic behaviour, the blocks are chained together to form a blockchain. This means that for each peer there exists a unique, ordered sequence of blocks. This order is indicated by sequence numbers that are included in the block, one for each peer. The sequence of interaction blocks that form the total history of a peer is then a chain of blocks. Other peers can request a section of the chain, in-

dicated by sequence numbers. When a certain block is missing from the sequence, it is immediately obvious, meaning it is no longer possible to filter blocks to only show a positive subset.

**Data alteration**
A more advanced strategy to misrepresent behaviour is to replace certain unfavourable blocks in the blockchain with other more favourable blocks. To prevent peers from doing this, each block contains a hash that refers to the previous block. This hash is a value that describes the previous block in such a way that any modification of the block will also result in a change in the hash of the block. This means that when someone with malicious intentions modifies a block, the hash will change. However, since the next block will still contain the old hash, the modification is detectable. Hence, to modify a block while maintaining internal consistency of the chain, all blocks that come after the block in question must also be modified. This makes it much harder to make changes to previous blocks without anyone noticing.

**The blockchain**
An illustration of a blockchain with the features discussed above, i.e. signatures, sequence numbers, and previous hashes, is shown in Figure 3.1. Storing data in a blockchain makes it hard to tamper with the data and storing interactions using blockchains should thus form a complete overview of the behaviour of all nodes that can be used as the foundation for a reputation mechanism.

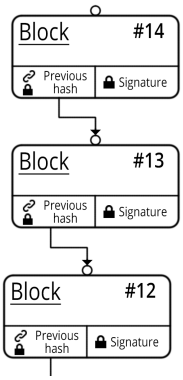



Figure 3.1: A simplified representation of three blocks in a blockchain.

## 3.2 Concurrency

Peers can have multiple simultaneous interactions with different counterparties. This causes problems when recording these interactions using the Multichain. As a result of the immutable nature of the blockchain, blocks cannot be altered once they have been inserted. Furthermore, due to the strict ordering, there is only one place where a block can be inserted into the chain. However, the signing of a block

by both parties may take some time, since it requires communication between both parties. During this time, the chain is effectively blocked, as the next block can not be appended to the chain. This is a result of the fact that a block with sequence number $n + 1$ cannot be created, until the block with sequence number $n$ is completely determined, since block $n + 1$ must contain the hash of block $n$.
This blocking of the chain leads to problems: if only one block can be signed at the same time the number of blocks that can be signed is severely limited. If the signing of a block takes $t$ minutes, the average amount of interactions that can be recorded on a blocking chain is $1/t$ blocks per minute. This limit can easily become problematic in many scenario's: in the Tribler use case file transfers over the Internet are recorded. A reasonable amount of time for a record to be signed, including sending a request to a peer, waiting for the peer to process it, and receive the reply, could be in the order of 2 seconds. This would limit the amount of interactions that can be recorded to 30 per minute, while there are nodes that have significantly more interactions than this limit. This means a blocking Multichain would not suffice for this application, and this is likely to be the case in many relevant scenarios.

There is another, more fundamental problem arising from blocking chains. It is possible that a series of requests form a blocking cycle. Imagine a node $A$ requesting a signature from node $B$. Node $B$ is however blocking this request because it has an open request towards node $C$ that must be resolved first. Imagine further that node $C$ has also blocked its chain, since it is requesting a block from node $A$. Of course, node $A$ is still blocked form accepting this request, since it has an open request towards node $B$. In such a scenario, none of the nodes can ever resolve their requests, and a form of permanent deadlock occurs. Due to the delays inherent in communication across the Internet, it is realistic that such a cycle would actually arise in practice. While it is possible to implement measures to recover from such a scenario, the Multichain would be polluted, the accuracy reduced, and time and resources would be wasted.

The above problems are resolved by having the Multichain feature concurrency; the ability to have multiple outstanding requests at the same time. The preexisting version of the Multichain does not have this feature, making it unsuitable for deployment on a large scale. The work in this thesis makes the Multichain non-blocking to solve these fundamental problems.

The hashes used to chain blocks together make it non-trivial to come to a non-blocking protocol. The solution is to base the chain on half blocks. These half blocks describe the interaction from the point of view of one of the interacting parties. This half block is then linked with the half block from the other party such that the interaction is again signed by both parties. Two half blocks together form the record of the interaction. The key difference is that the previous hash is based only on the information coming from the peer itself, not the counterparty. This al-

lows each peer to continue making blocks after appending a half block to its chain. This structure retains the feature of irrefutable proof by both parties, while simultaneously enabling concurrency across different peers in the network, allowing chains to grow without blocking waits.

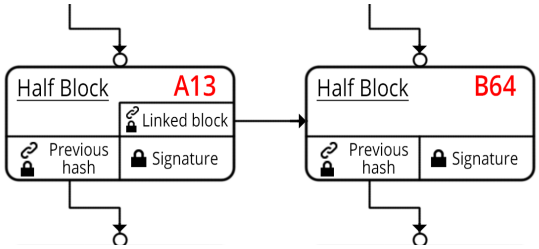



Figure 3.2: A simplified representation of half blocks in the Multichain.

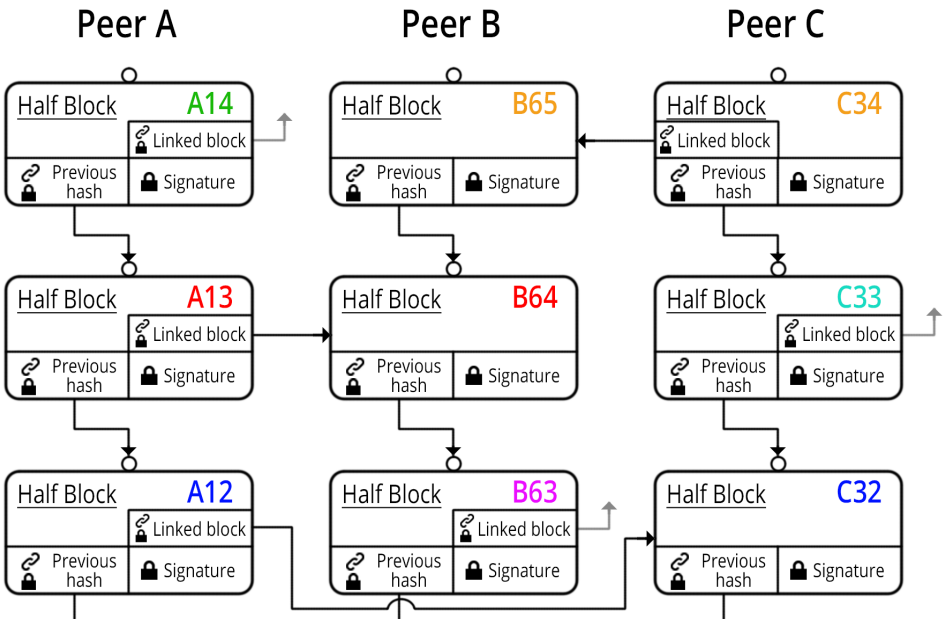



Figure 3.3: A simplified representation of some blocks in the Multichains of three peers, A, B, and C, and their interactions.

Figure 3.2 illustrates the concept of half blocks by showing the two halves that together record an interaction in a simplified format. Figure 3.3 shows these same blocks in a wider context, showing three half blocks per chain for three peers.

The figure shows an interaction between A and B, one between B and C, and one between A and C. Some other half blocks are also visible, corresponding to interactions with peers that are not shown. Half blocks can be identified by the combination of the identity of the peer that created the half block, and the sequence number of the half block in the chain of the creating peer. In these simplified figures half blocks are identified by the letter corresponding to the peer and a 2 digit sequence number, for example: A13. Looking at half block A13, we see that it contains a previous hash that refers to half block A12. This principle chains all of A's half blocks together. Since the hash describes all the contents of half block A12, half block A12 cannot be modified without making half block A13 invalid.
Also shown is the linked block field, which points to the half block B64. This means peer A and B have interacted, and this interaction is recorded by the combination of the blocks A13 and B64. We can also see that B64 has no linked block. This implies that peer B requested the interaction to be signed, and was the first to create his half block. It was not possible to point to the other half block at that time, since it did not yet exist. Peer B has then sent his half block to peer A, requesting it to create the corresponding half block. Due to the half block structure, peer B did not need to wait for a reply from peer A and could immediately continue and request the creation of a block with peer C, as it did in half block B65.

## 3.3 Block protocol

The creation of Multichain blocks is a process that takes place between two peers. This process takes place after an interaction (or a part of an interaction) has completed. The protocol to create the blocks is asymmetric, meaning both peers have distinct roles in the protocol. One of the peers initiates the process, and this peer then becomes the requester. The requester creates his half of the block containing information about the interaction, and the public keys of both parties. In the Tribler use case, the interaction data consists of the amount of bytes that were up- and downloaded. The requester half also contains the sequence number of the half block in the chain of the requester. Since the requester manages his own chain, this number is known by him at the time the half block is created. The combination of the public key and sequence number uniquely identifies the half block. The two halves of the block must be linked together to ensure both parties agree on the same interaction data. To link to another half block, we can use the combination of public key and sequence number as identification of a block. However, at the time the requester creates his half, it does not yet know what the sequence number of the block of the counterparty will be. Hence the requester only includes the public key of the counterparty, and leave the field for the counterparty sequence number blank. To create the blockchain structure, the requester then includes a SHA-256 hash of his previous block. The whole block is then signed using the keypair of which the public key has already been included. The requester then sends his signed half block to the counterparty, thereby requesting it to create and

send the other half block. The counterparty which receives the half block, then takes on the role of responder. The responder validates the incoming block based on its perception of the interaction, its own chain, and its available knowledge of the chain of the responder. After successful validation the responder creates a half block following the same procedure as the requester, with the notable difference that the linked sequence number actually contains the sequence number of the corresponding linked half block, namely the half block which the requester has just sent. The responder then sends this half block back to the requester ensuring both parties possess both halves. Together these blocks form irrefutable evidence that both parties have agreed on the transaction data, verifiable by any peer in the network that possess the blocks.

### 3.3.1 Signing initiation

Since the protocol has distinct roles for the requester and the responder, it is important to decide which role each party involved in the interaction will play. When two parties simultaneously decide to initiate, and both parties create a request block before they have received the other request block, two requester blocks describing the same interaction will exist. Due to the immutable nature of the blockchain, it is neither possible to link these blocks together, nor to remove either of these blocks from the Multichain. Hence, in this scenario we have created an intrinsic inaccuracy in the Multichain. The best possible resolution would now be to decide on one block to remain unsigned, and making the other block fully signed. However, deciding on which one is to be signed, and which one is not to be signed is again non-trivial, and one could easily end up in a scenario where both blocks are either signed or unsigned. It might be possible to resolve this ordeal by using a special block that indicates a perceived 'honest mistake' by the counterparty, nullifying a block. However, this solution comes with its own set of problems and thus simultaneous initiation is a scenario best avoided.
In order to avoid this, unambiguous rules must exist for deciding which party takes the role of initiating, and both parties participating in an interaction must adhere to the exact same rule set. This puts some constraints on the interoperability between different implementations. A natural way to determine the initiator is to let the party whose reputation will likely benefit most from the interaction initiate. Since this party has a bigger incentive to record the transaction, it has every reason to properly initiate the interaction. If we assume that both parties are honest, incentives are not crucial and any deterministic mechanism can be used to decide on roles. An example might be the alphabetic order of the names of both parties, assuming the name of the counterparty is know before signing.
In the Tribler use case, the primary determinant is the upload ratio; the party who has uploaded the most bytes will initiate the signing process. This party will likely have the most reputation to gain from the interaction. If both parties happen to have the exact same amount of bytes, the initiator will be decided based on the alphabetical ordering of the public keys.

