Leveraging blockchains to enforce cooperation

Pim Veldhuisen

Delft University of Technology

Leveraging blockchains to enforce cooperation in distributed networks

Master's Thesis in Computer Science

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

Pim Veldhuisen

6th March 2017

Author
Pim Veldhuisen

Title
Leveraging blockchains to enforce cooperation

MSc presentation
TODO GRADUATION DATE

Graduation Committee
TODO GRADUATION COMMITTEE Delft University of Technology

Abstract

TODO ABSTRACT

Preface

TODO MOTIVATION FOR RESEARCH TOPIC

TODO ACKNOWLEDGEMENTS

TODO AUTHOR

Delft, The Netherlands
6th March 2017

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 organizational 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 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. Another 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 1986 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.

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 would 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 behaviors. 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 centralizes 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 realize fruitful behavior not through binding rules imposed by a centralized 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.

BitTorrent is one of the most used systems with a successful decentralized incentive scheme [4]. 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. 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 [5]. Distributed global networks could still perform much better when all peers cooperate. The key to realizing this behavior 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 use a persistent score for each user [6]. This was found to stimulate the desired behavior among peers playing by the rules, but encountered problems with whitewashing; behavior 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 aggrandized reputation to others. As a result, a secure decentralized incentive system that incorporates history is still an open problem.

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. 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 utilized 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 [7]. 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 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 [8]. The concept also shows great promise in recording cooperative behavior. 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 historic reliability.

A preliminary version of the Multichain has been implemented in code by S. Norberhuis [9]. 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 an all-round blockchain based incentive system that can influence behavior in actual systems. It also takes some steps 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 generalized 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 is representative of other cases, has been the subject of scientific research in the past, and the Tribler platform provides a way to study the behavior of actual users.

The challenges and properties of the problems involved in realizing the Multichain system 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 **??** 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 summarizes the results of the research in a conclusion, and surveys future steps for the Multichain system.

Chapter 2

Problem Description

The goal of the multichain system is to enforce cooperation of peers in a decentralized manner. In order to achieve this, the peers should be incentivised is such a way that their goals are aligned with the goal of the network. Behavior 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 achieve this, the Multichain system must be influenced by the events that occur in the underlying network. These events must then result in some sort of judgement, which then affects the behavior of the peer in the original network. This creates a feedback loop, which, when designed properly improves the behavior in the underlying network. This concept is shown in figure 2.1. In the figure, the red interactions represent the underlying network in which we want to change behavior. 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 behavior in the network.

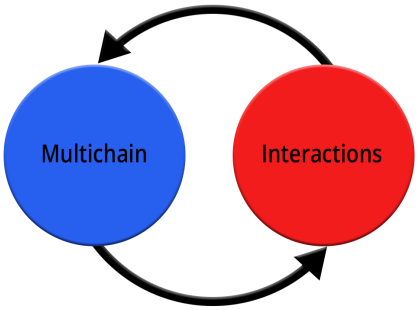


Figure 2.1: The feedback loop between the Multichain system and the underlying network.

To reward certain behavior in a peer-to-peer network, other peers must be made aware of this behavior. In the network, behavior consists of interactions between peers. We therefore need some sort of record-keeping system that keeps track of interactions between peers. This step is titled *record creation*. Peers must then discover the records of other peers, to be able to consider their behavior; *record discovery*. When a sufficient amount of information is available, a peer can analyze the behavior of other peers, and make some kind of judgement on it. A concise way of judging behavior is assigning a numerical score to each known peer. The score is assigned base on an *Accounting Mechanism*. The final step then attaches consequences to the perceived behavior by initiating (or avoiding) new interactions, and by allocation various amount of resources to it based on the perceived priority based on the reputation score. This is called the *Allocation Policy*. These steps are shown in figure 2.2 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 to peers. The record creation process also involves communication, since both peers must agree on the content of the record. This is again 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 behavior 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.