## 3.4 Attacks and defences

A system that provides rewards to some of its participants is likely to be subjected to agents that attempt to abuse the system, trying to obtain the rewards without putting in the efforts desired by the system. For the Multichain, it is likely that some peers will attempt to receive better service from the network without contributing. If successful, this behaviour is seen as unfair by honest participants, and when it becomes prevalent it can reduce the faith of the users in the system and the effectiveness of the system. Therefore, the system should defend itself against different attacks and be tamper-proof to be functional. Attacks on the system can be divided in two types; first of all attacks can be devised against the protocol itself. These attacks consist of abusing the options that the system offers. A second type consists of attacks against the implementation of the protocol, and consists of doing things that should not be possible at all.
The concept to prevent abuse is to operate on two levels. First of all, each peer has some reputation based on its interaction history. Other peers can evaluate this reputation using the information in the Multichain and provide or refuse services based on this reputation. The second level consists of some peers that are lying about their history using the Multichain. When evidence of such lies is acquired, the act of lying can be proved irrefutable by conflicting messages signed by the same peer [1]. This proof can then be broadcasted along the peers in the network. Peers that are found lying should immediately be refused any service for which the Multichain is used, because once the peer is identified as a cheater, his reputation based of the Multichain is no longer reliable. Additionally, we aim to discourage lying in itself.

We identify several possible attacks on the Multichain, as well as defences against these attacks. They a presented here starting with attacks that are mostly fundamental to the concept, gradually moving towards attacks that are mostly specific to the implementation of the concept.

**False request** A trivial way to cheat the record creation step is to create false requests, i.e. requests to sign an interaction that has never taken place. If these interactions consist of contributions of the cheater, the signed records would falsely improve his reputation. In the previous version of the Multichain system, any request was signed, making this a vulnerability. To prevent this attack, nodes must check incoming requests for validity. This implies each node must keep track of outstanding transferred amounts, and compare incoming requests with these numbers. A module to do exactly this is included in the Multichain system.

[1] The evidence is irrefutable assuming that the cryptography used to sign messages can not be broken, and the private key of the peers remains private.

**Deny requests** Reciprocally to the previous attack, a peer can also refuse to sign records of interactions that have actually taken place. If these interactions consist of the cheater receiving contributions, the absence of the signed records would falsely improve his reputation. While this sounds very similar to the previous attack, mitigating this attack is much harder since the honest peer cannot force the other party to sign. An obvious recourse is to not interact with this peer again. This alone however, will not solve the problem, since a peer with a good reputation can receive contributions form many different peers. By denying requests from different peers in a hit-and-run fashion, a cheater can gain a large amount of contributions from a large number of peers, without this being properly reflected in a lowered reputation. The Multichain system does not provide an effective way for honest users to communicate this form of cheating to other peers, since a simple message claiming some peer cheated in this manner could also easily be faked. To ascertain the truth about such attacks on the reputation system, an entirely separate meta-reputation system would have to be created, but this seems to be a counterproductive strategy.

One way that is productive in dealing with this attack is to use a policy of gradually developing trust between peers. Using such a policy, peer who have no prior direct interaction history first establish trust by signing smaller blocks, covering only parts of an interaction. When one of the smaller blocks is not signed, the interaction is aborted. A suitable scheme might start with small blocks (for example 1 MB), and use a exponential growth. Such a scheme incurs minimal initial risks, while allowing for rapid growth of the block size, keeping overhead costs of creating extra blocks limited.

**Hiding blocks** Once blocks are created, a way for attackers to improve their reputation is to hide blocks that portray them negatively. When someone requests blocks from them to gauge their reputation, they might send only their best blocks. Fortunately, the blockchain structure defends against such an attack; since every block has a sequence number, anyone can request specific blocks, which the attacker cannot simply refuse to provide without being detected as a fraud. Furthermore, since every block is available to both parties involved, an attacker can never prevent distribution of the block by the counterparty.

**Branching** To hide negative records from others whilst still using the blockchain structure, attackers might branch their Multichain. In cryptocurrencies such an attack might result in *double spending*. In a branching attack, the attacker's blockchain is at some point split into two sequences. By signing the use of contributions on one of the branches, and showing the other branch when a reputation is requested, the attacker can still attempt to hide some blocks. This however always creates evidence; the two branches must contain blocks with the same sequence number. When these blocks are distributed in the network, it is likely that some peer will in

time come into possession of both of the two conflicting blocks. Since the blocks are clearly in violation of the protocol, and are both signed by the attacker, these form irrefutable proof of lying. Such a proof consists of only the two blocks with identical sequence numbers, and can thus easily be distributed as evidence of the attacker being malicious, resulting in a ban from the network.

**White-washing** The principle of open enrolment makes it easy to create a new identity on the network. This means that anyone with a negative reputation can opt to leave their identity behind and create a new one, white-washing their reputation. This means that new identities must always be treated with some distrust; they have no record showing their reliability. However, it not possible to completely deny service to new identities; this would prevent legitimate new users from ever joining the network. This means there is always the inherent possibility of some leakage of contributions to fresh identities who will never reciprocate. This problem can be managed by giving fresh identities low priorities when allocating resources, and this is something the allocation policy step should implement.

**Sybil attack** Open enrolment also enables another type of attack involving multiple identities that correspond to the same agent: the Sybil attack. In such an attack, a number of fake identities create fake interactions between them. Since both parties involved in the record are complicit in the attack, they will both sign the fake interaction. The system of fake interactions can be used to unjustly boost the reputation of some of the identities of the attacker. Since there is no known feasible method of preventing such an attack in the Record Creation step, this attack must be dealt with in the accounting mechanism step. Fortunately, P. Otte has proven mathematically that, under certain conditions, the Sybil attack can be curbed [16].

**Denial of service** Every service on the Internet is to some extent susceptible to denial of service attacks, where attackers make such a large amount of requests to the service that the service is unable to process them. As a result, legitimate requests can also not be handled. To defend against such attacks, bogus requests should be detected quickly and handled without using to much resources. This should be kept in consideration when implementing the protocol.

**Hash Collision** One of the aspects that underpin the security of the Multichain is the assumption that it is not possible to create a block with a specific hash. If it would be possible to do so, an attacker could replace a block in his chain without invalidating the previous hash pointer. Although it would still be possible to detect such attacks by finding duplicate public key and sequence number pairs, replacement blocks with the same hash would violate the idea of immutability and would compromise the security of the Multichain. Because the nature of a hash is to reduce a large string of bits to a smaller string, there will always be multiple bit

strings that produce the same hash. A bit string, in this case a Multichain block, that produces the same hash as another is called a Hash Collision. While the theory guarantees that such hash collisions exist [2], hashing algorithms used for security purposes, known as cryptographic hashes, make it deliberately hard to find such a collision. With such hashes a brute force attack, consisting of guessing random bit strings until a collision is found, are usually computationally infeasible. However, sometimes weaknesses are found in hashing algorithms that allow for the efficient generation of collisions, breaking the security aspects of the hash. A well-known example is the MD5 hash, which was designed as a cryptographic hash in 1991 [18], shown to be vulnerable in 1996 [19], broken in 2004 [20], and has since been exploited in real-world attacks. The pre-existing version of the Multichain uses the SHA-1 hash to refer to previous blocks. However, the SHA-1 hash had already been subject to better-than-brute-force attacks and was theorised to be vulnerable to even more efficient attacks [21]. Since SHA-1 might be vulnerable to attacks in the future, the protocol was upgraded to use the safer SHA-256 hash. This decision was later supported by the release of a practical attack on SHA-1 [22].

## 3.5 Conclusion

The Multichain utilises the blockchain structure create a tamper resistant record creation system. The protocol to create blocks is specified such that it is capable of the accurate recording of interactions, while simultaneously being scalable due to the use of concurrency. Due to the requirement of open enrolment, some attacks cannot be prevented during the record creation process. These attacks must be dealt with in the other steps in the process.

[2] Given that the bit string has more entropy than the hash.

# Chapter 4

# Validation of the Record Creation Functionality

The aim of this thesis is not only to specify a protocol that works in theory, but to actually implement the protocol in software, in order to verify the functionality and demonstrate its usefulness in the real world. To achieve this goal, the code is continuously tested for correctness during the software development process. When features and aspects of the software are somewhat complete, they are evaluated using experiments where the software runs in conjunction with its file-sharing use case. Finally, the software is deployed and released to users to see its real-world performance. This chapter describes these validation steps and their results.

## 4.1 Validation framework

The protocol is tested in the context of the Tribler project. This is a project that studies advanced features peer-to-peer file sharing networks. Tribler incorporates *communities*, which are used to share data among a subset of peers. Some communities share content based on communal interest, but the system of communities is also used to implement some of the advanced features of Tribler. The Multichain system is implemented in Tribler as a community, where the participating peers share data about each others blockchains. The community is used as a wrapper in which the peers communicate and send requests for the signing of new interactions or the sharing of historical records.
Like all Tribler communities, the Multichain community uses the Dispersy framework to exchange messages among peers [23]. Dispersy enables the Multichain to send messages to other peers reliably without worrying about the protocols on the lower levels. However, the Multichain does not use the more advanced methods of message distribution available in Dispersy, instead relying on relatively simple point-to-point messages. This means that it would not be too problematic to remove the dependency on Dispersy, might it ever form a liability.
This is also one of the reasons why the blocks are stored in a database separate

from the Dispersy framework. In addition, such a separate database allows for more control over and optimisation of database operations that are specific to the Multichain. The database is implemented using the SQLite database engine [1].

## 4.2 Experiments

In order to verify the workings of the protocol, several experiments are run in a controlled environment where real-world situations are simulated. These experiments help to detect certain bugs and identify improvements to the protocol, which have since been implemented in the code.

### 4.2.1 Signing policy

An important implementation detail of the signing protocol is the timing; when to initiate the creation of a block. Two different policies are considered in this thesis. One policy is to sign a block at regular intervals in the amount of data transferred; when a certain threshold of outstanding data is reached, a block would be signed. This is called an MB limited policy. To evaluate the effects of different signing policies, experiments are run in conjunction with the Tribler file-sharing functionality. The experiments consist of several instances of Tribler with the Multichain simultaneously. During the experiments the instances communicate with each other and initiate file-transfers. These file-transfers are then recorded using the Multichain system. A graph of a Multichain produced by such an experiment is shown in Figure 4.1. This experiment uses a MB limited policy with a 5 MB threshold.

In the graph, each vertex represents a Multichain block. Note that at the time the experiment was run, concurrency was not yet implemented, so these are full blocks, not half blocks. Each block thus represents a part of an interaction between two peers, each part consisting of 5 MB of data. Each block has two arrows pointing to an earlier block. This is for each peer the last block in the chain before this block. Since there are two peers involved in an interaction, there are two arrows per block. Vertices in green are genesis blocks; these are the first blocks in a chain for a certain peer. Because it is the first, it has no reference to a previous block for this peer. A light green block is a genesis block for both peers, and thus has no reference to any previous block. A dark green block is an genesis block for one of the peers involved, and thus there is one reference to a previous block, from the peer for whom the block is not a genesis block. Red vertices represent half-signed blocks; these blocks were not countersigned by the other party and thus only include the request part. These blocks are not usable as a reliable record of interactions in the network.

[1] `https://www.sqlite.org`

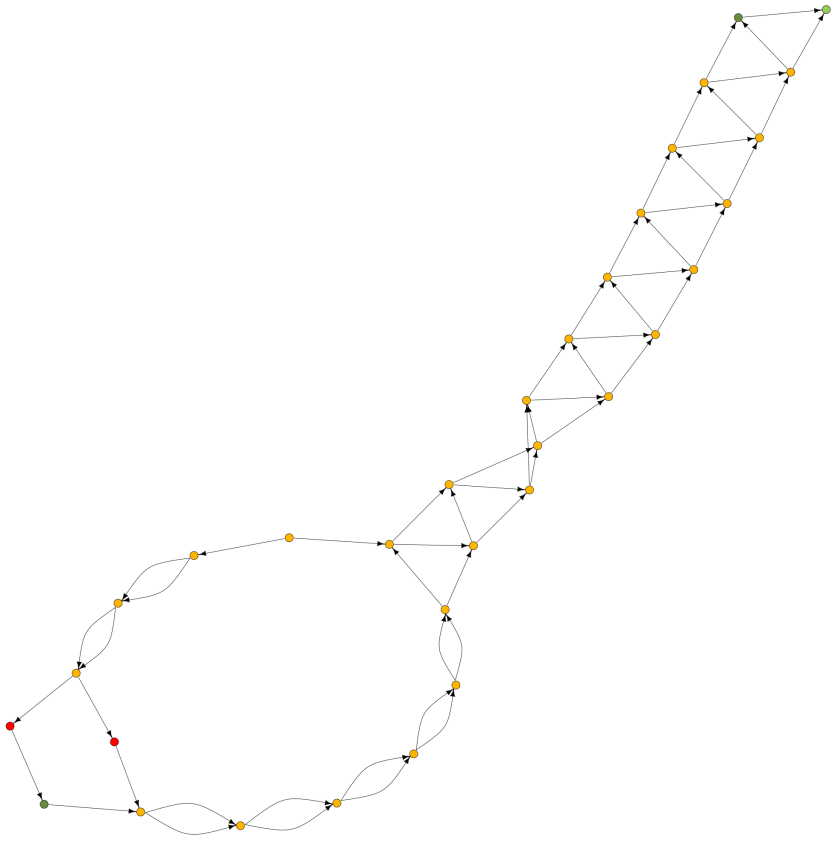


Figure 4.1: A block graph showing the blocks created during a few file transfers using the 5 MB block limit.

On the very bottom of the image, a structure is seen where both arrows from a block repeatedly point to the same block. This means the last block is the same for both peers; i.e. the previous interaction was between the same peers. The chain thus represents an interaction between two peers that was cut up into multiple 5 MB blocks. The structure seen on the top right part of the graph shows a similar occurrence but here three peers are involved; one of them interacts with two others simultaneously. This peer alternately creates blocks with both peers.

The second policy that is considered aims for a structure where each block corresponds to a single interaction. Since downloads in Tribler occur in encrypted tunnels that guarantee privacy, the end of an interaction correspond to the closure of such a tunnel. The second policy thus uses the tunnel close event as natural point to trigger the signing of a block.

To compare the two policies the amount of nodes that run in parallel during experimentation process is increased. A large scale experiment was run with 100 nodes interacting simultaneously. A Multichain resulting from an experiment using a MB limited policy is shown in Figure 4.2. The positions of the blocks are determined here using the Kamada-Kawai algorithm [24], as this seems to produce the best results among general purpose graph drawing algorithms. It is however still very hard to see the structure of the Multichain clearly with this amount of blocks. Creating a better layout is non-trivial, and it is likely impossible to prevent crossing edges. An important take-away from the graph is that there are again chains of repeated interactions, as a result of single interactions being split into multiple blocks, like in Figure 4.1. One such repetition chain extrudes from the graph on the left side of the image. The scaled-up experiment shows the cluttering resulting from small blocks, and emphasised the need for fewer blocks.

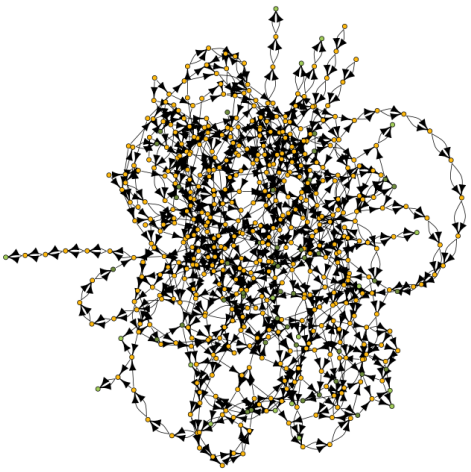


Figure 4.2: A block graph showing the blocks created during an experiment with 100 nodes using a 1 MB limit policy.

The same experiment was also run using the tunnel close policy. A Multichain resulting from that experiment is shown in Figure 4.3.
The Multichain resulting from the tunnel close policy has fewer blocks, and there are no longer any long chains of repeated interactions.

To compare the two policies, we gauge the number of blocks needed to record the file-transfers. Preferably, this number is as small as possible. We also consider the amount of repeat blocks; blocks for which the previous block of the requester is the

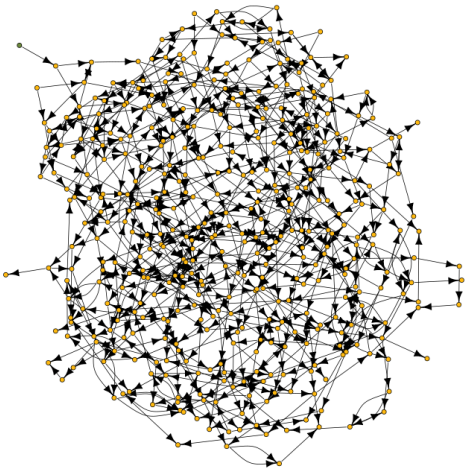


Figure 4.3: A block graph showing the blocks created during an experiment with 100 nodes using the tunnel close policy.

same as the previous block of the responder. Such blocks split a single interaction into multiple blocks, unnecessarily increasing the amount of blocks. It is therefore especially desirable to reduce the number of repeat blocks. The amounts of blocks in the experiments for both policies are shown in Figure 4.4.

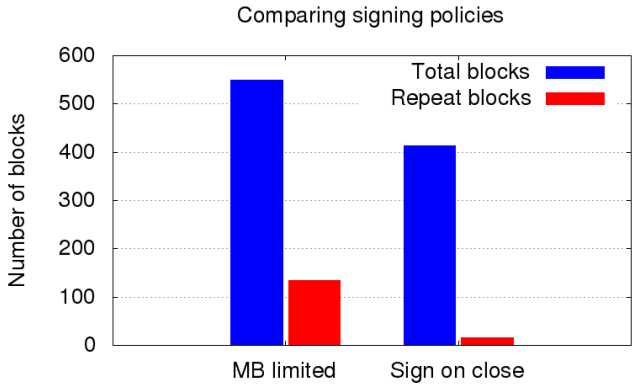



Figure 4.4: The amount of blocks needed to record interactions using different policies.

It is clear that with these settings, a MB limited policy creates more blocks in general, and in particular creates more repeated blocks. This is costly in terms of processing time, bandwidth and data storage. With a MB limited policy, these

cost would make it problematic to operate the Multichain network on more active nodes. While one solution would be to use a larger limit, a more elegant solution is to strive for a single block per interaction using the tunnel close policy. This measure greatly reduces the number of blocks, and thus the cost of operating the Multichain system. The larger block size also has an inherent problem; the average amount of pending bytes, i.e. the amount of transferred bytes that have not yet been included in a signed block, is proportional to the size of blocks. This means that when after an interaction a block is not signed by both parties, a larger amount of data is unaccounted for in the Multichain record. The failure to sign could be a result of a disrupted connection between peers – always a possibility due to the unreliable nature of Internet connections. In this case, there is a trade-off between the expected accuracy of the Multichain records, and the cost of running the protocol; a smaller block size, leads to higher accuracy, but a higher running cost in terms of bandwidth, processing time and storage space. Another reason why a block could fail to be signed is by a conscious refusal of a malicious peer to sign it when the block would lower its reputation. This would be an attack on the system, as discussed in Section 3.4.

In summary, the tunnel close policy increases the risks associated with failures and attacks somewhat due to a higher average amount of pending bytes. However, the policy does lower the amount of blocks required, and thus reduces the cost of running the Multichain. We therefore choose to use the tunnel close policy to initiate the signing process.

### 4.2.2 Performance

To use the Multichain system across a peer-to-peer network, the computational requirements must be kept in check, so that regular users can run the software on their computers. To check the performance of the software, an experiment is run using Tribler, creating a number of half blocks while measuring the elapsed time. The results of 10 runs of this process are shown is Figure 4.5. While the figure corresponds to just a single machine, it gives an impression of the order of magnitude of the amount of records a computer can sign per unit of time. This amount is rather large, with several thousand half blocks per minute being created. There is some variation between runs, but all runs show an order of magnitude that is easily sufficient for the Tribler use case, and likely for many other potential uses for the Multichain.
The graph shows a slight slowdown of the rate at which blocks are created. This is a result of the database filling up, making the insertion of new records and the retrieval of others slower. This slowdown is however so small that it should not be problematic, as long as the records that are gathered by the Record Discovery step are selected carefully, such that the rate is not too high. This problem is considered in Section 5.1.3.

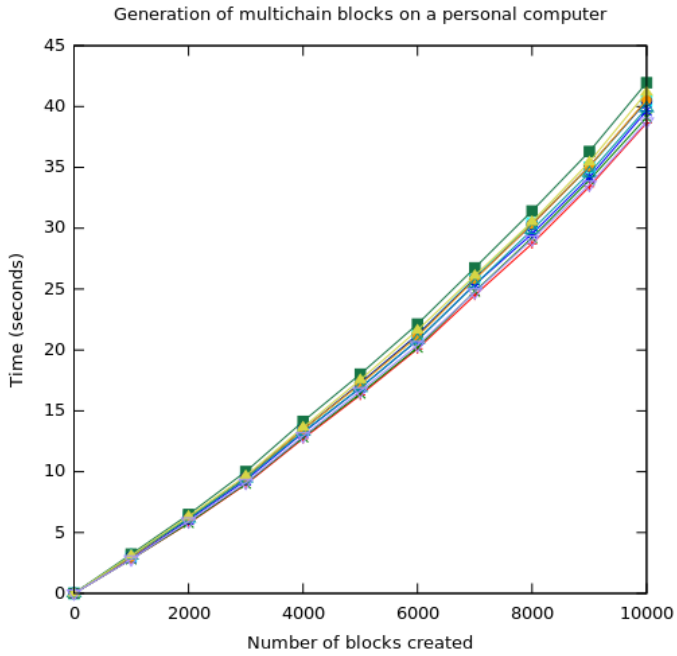



Figure 4.5: The amount of half blocks created over time during an experiment on a personal computer. It is clear that recording several interactions per second is easily within the capabilities of the process.

## 4.3 Deployment

While the experiments in simulated experiments provide some validation of the protocol, the real test for the protocol consists of its deployment in the real world across Tribler users all over the world. In this setting, all kinds of limitations and noise factors are present, and this provides a great test to the resilience and robustness of the software. Since the Tribler software is actively being used by real people, it is essential not to impede with the working of the software. This means that for the very first phase of the deployment of new features, a lack of correctness is an acceptable outcome, but under no circumstances may the working of the rest of the software be affected, for example by crashes. This idea is summarised as the 'do no harm policy'.

### 4.3.1 Closed group testing

The first integrated test was conducted with an early version of the software among members of the Tribler team. Although the number of peers and the geographical spread was very limited, this form of testing still brought many problems to light that were not seen when testing in a virtual environment. Some statistics are shown in Table 4.1.

| | |
|---|---|
| Number of blocks | 4823 |
| Number of distinct identities | 111 |
| Total MB uploaded | 441370 |
| Total MB downloaded | 13169451 |
| Average MB uploaded in a block | 91.5 |
| Average MB downloaded in a block | 2730.6 |

Table 4.1: A number of metrics resulting from the internal testing

The table shows a big difference between the amounts uploaded and downloaded. This is not impossible in the version tested here, since the blocks here contain the amounts up- and downloaded from the perspective of the requester of the block. The ratio seen here is however so extreme, that it is likely a sign of a bug. This turned out to be indeed the case. This was a bug not in the Multichain itself, but in the code that accounts for the data amounts during the interactions.

An excerpt from the block graph is shown in Figure 4.6. Another bug is clearly visible here; each block should have at most two other blocks referring to it, since each peer involved can have only one next block pointing towards it. However, the graph clearly shows several nodes that have much more blocks referring back to them. Since there were no attackers in this scenario it must be the result of a bug. As a result of the experiment, the bug was detected to be the result of unforeseen behaviour of one of the SQL-queries used in certain scenario's.

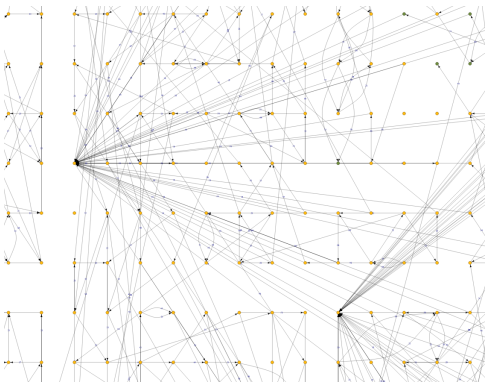


Figure 4.6: An excerpt from an early Multichain block graph, highlighting inherent inconsistencies.

The bugs that were detected were fixed, and the testing suite used to evaluated the software was updated to include tests that would detect bugs of these types. The chain itself was discarded after the bugs were fixed. Discarding the chain was not a problem since this was only an internal release.