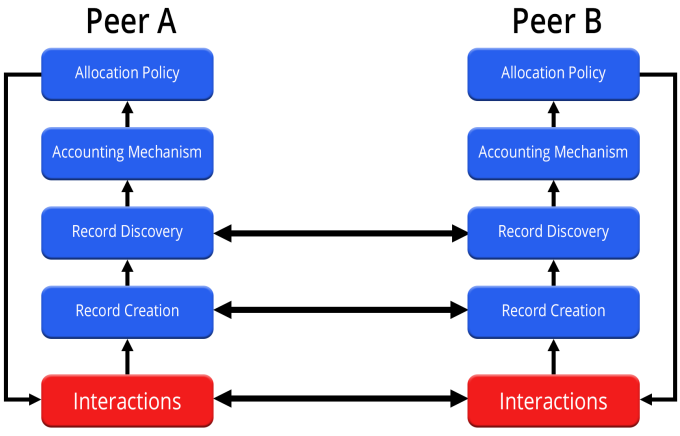


Figure 2.2: The different steps taken by the multichain system to enforce cooperation of peers.

Since the different steps can operate somewhat independently of each other, 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 [10], 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 behavior 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 behavior 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 layer [11] has mathematically 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 [9], 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 layer has until now remained unexplored in the context of multichain, and this thesis evaluates different protocols that could perform this function.

We first consider the requirements for the record creation layer. The first, most obvious requirement is that the record creation process is sufficiently accurate. In order to reward certain behavior, me must first have records faithfully describing the behavior. 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 behavior 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 behavior. 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 is proportional to the square of the users. 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 realized. This means 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 will 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 behavior of other peers. This means 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 layer 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

Record creation is not enough for rewarding good behavior; this behavior 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 behavior 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 realizing 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 behavior 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 way this may be desirable behavior, since having a lot of records shows a high capacity for interactions, which might be an indicator of a high capacity for handling requests for records. However, when specific nodes are polled too much, this might lead to capacity problems, disturbing the functioning of the network.

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

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

For both layers 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 operate autonomously. This constraint provides a lot of challenges regarding the availability and trustworthiness of information. Another premise is the principle of open enrollment; 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.

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. To store these records in a reliable and tamper proof manner, blockchains are used.

3.1 Blockchains

The interactions between peers are stored in blocks. The blocks contains the transaction data that is relevant to the behavior that the system aims to influence. In our 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.
While individual blocks can be used to prove that certain interactions have occurred, this does not ensure an accurate representation of the behavior 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 behavior, 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 a 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, indicated 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. A more advanced strategy to misrepresent behavior is to replace certain unfavorable blocks in the blockchain with other more

favorable 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. 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 means it is much harder to make changes to previous blocks without anyone noticing.
The resulting record of interactions should thus form a complete overview of the behavior of all nodes that can be used as the foundation for an evaluation of reputation.

3.2 Concurrency

Peers can have multiple simultaneous interactions with different counterparties. This could potentially can cause 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: since 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 $1/t$. 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 amount of interactions that can be recorded to 30 per minute, while there are many nodes that have 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 arsing 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 make it clear that the Multichain should feature concurrency; being able to have multiple outstanding requests at the same time. The previous version of the multichain did not have this feature, making it unsuitable for deployment on a large scale. The work in this thesis makes the multichain non-blocking, solving the issue.

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 allows 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.

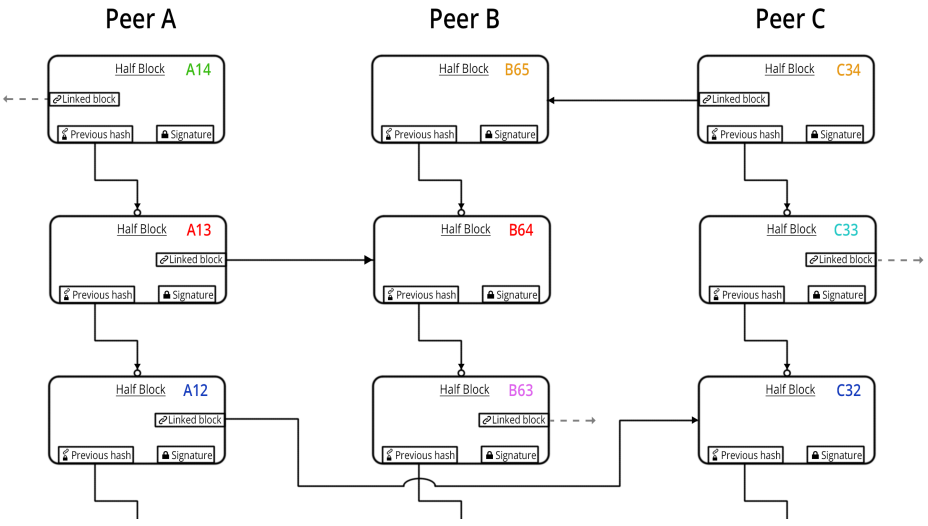


Figure 3.1: A simplified representation of some blocks in the multichains of three peers .