### 4.3.2 Alpha release

After updating the software to deal with the bugs that were found during the internal release, the first version of the Multichain system was released to the public. This version only covered the first step of the process, Record Creation, and included a limited amount of features. This version did not yet include the half-block structure, and did not include validation of requests or blocks. This lack of features meant that updates would be required, and that these features might break compatibility with older database versions. The decision was made not to aim for migration of the database, but to simply discard the data when a new format would become available. This means that any records of contributions created in this version of the Multichain are lost in the long run. This choice was communicated to the users, and the update was optional, making it primarily for users interested in the technology.

As the most important goal of the release is to evaluate the functioning of the software in the real world, it is important to consider how we enable useful measurements. The fact that the system is distributed makes it hard to monitor the behaviour of the different nodes. Since the record discovery step had not yet been implemented at the time of the release, records from other peers were not collected. To check the operation of the Multichain, E.M. Bongers, another member of

the Tribler team, implemented crawler functionality which enables dedicated machines to gather blocks from the network. This results in a data set that can be used to analyse the Multichain that is created by the network using the Alpha release.

A snapshot was taken on the 10th of August 2016, containing all blocks created before that date. Some statistics resulting from the data set are shown in Table 4.2. The first 256 blocks that were gathered by the crawler are displayed in Figure 4.7. The nodes in this graph are positioned using a space-filling curve, an innovation from the same E.M. Bongers. As expected, the up- and downloaded amounts here are much more balanced, and there are no incidents where hashes or public key and sequence number pairs are reused. As such, most blocks will have exactly 2 incoming edges, and two outgoing edges, referring to the next and previous blocks of both participants. Edges that are missing can be explained by the fact that the relevant blocks have not been gathered as one of these 256 blocks.

| | |
|---|---|
| Number of blocks | 43908 |
| Number of distinct identities | 708 |
| Total MB uploaded | 379633 |
| Total MB downloaded | 526839 |
| Average MB uploaded in a block | 8.6 |
| Average MB downloaded in a block | 12.0 |

Table 4.2: A number of metrics gathered by crawling the nodes in the Alpha release

## 4.4 Conclusion

The functioning of the record creation protocol is validated on multiple levels. Using both specific tests, experiments and a release of the software integrated in Tribler the protocol is shown to work as designed, standing up to all practical challenges in the Internet environment. This shows that the record creation step is not just conceptually valid, but actually functions in an Internet use case.

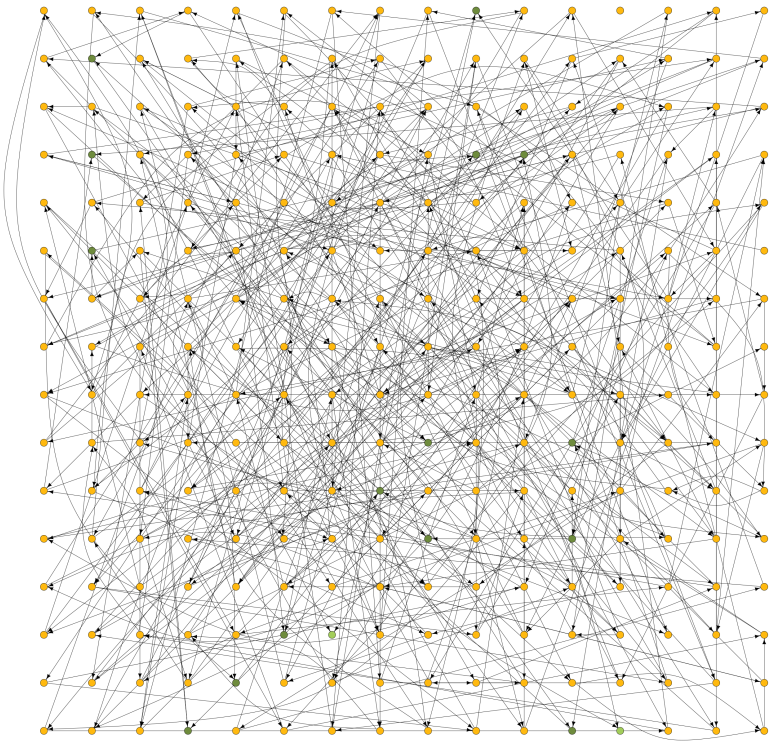


Figure 4.7: A Multichain block graph showing the 256 first crawled blocks from the Alpha release.

# Chapter 5

# Record Discovery Algorithms

Once the protocol to create blocks is used by the network, interactions by all peers are securely recorded on the Multichain. However, in order to accurately evaluate behaviour of other peers using the *Accounting Mechanism*, individuals must first find out about the interactions they are not directly involved in. In order to do this, peers must share blocks with others in the network. This is done in the *Record Discovery* step, which can be approached in several different ways. This chapter considers some algorithms for this purpose and evaluates them using a simulation, considering performance and load balancing.

## 5.1 Design

To discover records from other peers, nodes find out which other nodes are currently online, and request from them a subset of their blocks. The other node will then send the indicated blocks to the origin of the request, allowing it to acquire information about interactions it was not itself involved in. Since each node is responsible for maintaining its own chain, it should always be able to handle requests for its own blocks. For example, if node A wants to find out about node B, it could ask node B for its last 100 blocks. Node B should always be able to provide these blocks. It is also imaginable one would want to acquire information indirectly, for example node A could request from node C all available blocks regarding node B. However, node C might not always be able to provide such blocks.

### 5.1.1 Exploration strategies

Two general strategies might be envisioned around acquiring blocks. The first would consist of requesting the information when needed. This would mean that when an interaction with a certain peer is being considered, the system would then try to verify the reputation of the peer in question by requesting blocks regarding

the peer before starting the interaction. While this is efficient in the sense that only relevant information is collected, such a manner of operation might introduce delays in the interactions. Furthermore, this might make it easier to craft attacks specific to the request, since the time between the gathering of information and the decision whether or not to interact is relatively short, reducing the time span in which contradictory information might arise.
Another strategy is to explore the Multichain ahead of time, gathering information continuously. In this way when an interaction with some peer is considered, there is often already an idea of the reputation of that peer allowing a decision to be made much faster. Furthermore, this exploration enables the system to detect conflicting blocks that are around in different parts of the network, increasing the capability of the network to prevent fraudulent behaviour. This increases the attack resilience of the system. Additionally, a continuously available overview of the reputation of peers in the network could potentially be used for many additional purposes, such as searching for information, or preventing spam.

### 5.1.2 Random walks with Dispersy

A commonly employed strategy to explore a graph is that of a random walk [25], [26], [27]. This strategy gathers information by querying nodes, then traversing a randomly chosen edge, and proceeding to query that node. This process is then repeated for a number of times, until a new random walk is started. In this way, information is gathered about the nodes in the vicinity of the node that is walking. The number of steps taken from the starting point can be a fixed value, but can also be subject to some variation by using a teleport probability $p$. When using this variation of the algorithm, after each step there is a probability $p$ of teleporting home, restarting the random walk. The resulting length of the path is geometrically distributed with expected value $1/p$.
Walking in a peer-to-peer network is hindered by the fact that not all participants are online at all times. Peers which are offline cannot be contacted and cannot provide information. To walk the network where some fraction of the peers is online, the algorithm maintains a list of nodes of which it is aware that they are currently available. The association with the peer being online then forms an edge, over which the algorithm can walk. Note that by this definition an edge between two nodes does not necessarily mean that there exists an Multichain record between them.

When querying a node in a random walk, various types of information can be transferred. To be able to sustain the walk, at least one edge must be given. In the random walk used by Dispersy and in the experiment, only one edge, corresponding to an online peer, is given. This peer will then be added to the list of online peers by the originator of the walk. Additionally, in the experiment the Multichain blocks of the peer being queried are shared. The algorithm for random walking as described is shown in pseudo-code of Algorithm 1.

**Algorithm 1** Random walking *with* teleport probability in Dispersy

```
for each step:
    if uniform_random_variable() < teleport_probability:
        # teleport back home and random walk from there
        next_node = pick_random_from(connected_neighbours)
        walk_towards(next_node)
    else:
        # continue the walk
        next_node = connected_neighbours.last_added()
        walk_towards(next_node)

def walk_towards(node):
    new_peer = node.request_neighbour()
    connected_neighbours.add(new_peer)
    new_peer.request_blocks()
```

The fact that a discovered peer is added to the list of connected peers gives this form of random walking remarkable characteristics. It means that even when using walks with a fixed length, all nodes can eventually be explored. This is a result of the fact that a newly discovered peer is added to the local list of connected peers, and by this the distance is reduced from two hops to one hop. In effect, the graph is modified during the walking process. This non-standard method of walking which is used in Dispersy allows for a significant simplification in the walking algorithm. We can use a walk length of one, or equivalently a teleport probability of 1. This removes the need for a conditional branch. The resulting pseudo-code in Algorithm 2 shows that the resulting code is significantly simpler. We hypothesise that, due to the effects of adding peers to list of connected nodes, the simpler algorithm is in practice equivalent to the canonical version with a teleport probability.

**Algorithm 2** Random walking *without* teleport probability in Dispersy

```
for each step:
        next_node = pick_random_from(connected_neighbours)
        walk_towards(next_node)

def walk_towards(node):
    new_peer = node.request_neighbour()
    connected_neighbours.add(new_peer)
    new_peer.request_blocks()
```

### 5.1.3 Focused walking

Since the storage and processing of large amounts of blocks is costly, the goal of the record discovery process is not merely to gather blocks as fast as possible, but to gather the most relevant blocks. Blocks are relevant if they have an impact on the other steps in the process, namely the accounting mechanism and the allocation policy. Blocks that impact these steps change the behaviour of the network and can actually spur cooperation. To create a useful reputation method for the Multichain we can incorporate the concept of transitive trust. Transitive trust means that peers with a high reputation can transfer some of their reputation onto others that they find trustworthy themselves. If a reputation algorithm is based on transitive trust, the neighbours of a node with a high ranking have a higher expected ranking than a random node in the network. A way to potentially obtain more relevant information while walking is thus to not pick a next node to explore at random, but to prefer nodes with a higher reputation (from the node's own perspective). This enables a secondary feedback loop, as shown in Figure 5.1, and could allow the process to gather information more effectively. Such a feedback loop could also have downsides: firstly, if the focus is too strict, the process could only explore a local part of the graph, never noticing peers that are also worthy of interactions. Another problem could occur in the load balancing; since the algorithm prefers nodes with a high reputation, such nodes might receive much more requests for blocks and could become overloaded. On the other hand, by gathering blocks primarily from nodes with a high reputation the network can also protect itself against spam attacks. Furthermore, it might also increase attack resilience, since it is harder for low reputation nodes to spread false information. To evaluate these effects of focused walking, the functionality is implemented and simulated.

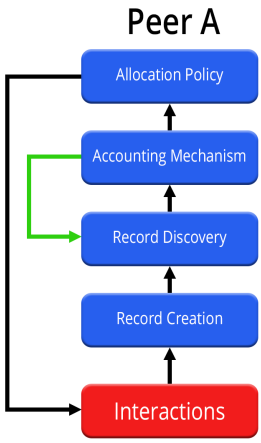



Figure 5.1: The secondary feedback loop that is introduced by focused walking, indicated in green.

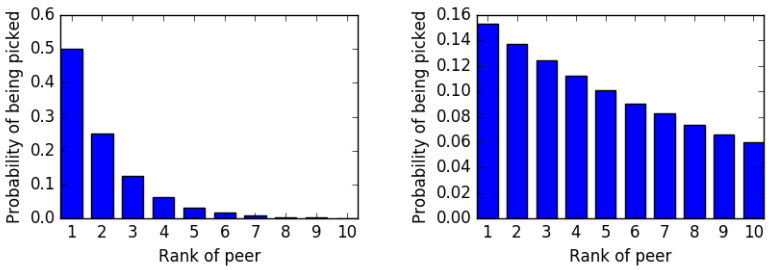
To focus on walking towards nodes with a higher reputation, a reputation score for each available live peer is first calculated, and these peers are then sorted in order of their score. We now have a preference for picking the peer with the highest reputation. However, if we would always pick the peer with the highest reputation, we might not explore the entire graph and we would cause load balancing issues for peers in the network that have a good reputation. We therefore only pick the highest ranked peer with a probability $\phi$, with $0 < \phi < 1$. If we do not pick the highest ranked peer, we are now most interested in the second highest ranked peer. Since we are facing the same dilemma here, we again not always pick this peer, but repeat the same procedure and pick it with the same probability $\phi$. This process is true for all peers in the order of their ranking, so we generalise the procedure for the entire list: The algorithm picks the peer with the highest score with a probability $\phi$. If it does not pick the peer with the highest score, it moves on to the next peer, and picks it with a probability $\phi$. This process is repeated until a peer is picked. If the end of the list is reached, it loops back to the beginning. This algorithm is described in psuedo-code as Algorithm 3.

**Algorithm 3** Selecting a neighbour in a focused manner

```
index = 0

# Select an edge from the ranked connected neighbours:
while uniform_random_variable() > phi:
    index = (index + 1) % len(ranked_live_edges)

next_node = ranked_connected_neighbours[index]
walk_towards(next_node)

def walk_towards(node):
    new_peer = node.request_neighbour()
    connected_neighbours.add(new_peer)
    new_peer.request_blocks()
```

The peer that is picked will be the target of the next walk step. The extend to which the walking is focused on peers with a high reputation is determined by the parameter $\phi$. When the parameter is 1, all walks will be directed to the peer with the highest reputation, resulting in repeated requests to the peer with the same peer, until a peer with an even higher score is found. When $\phi$ approaches 0, there is no bias towards nodes with a higher score, and the walk becomes purely random. While focused walks have potential to increase the efficiency of the walk, a $\phi$ that is too high will have adverse effects. The walk will in many cases converge rapidly to a steady state, where only a small subset of nodes is explored. Furthermore, a high $\phi$ will result in a high fraction of requests being directed to nodes with a high reputation, creating load balancing issues. An important challenge in applying this algorithm is thus to find an appropriate value for $\phi$.

The algorithm results in a probability distribution, where every peer has a certain probability of being picked according to its rank. A few examples of such distributions are shown in Figure 5.2, showing the effect of the $\phi$ parameter. Instead of using algorithm 3 directly, the resulting distribution as shown in Figure 5.2 could also be precalculated for fixed values for $\phi$ and the number of peers. This can improve the performance of an implementation somewhat, but is not critical in simulation.




(a) $\phi = 0.5$ (b) $\phi = 0.1$

(c) $\phi = 0.05$ (d) $\phi = 0.01$

Figure 5.2: Probability of being picked when having a certain rank for 10 connected peers.

## 5.2 Evaluation

The performance of the different algorithms as described earlier and the impact of variation in the different parameters are evaluated using simulation runs. The simulation incorporates a number of nodes, which represent real-world users. The interactions between these users are then simulated based on discrete events occurring at certain times. These events are stored in a queue that is ordered based on time, and then simulated in that order. Events can themselves insert new events into the queue, creating an enduring process. Each node is an instance of a node class, and events primarily consist of functions of nodes being called. In this manner, the

distributed system is simulated in a single thread. This means the run time of the simulation does not have a one-to-one correspondence to the simulated time, as the run time depends on the amount of events generated by the simulation. The main benefit of this architecture is that all events are executed in order, guaranteeing none are dropped due to a lack of resources. The downside is that it is not possible to effectively make use of distributed computing power meaning simulations will generally take a long time. To deal with this problem the simulator has the option to base the simulation on a limited number of blocks. This subset is selected chronologically, such that it forms a coherent set. The code for this simulator is available on GitHub [1].

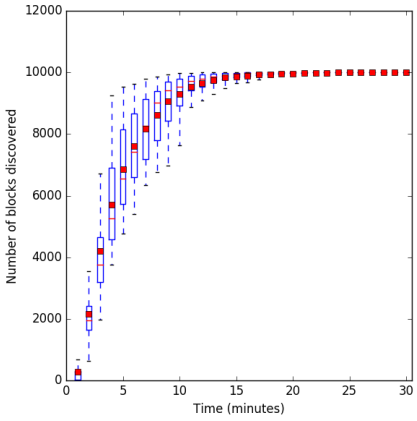
To evaluate the Record Discovery process, we use the simulation to exchange blocks among nodes. The Multichain records the nodes start out with are based on the data set that is obtained through the deployment of the Record Creation protocol as described in Section 4.3, making this a trace-driven simulation. Each real-world node that is known from the release is mapped to a simulated node in the experiments, and given its corresponding blocks. By setting up the nodes in such a manner, the characteristics of the data set are representative of real-world behaviour. After the initialisation of the nodes with their blocks, they will begin walking using the different algorithms. During the process, the exchange of blocks and the load on the nodes is monitored.
Reputation scoring of Multichain peers plays an important role in the discovery of blocks, both as a metric of the accuracy of the local subset of blocks as an approximation of the total set, and as an heuristic as to which peers could provide relevant blocks. An implementation of the accounting mechanism is therefore included in the simulation. This implementation is based on the temporal PageRank method for Multichain from the work of P. Otte [16]. The essential characteristic of this method is the use of transitive trust, enabling the evaluation of focused walking. Aspects of the discovery algorithms that are relevant to the requirements of the record discovery step are assessed using the simulation, as described in the sections below.

### 5.2.1 Teleport probabilities

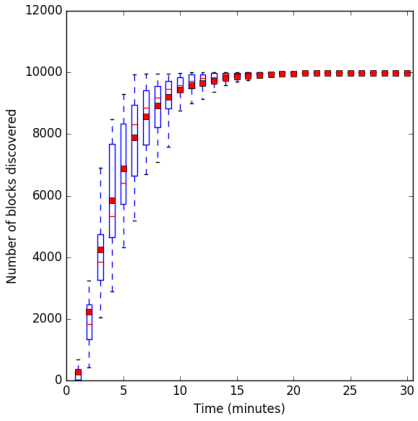
To evaluate the influence of the differences in teleport probabilities between these variants of the random walking algorithms, simulations are run with different values. The graphs in Figure 5.3 show the distribution of the amount of blocks each node has over time. For every minute in the simulation, this distribution across the nodes in the simulation is indicated by a boxplot. In this simulation the amount of blocks is limited at 10000 blocks. The graphs show that due to the special nature of walking in Dispersy, the different teleport probabilities do not alter the process

[1] `https://github.com/pimveldhuisen/multichain-walker-simulation`

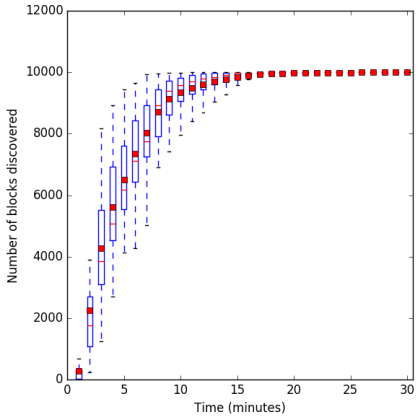
of acquiring blocks radically. To limit the number of variables we use a teleport probability of 1 for the rest of the experiments. This simplifies the analysis of other variants, and implementing this choice in practice allows for simplification of the code.




(a) Teleport probability =1




(b) Teleport probability = 0.5




(c) Teleport probability = 0.25

Figure 5.3: The block collection process using different teleport probabilities.

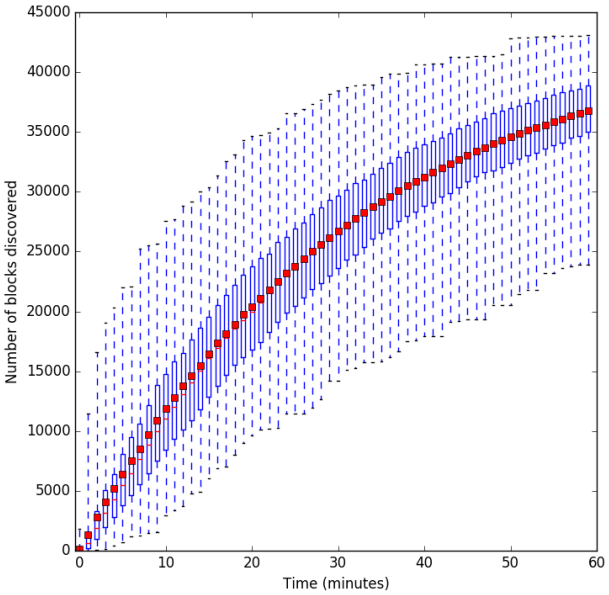
### 5.2.2 Convergence

The goal of the discovery algorithm is to acquire blocks. The most basic metric is thus the progression of the acquired blocks over time. This progression is shown is Figure 5.4a for the random walk and in Figure 5.4b for the focused walk. For each point in time a boxplot is constructed, which indicates the distribution of blocks available at the different nodes. In these boxplots the red line indicates the median of the distribution, the box the borders of the first and third quantile, the whiskers the maximal and minimum values, and the red square the average. The

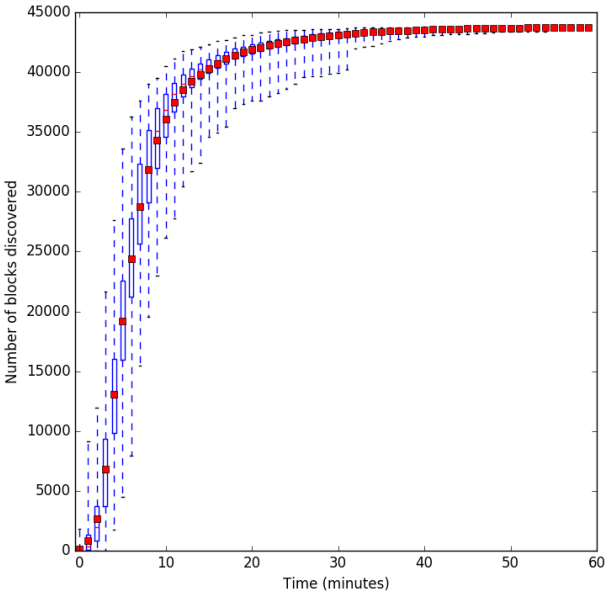
figures shows that in both cases the amount of blocks monotonically increasing; at a later time step, nodes will have either more blocks or the same amount of blocks available in their database. It is also visible that the amount of blocks available for each nodes trends towards a limit; this limit is the amount of blocks in the data set. This means that as a result of the relatively small network and the fact that no new blocks are created during the simulation, nodes develop towards having full information of the network, i.e. having all available blocks in their database. Significant differences are visible between the two algorithms. It is clear that the focused walking results in a much faster acquisition of blocks, which indicates the algorithm is better at realising the goal of the record discovery step.

### 5.2.3 Efficiency

As mentioned in Section 5.1.3, not only the amount of blocks gathered are important but also the relevance of the blocks gathered. One way to take into account the relevance of blocks is to use the accounting mechanism, which is the next step in the Multichain system. A block is more relevant if it has a larger impact on the scores assigned by the accounting mechanism. Since the last step in the progress, the allocation policy, is assumed to operate based on the comparison of scores of different peers, rather than the absolute values of the scores, we are mainly interested in the ranking of peers. If we first consider the full set of records we can use the accounting mechanism to assign scores to each peer based on this set. By then ordering the peers based on their score, we create a *'true'* ranking of all peers. Since the accounting mechanism used by the peers to calculate scores is dependent of the location of the peer in the interaction graph, each peer has its own point of view of the network, and thus its own unique ranking of peers. We can consider this ranking of peers based on all the blocks in the system as the *ground truth*, the most accurate ranking the system could possibly provide. However at the start of the experiment each peer only has its own chain, containing transaction that directly involve the peer. Based on this subset of blocks, the peer can also calculate the score and ranking of all known peers, but this ranking will not be as accurate as the ranking based on the full set of blocks. As the peer gathers more blocks during the experiment, the accuracy of the ranking will improve, and the ranking will converge towards the ground truth. To evaluate the performance of the record collection process we can compare the ranking based on the collected subset of blocks versus the ranking based on the complete set of blocks. Comparing the rankings is however non trivial, since some of the peers in the complete ranking will not occur at all in the subset ranking. While a lot of research has been done regarding the measurement of ranking similarity [28], most of it is specific to rankings that are in the same domain, i.e. have the same set of entries being ranked. However, when comparing two rankings of peers the domain might differ, since some peers are simply not known. We thus need a metric that can handle such a scenario. A




(a) Using the random walking algorithm.




(b) Using the focused walking algorithm ($\phi = 0.1$).

Figure 5.4: The block collection process using different algorithms.

PhD thesis by D. Gkorou [29] specifies a metric for ranking similarity that does allow for dissimilar entries. The metric is defined as follows.