Figure 3.1 illustrates the structure of the blockchain, showing three half blocks per chain for three peers in a simplified format. Half blocks can be identified by the combination of 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 the implementation these fields are a 74 byte public key and a 4 byte sequence number, but in this figure 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, 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. 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.

3.3 Block protocol

The creation of the 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 will initiate the process, and this peer then becomes the requester. The requester will create 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 will only include 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 of his previous block. The whole block is then signed using the keypair of which the public key has already been included. The requester will then send 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 will validate the incoming block based on its perception of the interaction, its own chain, and its available knowledge of the chain of the responder. If everything seems valid the responder will create a half block following the same procedure as the requester, with the notable difference that the linked sequence number will actually contain the sequence number of the corresponding linked block, namely the block which the requester has just sent. The responder will then send 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.

The different fields present in a half block in the Tribler use case, including their type and their size are enumerated in table 3.1

	Field	Type	Size (Bytes)
	Transaction:		
1	Bytes uploaded	Unsigned integer	8
2	Bytes downloaded	Unsigned integer	8
3	Total bytes uploaded	Unsigned integer	8
4	Total bytes downloaded	Unsigned integer	8
	Identity:		
5	Public key	Character array	74
6	Sequence number	Unsigned integer	4
	Counterparty identity:		
7	Linked public key	Character array	74
8	Linked sequence number	Unsigned integer	4
	Validation:		
9	Previous hash	Character array	32
10	Signature	Character array	64
	Total:		**284**

Table 3.1: Data contained in a half block

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, it seems obvious that simultaneous initiation is a scenario best avoided. To achieve 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 of 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. Assuming that both parties are honest, 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 defenses

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 behavior is seen as unfair, 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 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 idea 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. (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.) 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, both because no reputation can be reliably ascertained form the multichain, and to discourage lying in general.

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.

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 a rabbit hole that leads nowhere.

One way to deal 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 cheaters to polish up their reputation is to hide blocks that portray them negatively. When someone requests some blocks from them to gauge their reputation, the might send only their best blocks. The blockchain structure however defends against such tricks; since every block has a sequence number, anyone can request specific blocks, which the cheater cannot simply refuse to provide without being detected as a cheater. Furthermore, since every block is available to both parties involved, a cheater can never prevent distribution of the block by the counterparty.

Branching To hide some negative records from others regardless of the blockchain structure, attackers might branch their multichain. In this 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, some peers will eventually find the two conflicting blocks. Since the blocks are clearly in violation of the protocol, and are both signed by the attacker these for 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 know

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 [11].

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 [1], 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 [12], shown to be vulnerable in 1996 [13], broken in 2004 [14], 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 theorized to be vulnerable to even more efficient attacks [15]. Since SHA-1 might be vulnerable to attacks in the future, the protocol was upgraded to use the safer SHA-256 hash [2]. This decision was later supported by the release of a practical attack in SHA-1 [16].

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

[2] See: `https://github.com/Tribler/tribler/commit/f353156cd0de769f6874d375800a34cb0c75e7f0#diff-df44fd1a9b4533421509f4ddf8ddbaa4`

Chapter 4

Evaluation of the Record Creation functionality

The goal of this thesis is not to merely contrive 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. During the software development, the code is continuously tested for correctness using unit tests. When features and aspects of the software are somewhat complete, they are evaluated using experiments where the software runs integral. Finally, the software is deployed and released to users to see its real world performance.

4.1 Implementation

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 historical record.
Like all Tribler communities, the Multichain community uses the Dispersy framework to exchange messages among peers [17]. Dispersy enables the Multichain to send messages to other peer 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 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 SQLite [18].