Denoting the full ranking as $R$, and the subset ranking as $R_i$, with the reputation scores of the entries as $s$ and $s_i$ respectively, and the rank of an entry in its list as the function $\sigma()$:

$$RS(R, R_i) = 1 - \frac{\Sigma_{u \in R_i} s(u)(\sigma(s(u)) - \sigma(s_i(u))^2}{D} \tag{5.1}$$

where:

$$D = \Sigma_{u \in R_i} s(u)(\sigma(s(u)) - \sigma(s_w(u))^2 \tag{5.2}$$

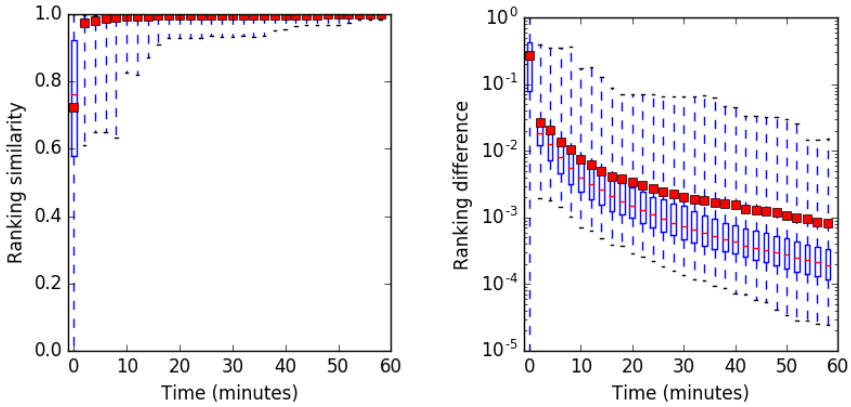
with $s_w$ being the sequence containing the entries of $R_i$ in reverse order.

The ranking similarity is a number between 1 and 0 that indicates how similar two rankings are. A value of 1 indicates the rankings are identical, while 0 indicates they are completely different. Since we use the ranking similarity to compare a ranking based on limited information to a ranking based on complete information, a value close to 1 indicates the record discovery algorithm has performed its job well by collecting the blocks that are relevant to the ranking of peers.

The ranking similarity during the experiments is evaluated for both the random walk and the focused walk. The development of this metric is shown for both algorithms in Figure 5.5 and Figure 5.6 respectively. The graphs show both the ranking similarity and the ranking difference, which is simply 1 - the similarity. The similarity gives an intuitive idea of what is happening, while the difference on a logarithmic scale helps to quantify the differences between the different algorithms. Analysing the graphs we can see the focused walking algorithm outperforms the random walk here even stronger than in the previous metric. This again indicates that focused walking is the best suited algorithm.

### 5.2.4 Load balancing

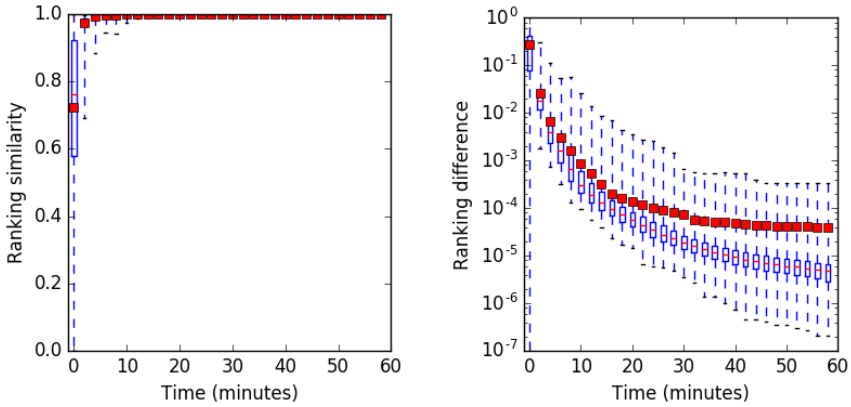
Focused algorithms might visit some nodes more often than others, based on their reputation or their position in the network. While this can lead to efficient discovery of relevant blocks, this imbalance might cause capacity problems for nodes that are visited often. Figure 5.7 and Figure 5.8 show the load for the different nodes by representing each node with a dot. The vertical axis denotes the total number of requests each node has received during the experiment which consists of a period of one hour. It is clearly visible that there are huge differences between the two algorithms, with the focused walking resulting in more variation in the amount of requests per node. Using the random walk, the amounts lie easily between half and double the average, while with focused walking, they vary between almost none to up to 20 times the average. We can analyse these distribution using QQ-plots




(a) Ranking similarity.

(b) Ranking difference on a logarithmic scale.

Figure 5.5: The convergence of the ranking similarity using the random walking algorithm.




(a) Ranking similarity.

(b) Ranking difference on a logarithmic scale.

Figure 5.6: The convergence of the ranking similarity using the focused walking algorithm ($\phi = 0.1$).

comparing their quantiles to known distributions. These plots are shown in Figure 5.9 and they indicate that the random walk leads to a normal distribution, while the focused walking does not. This imbalance in load might be problematic. The problem might be alleviated by the fact that different nodes also have different capacities; it is not unlikely that nodes with a high reputation also have more capacity to handle requests, as this capacity is what brought them their favourable reputation. To some extent, this might even make an uneven distribution of requests

desirable; in such a scenario record discovery would rely more on nodes that have the capacity to support this process. However, the imbalance shown in these figures is too great to lead to a properly functioning network. Some of the more popular nodes would not be able to handle this many requests. It is however possible to change the $\phi$ parameter of the focused algorithm.

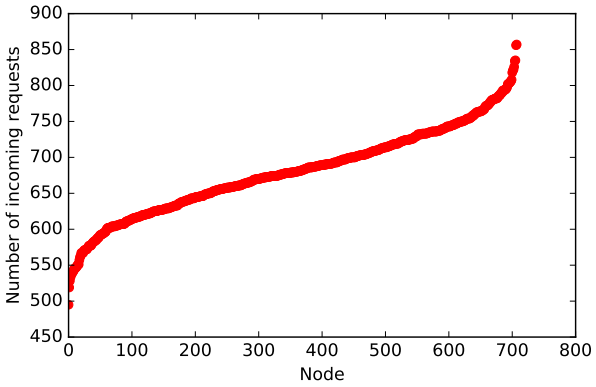



Figure 5.7: Ordered values of the amount of incoming request per node during the simulation using the random walking algorithm.

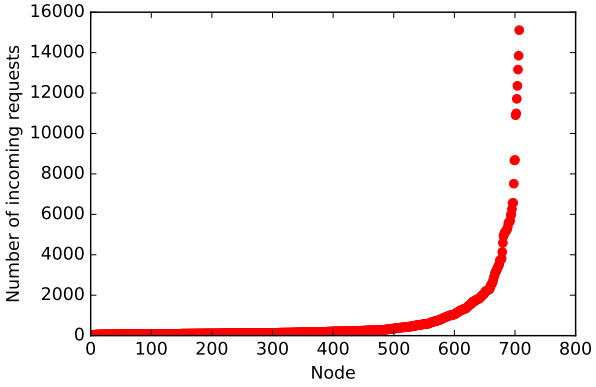
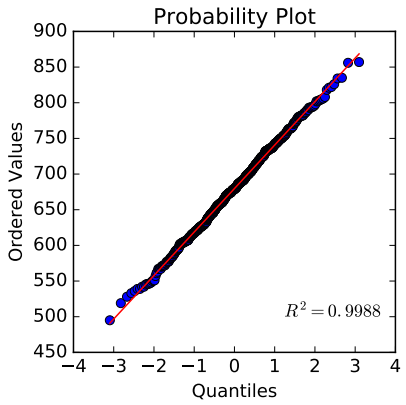


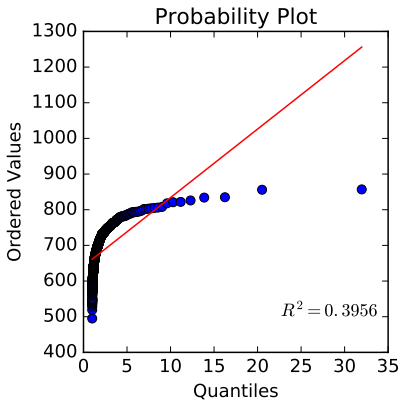

Figure 5.8: Ordered values of the amount of incoming request per node during the simulation using the focused walking algorithm ($\phi = 0.1$).

### 5.2.5 The $\phi$-parameter

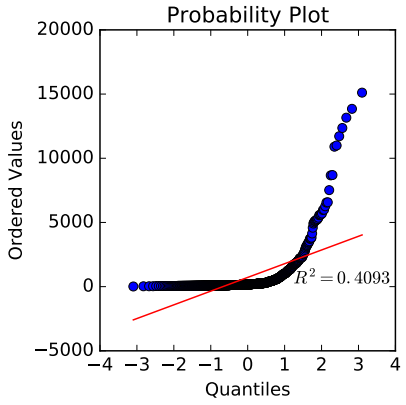
Since the results of focused walking are promising in terms of block gathering, but problematic in terms of load balance, it is both useful and necessary to investigate the algorithm further. The $\phi$ parameter used in the algorithm is very influential and can be varied to change the behaviour of the outcome. Lowering the para-



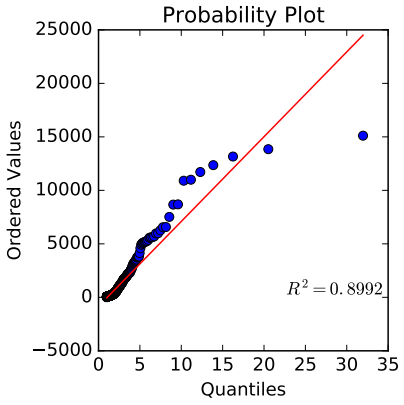

(a) Probability plot comparing the load of the random walk algorithm with a normal distribution




(b) Probability plot comparing the load of the random walk algorithm with a Pareto(2) distribution



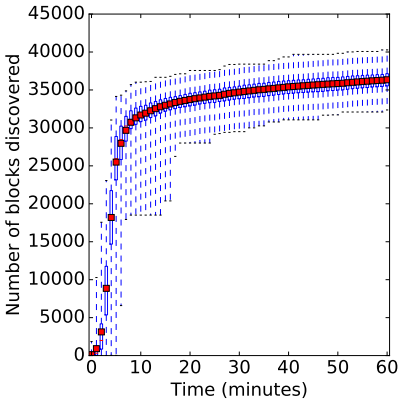

(c) Probability plot comparing the load of the focused walk algorithm with a normal distribution



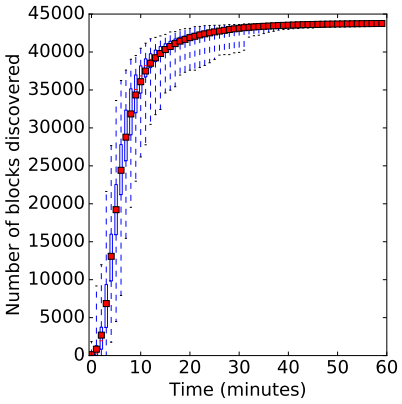

(d) Probability plot comparing the load of the focused walk algorithm with a Pareto(2) distribution

Figure 5.9: Probability plots comparing the distributions of load for different algorithms with standard distributions. The random walk seems to result in a normal distribution, while focused walking results in a distribution that more resembles a Pareto distribution.

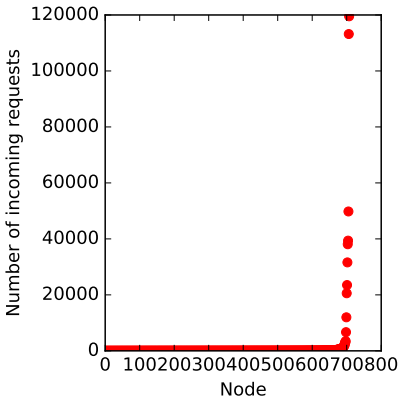
meter makes the walk less focused, which should improve the load balance, but this could come at a cost of reduced speed of block gathering and ranking convergence. To evaluate these effects a number of simulations have been run with different $\phi$-values. The results are shown in Figure 5.10 and Figure 5.11. Note that for the plots of the load distribution, the scale of the vertical axis varies widely, since the distribution itself varies widely.



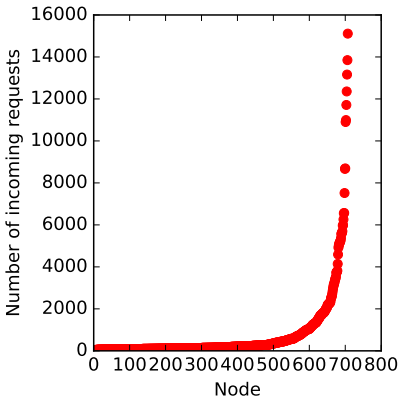

(a) Blocks over time



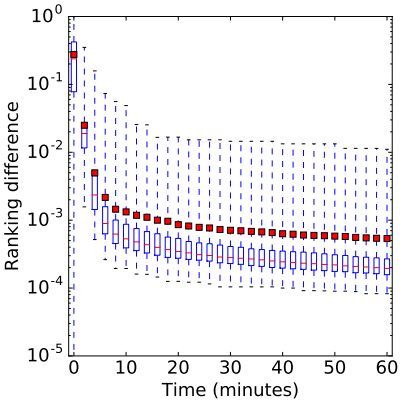

(d) Blocks over time




(b) Load distribution

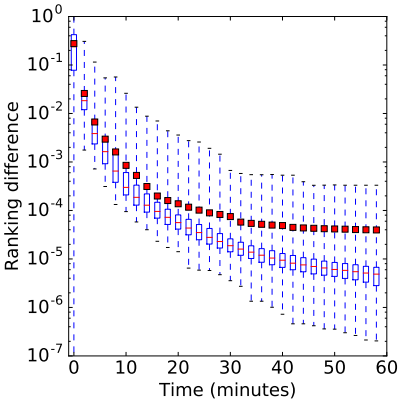



(e) Load distribution




(c) Ranking difference

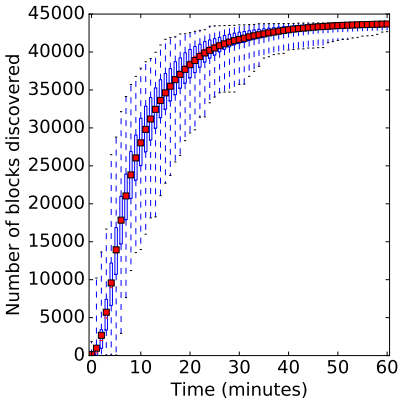
$\phi = 0.5$



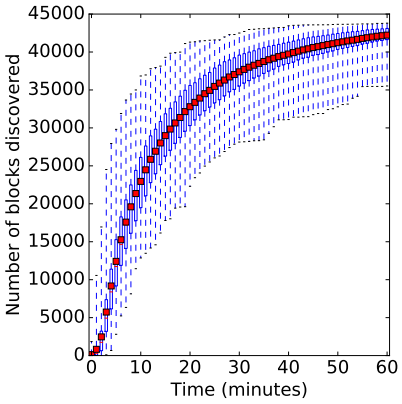

(f) Ranking difference

$\phi = 0.1$

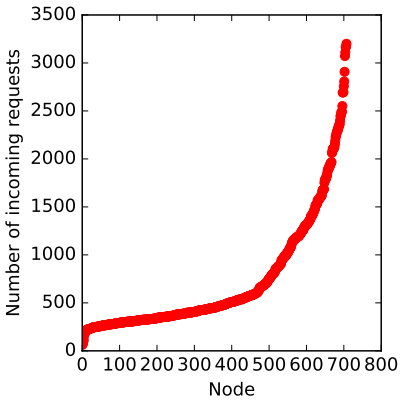
Figure 5.10: Results of the focused walking algorithm with $\phi = 0.5$ and $\phi = 0.1$



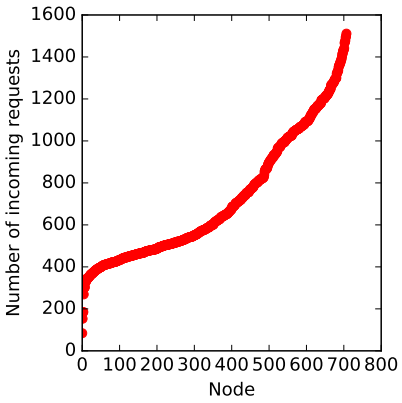

(a) Blocks over time



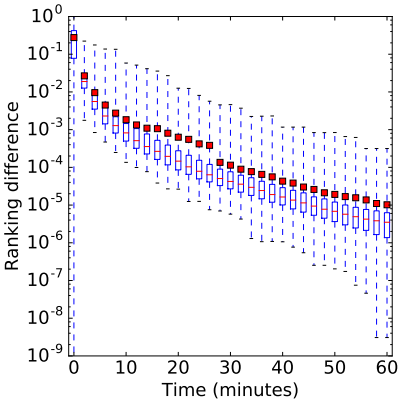

(d) Blocks over time




(b) Load distribution

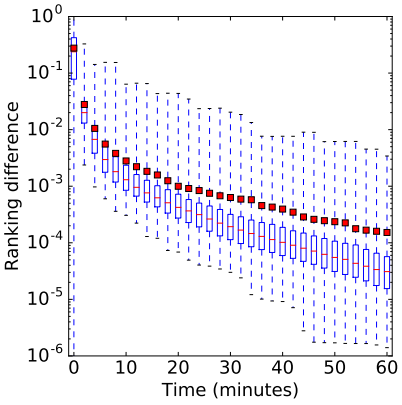



(e) Load distribution




(c) Ranking difference

$\phi = 0.05$




(f) Ranking difference

$\phi = 0.01$

Figure 5.11: Results of the focused walking algorithm with $\phi = 0.05$ and $\phi = 0.01$

We see that for a very high value of $\phi$ the graph is not fully explored, and the nodes get to a point where they are saturated. This also means that for some nodes, the rankings do not converge, and some inaccuracy about their reputation will remain. These inaccuracies will likely increase when the networks gets bigger, as the part of the graph that is explored becomes a smaller fraction of the total graph. While collecting a limited amount of blocks is very efficient in terms of storage space, the inaccuracy in determining the reputation is problematic. For optimal exploration $\phi$ should thus be lower. The results for $\phi = 0.1$ are better in these respects: all blocks are eventually explored, and the ranking converges. However, load balancing is still problematic for this value of $\phi$. Lowering the value further to $0.05$ results in a much better load, while the performance remains fairly strong, in particularly for the convergence speed of the ranking similarity. Therefor, an optimal value for $\phi$ lies likely around $0.05$, depending on the exact characteristics of the network. When evaluating the algorithm with a value of $0.01$ the behaviour is, as expected, very similar to the random walking algorithm, with excellent load balance, but poor performance.

### 5.2.6 Method Limitations

The simulation used to obtain these results is a representation of the network as it would behave running the Multichain system. However, the correspondence between the simulation and the real-world scenario is of course not perfect. A number of real-world factors are not taken into account in this simulation. In the simulation, all the peers are online for the duration of the experiment. However, in the real world sharing blocks would be hampered by churn; people constantly go online and offline, leaving and entering the network. Furthermore, many peers in the real world are connected to the internet through a network address translation (NAT) system. NAT allows multiple internet users to share the same IP address, but this feature implies reduced connectivity. This might mean that some peers might not be able to connect to each other, making it impossible to share blocks directly. Another limitation in the simulation is the fact that no new blocks are created during the run. This means the process is done at some point, when each peer has all the blocks. In the real world, blocks would continuously be created, never allowing the discovery process to be completely done. This also makes a major difference for the usefulness of revisiting a peer that has been visited previously.
An issue with the data set is the role of exit nodes. These exit nodes are servers hosted by the Tribler organisation and support the network while it is not yet mature. While such high capacity nodes would likely always remain to be active in the network, widespread adoption would make the role of these nodes less prominent. This might make the data set less representative of real-world use when the network would become very popular.

A final point is that in the data set, the system was not yet completed with only the record creation and record discovery steps partially implemented. The feedback

loop that is created is not yet present. The goal of the Multichain is however to change the behaviour of peers, and such a change in behaviour also changes the structure of the graph.

The issues listed above make the simulation not a perfect analogue to the real-world situation, but it is still a useful tool to explore the effects of different types of algorithms. The general effects of the different choices should impact real-world behaviour of the system in a comparable way.

### 5.2.7 Conclusion

The simulation shows promising results for focused walking. Under certain assumptions, the focused walking algorithm with the right parameters performs better than an unfocused walk, by collecting blocks faster, and more importantly, collecting more relevant blocks. By collecting more relevant blocks, the ranking based on the available subset converges faster towards the ranking based on complete information, meaning nodes can more accurately judge behaviour based on limited information. The simulation also shows that when picking the parameter $\phi$, it is possible to keep the load balance at an acceptable level, while still experiencing the benefits of the algorithm to a great extent.

# Chapter 6

# Record Discovery in Tribler

The previous chapter has considered the record discovery step using a simulation. Such a methodology is useful to explore different algorithms and parameters in a controlled environment. Algorithms can be tested using all the centralised data, avoiding the need to deal with all the implementation challenges a distributed system brings. Furthermore, the absence of message handling over the Internet removes overhead, allows the simulation can be executed in a way that is resource efficient and completely deterministic. This makes such a simulation useful to provide insight into the different options, but it provides limited validation of the proper functioning of the steps in a distributed setting. To verify it in this domain, a record discovery algorithm has been implemented in Tribler and tested in a distributed setting using the DAS-5 supercomputing cluster[1].

This experiment is focused on the correct functioning of the mechanism and minimal resource usage. The functioning of the discovery process is thus different from the ones described in the previous chapter. In the Tribler version, nodes only add peers to their list of connected neighbours if there is a shared Multichain record between the two. This means that peers in general only discover blocks from peers that are at most two hops away from them; peers that are in the connected neighbours list are one hop away, and a walk will go to one of their connected neighbours. This limits the breadth of the search, bounding the amount of blocks. It also stimulates the discovery of relevant blocks, as blocks from peers that are far removed in terms of hops are less likely to have a big impact on the reputation of peers in the network. To find initial peers to connect to, the system uses bootstrap nodes, that keep track of all peers regardless of the Multichain connection. This might lead to some exploration of blocks from peers that are more than two hops removed.

[1]The DAS-5 (The Distributed ASCI Supercomputer 5) is a six-cluster wide-area distributed system consisting of several processing nodes designed to provide a common computational infrastructure for associated researchers.

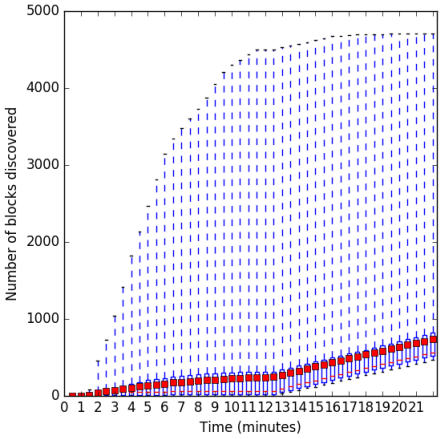
The walking in this version is random, i.e. not focused. While focusing should increase the efficiency of discovery in large networks, it is not included here, to keep the code as simple as possible in its early testing phase, to prevent any unforeseen load-balancing issues, and because the computation of the different reputations is resource intensive. To further limit resources, not all blocks will be transferred upon establishing a connection, but only a limited number per walk step. When a node is introduced to a peer, the node will send a discovery request to that peer. The discovery requests contains the most recent known consecutive interval for that peer, and the number of new blocks it wants to receive. This interval is indicated by two sequence numbers; the head sequence number is simply the sequence number of the most recent already known block of this peer. The tail sequence number is the sequence number of the oldest block, such that all blocks between the tail and head block are known. The receiver of the request will reply with the requested number of blocks around the interval, preferring newer blocks but sending older blocks when there are insufficient new blocks. For example a request could indicate that the blocks between 20 and 40 are already known, and that 5 new blocks are requested. If the receiver's chain goes up to block 43, it would send block 41, 42, 43, 19 and 18. If the original requester would send another request in a different walk step, the know interval would be 43 to 18.[2] If the rate at which new blocks are created is less than the rate at which blocks are requested, continuing requests would lead to the discovery of the complete chain.