4.2 Experiments

In order to verify the workings of the protocol, several experiments were run in a controlled environment were real world situation were simulated. These experiments helped 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. The initial idea was to do this at regular intervals in the amount of data transferred; when a certain threshold of outstanding data is reached, a block would be signed. The threshold was originally set at 1 MB, but due to the amount of blocks this would create, it was soon increased to 5 MB. An experiment was done using this policy by running several instances of the code simultaneously. During the experiment the instances would communicate with each other using Internet protocols mimicking the real world scenario of globaly diverse peers. A graph of the Multichain produced by this experiment is shown in figure 4.1.
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 origin 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.

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 also between them. 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

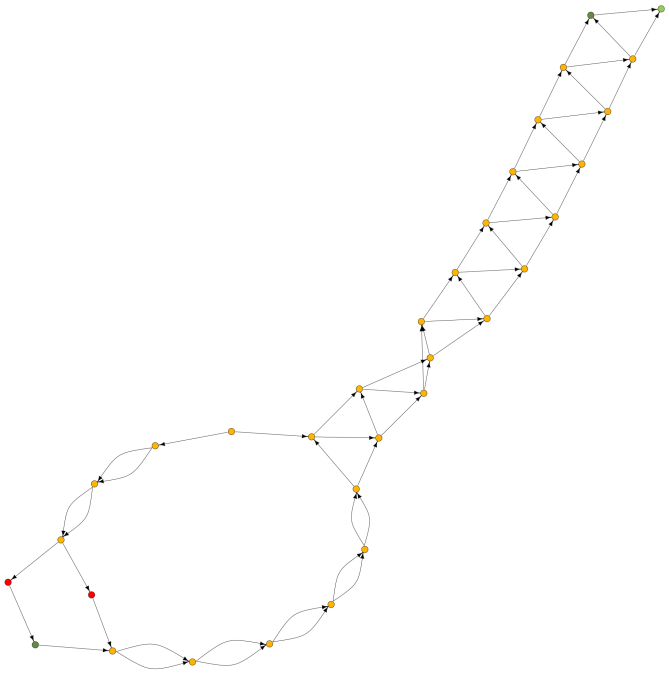


Figure 4.1: A block graph showing the blocks created during a few file transfers using the 5 MB block limit.

occurrence but here three peers are involved; one of them interacts with two others simultaneously. This peer alternately creates blocks with both peers.

While covering a continuous interaction by multiple smaller blocks results in a lot of entanglement, improving security, it also results in a lot of blocks being created. This is costly in terms of processing time, bandwidth and data storage. With a 5 MB limit, 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. To achieve this, the signing process is initiated only when an interaction has come to an end. 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.

The next step in the experimentation process consists of increasing the amount of nodes that run in parallel. A large scale experiment was run with 100 nodes interacting simultaneously. A Multichain resulting from such an experiment is shown in figure 4.2. As visible, it is non-trivial to show the structure in a clear layout, without crossing edges. The graph is plotted here using the Kamada-Kawai algorithm [19] which seems to produce the best results among general purpose layouts. Note that there are still chains of repeated interactions, as a result of single interactions being split into multiple blocks. One such repetition chain extrudes from the graph on the left side of the image. The scaled-up experiment showed the cluttering resulting from small blocks, and emphasized the need for fewer blocks.

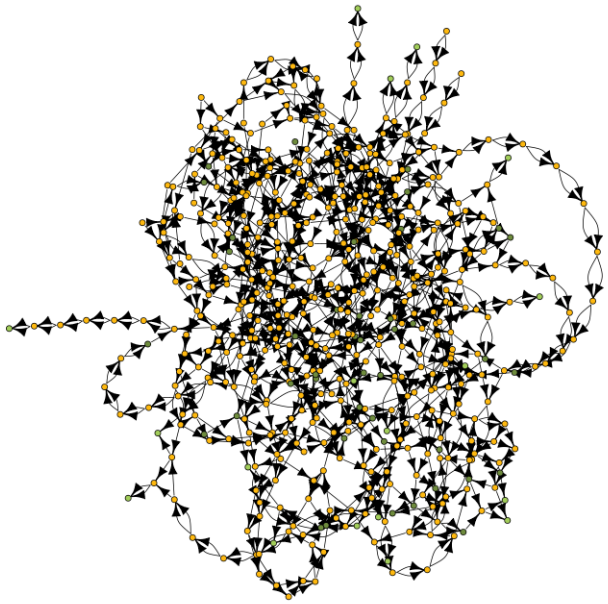


Figure 4.2: A block graph showing the blocks created during an experiment with 100 nodes creating 1 MB blocks.