The code is implemented in Tribler and run in a distributed setting on the DAS-5. Results are shown in Figure 6.1. The experiments consists of 1 minute during which the nodes are initialised, but do not yet utilise the Multichain. Next, records are created for 10 minutes, to further test the record creation process, but more so to create blocks that are available for discovery. The record creation is a replay of the trace acquired from the release. Each node that has been crawled is mapped to a node in the experiment. Because we do not know the private keys of the nodes in the data set, each node has a new key pair to recreate the blocks. This key pair can then also be used in the record discovery process to verify the availability of common records. Each node in the experiment checks the blocks that are created by its corresponding node in the data set. If the responding party is also included in the experiment, a new request is made to recreate the block in the experiment. This is done at a rate of 1 block per second. After the record creation process, the experiment is paused for 1 minute, where nothing happens. After this, the block discovery process is run for 10 minutes. In the real-world use case these processes would take place simultaneously but they are separated here to provide more insight into the process. The different phases of the experiment are apparent in the block graph which first shows slow growth during the creation phase, then a con-

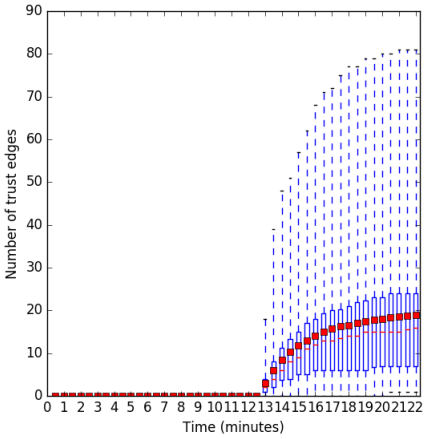
[2]Unless the requester already knew about other blocks. For example, if it already had blocks 18 to 12, the new interval would be 43 to 12

stant level, and the rapid growth in the discovery phase. The number of trust edges, i.e. the connected neighbours, grow quite fast during the discovery process, saturating to a point were each peer has found all online one-hop peers. The load graph shows that the distribution of the number of requests made to each nodes is quite balanced.

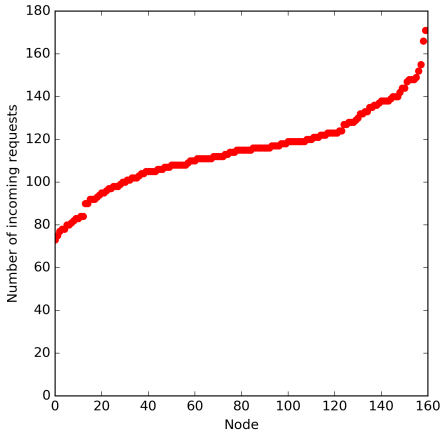
Overall these results show that the record discovery process is functional in a distributed system and leads to the distribution of records across the system. It is not yet clear if this is the optimal manner to do so, nor is this method guaranteed to be attack-resilient. To guarantee such properties, further evaluation is required. The protocol as it is however has shown to be acceptable in terms of load balancing, and can be used in Tribler in conjunction with a first version of the other steps in the Multichain.



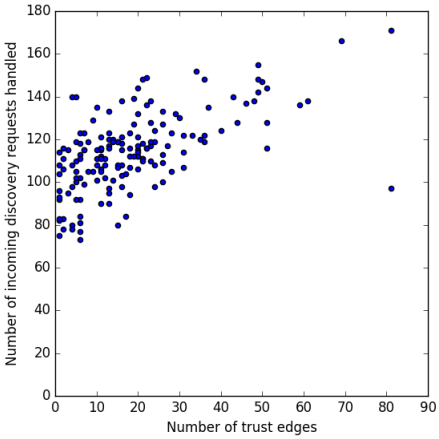

(a) Blocks over time




(b) Number of edges over time




(c) Amount of discovery requests handled by different nodes




(d) Scatter plot relating the amount of trust edges to the amount of discovery requests processed.

Figure 6.1: Results of the record discovery experiment using Tribler

# Chapter 7

# Conclusions and Future Work

## 7.1 Conclusions

The first research question is:

> *How can we create records of historic behaviour in distributed networks in a manner that is accurate, tamper-proof, and scalable?*

The first part of the work in this thesis is attempts to answer this question using the record creation protocol. This protocol has been redesigned based on the prior work, and has now matured into a solid foundation to keep track of behaviour in distributed systems. The protocol has been experimentally validated by experiments and deployment to real users. This system has proven itself functional and accurate in this setting. By design, the protocol is also scalable; record creation happens between two peers, irrespective of the amount of users in the system. Since each peer maintains its own blockchain, the growth of each chain is proportional to the amount of interactions the peer has per unit of time, not the amount of interactions in the entire network. Since the chain is non-blocking, deadlocks cannot occur. These properties make the record creation protocol scalable. The protocol is also designed to be tamper-proof and measures are in place to make it non-trivial to cheat. However, the system has not yet suffered from attacks from malicious users. When the protocol will see more extensive usage and its contents will become more important, the system is likely to see attempts to exploit it. To serve its original purpose, the system must show to be resilient against such attacks. Smaller vulnerabilities in the implementation can likely be patched when they are discovered, but fundamental flaws could render the system useless. Much work, including parts of this thesis, has been put into ensuring that such flaws are not present in the system, but it is not possible to guarantee the absence of attacks that were not considered. While the system has shown to be functional, its attack resilience will have to be proven by exposure to the Internet.

The second research question is:

> *How can we distribute records of historic behaviour in distributed networks in a manner that is informative, efficient and load-balancing?*

The second part of this thesis attempts to answer this question using record discovery system. Since this part of the system had not been previously subject to a lot of research, the work regarding this part was broader oriented, considering the various options available. Records can be obtained from a variety of sources, and different systems can have widely varying effects on the informativeness, efficiency, load balancing and resilience of the system. The different simulations and emulations show that it is feasible to explore a considerable part of the records in the system, such that peers can make an informed estimate of the behaviour of other peers. By making use of a focused walking algorithm during the record discovery process, incorporating the reputation score, the efficiency of the walk can be increased while keeping the load balance acceptable.

The main contributions of this thesis are:

- Identifying four steps that are necessary to create a system that can successfully influence behaviour in distributed systems.
- Formulating a protocol that can be used to create tamper-proof records, improving upon existing work by solving inherent problems in the design.
- Validating the proper functioning of the record creation process in the a distributed environment across the internet, integrated with file-sharing mechanisms.
- Analysing the design space for a record discovery process and showing that a focused approach yields significant benefits.
- Validating the functioning of the record discovery process in a distributed environment.

## 7.2 Future work

We can discern two main areas of future work: completing the Multichain system in BitTorrent networks, and applying its principles in other areas.

For the Multichain system to reach its end goal of influencing behaviour and establishing cooperation, all four steps described in Section 2.2.1 must be implemented. The fourth step closes the feedback loop, giving peers incentives to change their behaviour. Only when this feedback loop is complete, can the Multichain be evaluated in its entirety. Considering the four steps at the present time, we see that the first step is complete. The record creation step has been designed, implemented

and experimentally validated. This thesis also presents an implementation for the second step that is feasible for use in the system. However, the implementation of this step can significantly improved by using focused walking as described in this thesis. The third step was not considered in this thesis, but the work of by P. Otte[16] presents two possible designs for the third step. More research is needed to implement these designs at acceptable computational costs. The fourth step, which attaches consequences to perceived behaviour, has not yet been designed. This step is however relatively simple. Future work should focus on implementing these last steps, and combining them into a functional system. The system can then be measured and evaluated, to see if it functions as desired.

File-sharing in peer-to-peer networks is a rather limited field of application, but the concept of cooperation is prevalent throughout society. It is therefor also important to investigate other areas where a system such as the Multichain, or its underlying components can be gainfully utilised. Part of this research is already ongoing, and concepts introduced in this thesis are also applicable in other use cases, such as that of distributed markets.

# Bibliography


[1] G. Hardin, "The tragedy of the commons," *Science*, vol. 162, no. 3859, pp. 1243–1248, 1968.

[2] E. Adar and B. A. Huberman, "Free riding on Gnutella," *First monday*, vol. 5, no. 10, 2000.

[3] F. Le Fessant, S. Handurukande, A. M. Kermarrec, and L. Massoulié, "Clustering in Peer-to-peer file sharing workloads," in *Proceedings of the Third International Conference on Peer-to-Peer Systems*, IPTPS'04, (Berlin, Heidelberg), pp. 217–226, Springer-Verlag, 2004.

[4] R. Thaler, "Toward a positive theory of consumer choice," *Journal of Economic Behavior & Organization*, vol. 1, no. 1, pp. 39–60, 1980.

[5] H. Gintis, "Beyond homo economicus: evidence from experimental economics," *Ecological economics*, vol. 35, no. 3, pp. 311–322, 2000.

[6] B. Cohen, "Incentives build robustness in BitTorrent," in *Workshop on Economics of Peer-to-Peer systems*, vol. 6, pp. 68–72, 2003.

[7] S. Jun and M. Ahamad, "Incentives in BitTorrent induce free riding," in *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, P2PECON '05, (New York, NY, USA), pp. 116–121, ACM, 2005.

[8] M. Yang, Z. Zhang, X. Li, and Y. Dai, "An empirical study of free-riding behavior in the Maze P2P file-sharing system," in *Peer-to-Peer Systems IV*, pp. 182–192, Springer, 2005.

[9] M. Meulpolder, J. A. Pouwelse, D. H. J. Epema, and H. J. Sips, "BarterCast: A practical approach to prevent lazy freeriding in P2P networks," in *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pp. 1–8, May 2009.

[10] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008.

[11] P. D. Bó, "Cooperation under the shadow of the future: experimental evidence from infinitely repeated games," *The American Economic Review*, vol. 95, no. 5, pp. 1591–1604, 2005.

[12] S. D. Norberhuis, "MultiChain: A cybercurrency for cooperation," MSc thesis, Delft University of Technology, December 2015.

[13] M. A. Nowak, "Five rules for the evolution of cooperation," *Science*, vol. 314, no. 5805, pp. 1560–1563, 2006.

[14] R. Axelrod, "Effective choice in the prisoner's dilemma," *Journal of conflict resolution*, vol. 24, no. 1, pp. 3–25, 1980.

[15] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, *et al.*, "Manifesto for agile software development," 2001.

[16] P. Otte, "Sybil-resistant trust mechanisms in distributed systems," Master's thesis, Delft University of Technology, 2016.

[17] A. Madureira, F. den Hartog, H. Bouwman, and N. Baken, "Empirical validation of metcalfes law: How internet usage patterns have changed over time," *Information Economics and Policy*, vol. 25, no. 4, pp. 246–256, 2013.

[18] R. Rivest, "The MD5 message-digest algorithm," RFC 1321, April 1992.

[19] H. Dobbertin, "The status of MD5 after a recent attack," *CryptoBytes*, vol. 2, no. 2, 1996.

[20] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD.," *IACR Cryptology ePrint Archive*, vol. 2004, p. 199, 2004.

[21] X. Wang, Y. L. Yin, and H. Yu, "Finding collisions in the full SHA-1," in *Annual International Cryptology Conference*, pp. 17–36, Springer, 2005.

[22] "Announcing the first SHA1 collision." `https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html`, Februari 2017.

[23] N. Zeilemaker, B. Schoon, and J. Pouwelse, "Dispersy bundle synchronization," *TU Delft, Parallel and Distributed Systems*, 2013.

[24] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," *Information processing letters*, vol. 31, no. 1, pp. 7–15, 1989.

[25] L. Lovász, "Random walks on graphs," *Combinatorics, Paul erdos is eighty*, vol. 2, pp. 1–46, 1993.

[26] A. Awan, R. A. Ferreira, S. Jagannathan, and A. Grama, "Distributed uniform sampling in unstructured peer-to-peer networks," in *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 9, pp. 223c–223c, IEEE, 2006.

[27] A. Mohaisen, A. Yun, and Y. Kim, "Measuring the mixing time of social graphs," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pp. 383–389, ACM, 2010.

[28] S. E. Stemler, "A comparison of consensus, consistency, and measurement approaches to estimating interrater reliability," *Practical Assessment, Research & Evaluation*, vol. 9, no. 4, pp. 1–19, 2004.

[29] D. Gkorou, *Exploiting Graph Properties for Decentralized Reputation Systems*. PhD thesis, Delft Univeristy of Technology, 2014.