To implement a policy where an interaction corresponds to a single block, the tunnel close event is used. Since downloads in Tribler occur in encrypted tunnel that guarantee privacy, the end of an interaction correspond to the closure of such a tunnel. The tunnel closure is a natural point to trigger the signing of a block. This policy was implemented and tested in experiments. A Multichain resulting from such an experiment is shown in figure 4.3.

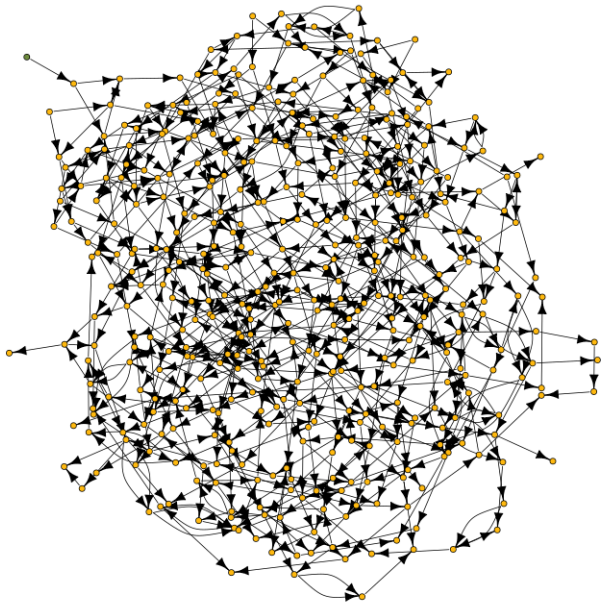


Figure 4.3: A block graph showing the blocks created during an experiment with 100 nodes using the tunnel close event.

The Multichain resulting from this policy has fewer blocks, and a simpler structure. Note that there are no longer chains of repeated interactions.

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 are shown is figure 4.4. While the figure is just a single run on 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. This is enough 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.

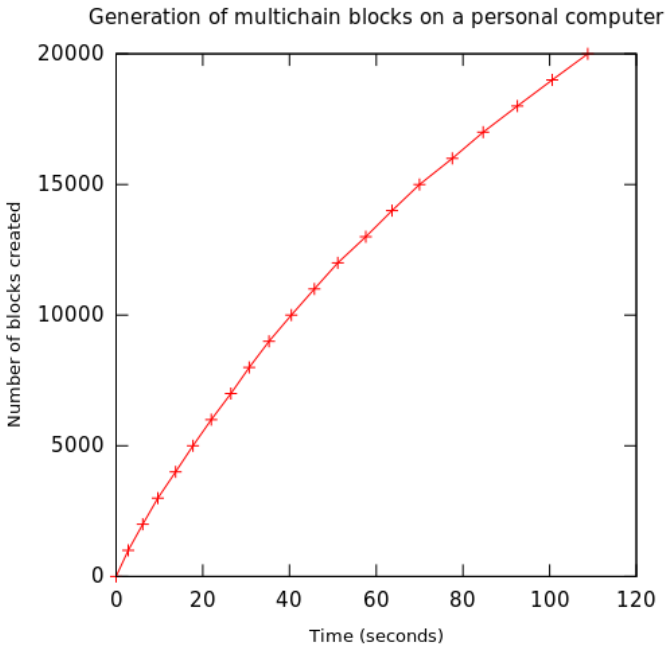


Figure 4.4: The amount of half blocks created over time during an experiment on a personal computer.

4.3 Deployment

While the experiments in simulated experiments provide some validation of the protocol, the real test for the protocol consists of it's 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 summarized as the 'do no harm policy'.

4.3.1 Internal testing

The first integrated test was conducted with an early version of the software among members of the Tribler. Although the number of peers and the geographical spread was very limited, this for 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 fundamentally speaking possible 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 here is however so extreme, that it 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 4.5. 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 behavior of one of the SQL-queries used in certain scenario's.

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 problems since this was only an internal release.

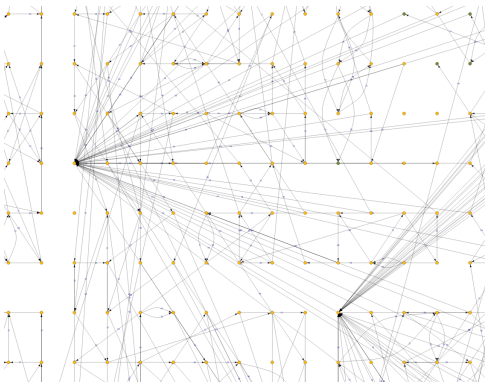


Figure 4.5: An excerpt from an early multichain block graph, highlighting internal inconsistencies.

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, an 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 would be 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 behavior of the different nodes. To check the operation of the Multichain, E.M. Bongers another member of the Tribler team, implemented crawler functionality which enables one of our machines to gather all blocks from the network. This results in a dataset that can be used to analyse the Multichain that is created by the network using the Alpha release.

A snapshot was taken on the 10 of august 2016, containing all blocks created before that date. Some statistics resulting from the dataset are shown in table 4.2. The first 256 blocks that were gathered by the crawler are displayed in figure 4.6. The nodes in this graph are positioned using a Hilbert 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

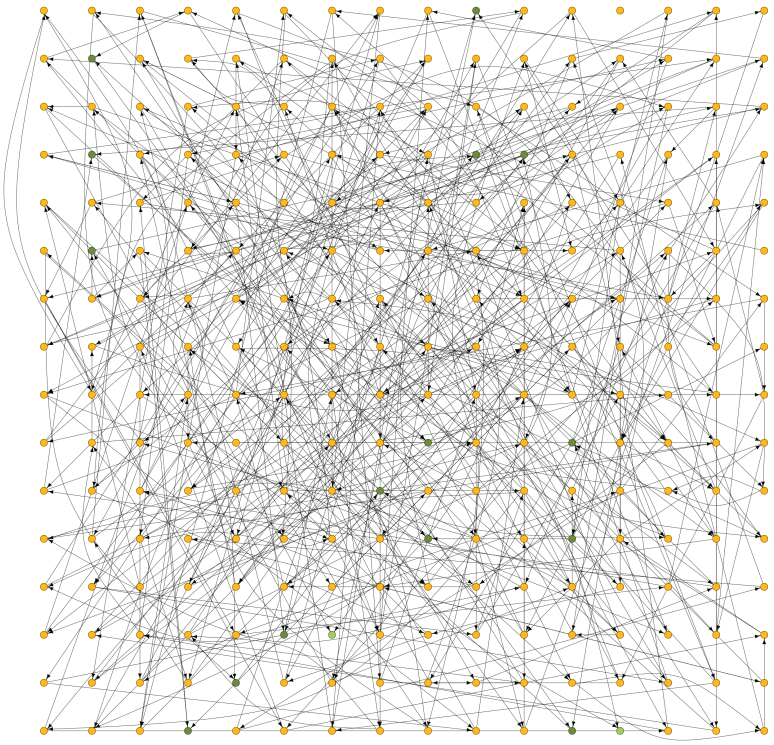


Figure 4.6: A Multichain block graph showing the 256 first crawled blocks from the Alpha release.

Chapter 5

Record Discovery Algorithms

5.1 Design

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 behavior 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. Nodes achieve this by finding out which nodes are currently online, and requesting 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 it's own chain, it should always be able to handle requests for it's own blocks. For example, if node A wants to find out about node B, it could ask node B for it's 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 from node B. However, node B might not always be able to provide such blocks.

5.1.1 Continuous exploration

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 provide fake information, 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 behavior. 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.
The different algorithms and parameters that can be used to explore the Multichain are investigated using a simulation. This simulation is based on the results of the experimental release of the record creation protocol as discussed in section 4.3.2. Each real world node that is know from the release is mapped to a simulated node in the experiments, and given it's corresponding blocks. A simulation of block exploration is then run using these nodes. By setting up the nodes in such manner, the characteristics of the data set are representative of real world behavior.

The simulation is based on discrete events occurring at certain times. These events are stored in a queue that is ordered based on time. Events are then simulated in that order, and events can themselves insert new events into the queue. 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. 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. All events are executed in order, guaranteeing none are dropped due to a lack of resources. The code for this simulator is available on GitHub [20].

5.1.2 Random walks

A commonly employed strategy to explore a graph is that of a random walk [21], [22], [23]. Here information is gathered by querying nodes, then traversing one of it edges, 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 a binomial distribution 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 below.

```
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)
```

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, 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 or one, or equivalently a teleport probability of 1. This removes the need for a conditional branch. The resulting pseudo-code below shows that the resulting code is significantly simpler.

```
next_node = pick_random_from(connected_neighbours)
walk_towards(next_node)
```

To evaluate the differences between these variants of the random walking algorithms, [Graphs of teleport is 1,2,3]
Since edges are added, it doesn't make a lot of difference. We just use teleport is one for the rest of the experiment.

5.1.3 Focused walking

One way to potentially obtain more relevant information is to not pick a next node to explore at random, but to prefer nodes with a higher reputation from your perspective. Such a node is more likely to have information regarding other nodes with

high reputation, and thus furthers the general goal of record discovery: obtaining information about the nodes with the highest reputation. A potential downside of this manner of walking is that it will lead to more requests to nodes that have a good reputation towards most of the nodes in the network. This could mean that well-behaving nodes are overloaded by the record discovery system.

Generalized explanation: For any transitive scoring algorithm, the neighbours of an node with a high ranking have a higher expected ranking than a random node in the network. It thus makes sense to walk here. However this might cause load balancing problems, and, if done to strict, result in convergence to a non-optimum point.

To focus on walking towards nodes with a higher score, the score for each available live peer is first calculated, and these peers are then sorted in order of their score. The algorithm then picks the peer with the highest score with a probability α. If it does not pick the peer with the highest score, it moves on to the next peer, and picks it with a probability α. 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 below.

```
index = 0

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

next_node = ranked_connected_neighbours[index]
walk_towards(next_node)
```

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 α. 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 α 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, an α that is to high will have adverse effects. The walk will in many cases converge rapidly to a steady situation, where only a small subset of nodes is explored. Furthermore, a high α 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 α.

The algorithm results in a certain distribution

5.2 Results

The performance of the different algorithms as described earlier and the impact of variation in the different parameters are evaluated by experiments. These experiments consist of a simulation of these algorithms in a scenario with a number of nodes, which represent real world users. The multichain records the nodes have are based on the data set that is obtained through the deployment of the record creating protocol as described in section 4.3. 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. The scoring module is therefore included in the evaluation.
Several aspects are assessed when evaluating the algorithms, as described in the sections below.

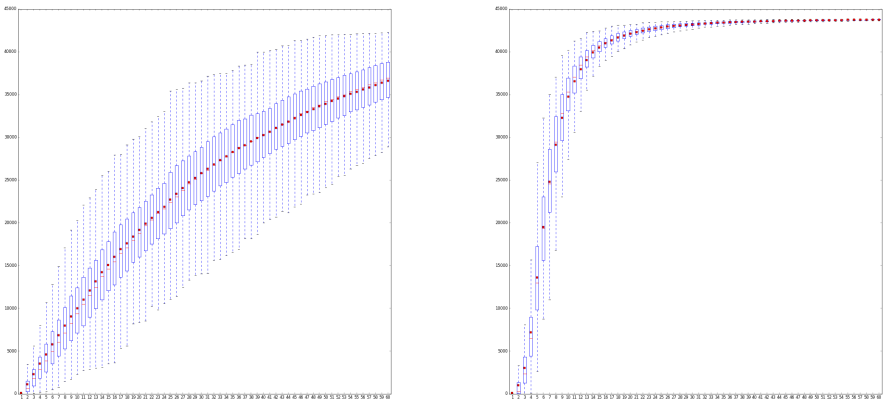
5.2.1 Convergence

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 **??** for the random walk and figure **??** for the biased walk. For each point in time a boxplot is constructed, which indicates the distribution of blocks available at the different nodes. The figures shows that in both cases the amount of blocks monotonically increasing; at a later time step, nodes will have more 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 dataset. 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.

5.2.2 Efficiency

However, the total set of blocks is proportional to the amount of users and the time the system has been running. This means that on a large network, it is not feasible for most nodes to obtain the complete set of blocks. The task of the discovery algorithm thus becomes to discover only the most relevant blocks.

One way to take into account the relevance of blocks is to evaluate the Accounting Mechanism used in 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 operate based on the comparison of scores of different peers, rather than the absolute values of



(a) Using the random walking algorithm. (b) Using the biased walking algorithm.

Figure 5.1: The block collection process using different algorithms.

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 [24], most of it is specific to rankings that are in the same domain, i.e. have the same set of entries being ranked. However, a Phd thesis by Dimitra Gkorou [25] contains a metric of ranking similarity that does allow for dissimilar entries. 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 $\sigma()$, this metric is defined as:

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

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.

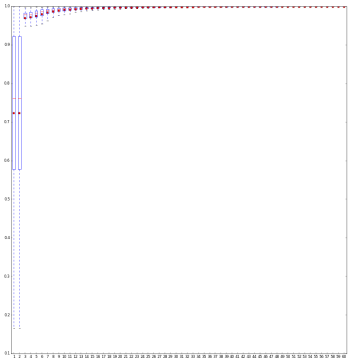


Figure 5.2: The convergence of the ranking similarity using the random walking algorithm.

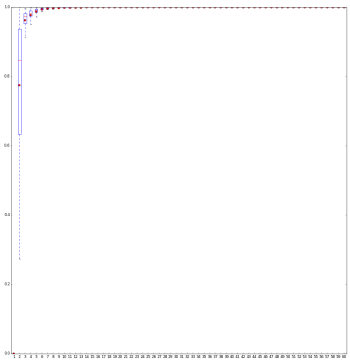


Figure 5.3: The convergence of the ranking similarity using the biased walking algorithm.

5.2.3 Load balancing

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.

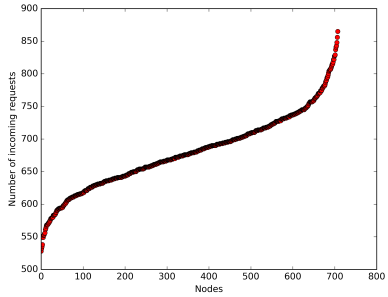


Figure 5.4: Ordered values of the amount of incoming request per node during the simulation using the random walking algorithm.

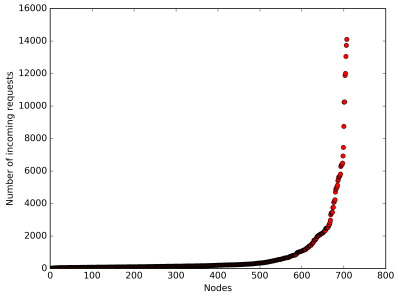


Figure 5.5: Ordered values of the amount of incoming request per node during the simulation using the biased walking algorithm.

```
    for algorithm in algorithms:
            create_distribution_graph(introduction_requests_per_node)

for algorithm in algorithms:
            create_distribution_graph(blocks_sent_per_node)
```

On the other hand, an egalitarian distribution of request might actually not be the best case scenario, since not every node has the same capacity. If we use the total amount of MBs recorded in the multichain for that node as a proxy for the bandwidth capacity of the node, we can normalize the this data to the capacity of the node.

```
for algorithm in algorithms:
    create_distribution_graph((blocks_sent/bandwidth)_per_node)
```

5.2.4 Parameter variation

Variation of alpha with smaller dataset.

5.2.5 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 numb - No churn - No NAT - No new blocks created - Exit nodes
- The overall feedback loop is not yet closed.

Chapter 6

Trust Edges

Aso did someting in dispersy to actually test stuff in a distributed way.
Here we used the requirement that an edge in the live overlay must also be an edge in the Multichain interaction graph. This is much stricter than in chapter 5
Walking is random, not foccused on high reputation peers.
Some graphs showing:
- Live edges over time - Load balance - Block accumulation

Chapter 7

Conclusions and Future Work

- There is a decent protocol in place for record creation - Well tested, and proven in the real world - Vunerabilities to noivel attacks unnknown, not battle hardned
- Some inversigation into record sharing and edge traversal, directed walking seems to have benefits, more research nessecary.
- Accounting Mechanism still needed, most are very costily.
More work is required to close the loop, and if it works reamins to be seen, but we've not seen any fundamental problems.

7.1 Conclusions

TODO CONCLUSIONS

7.2 Future Work

TODO FUTURE WORK

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

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

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

[12] R. Rivest, "The md5 message-digest algorithm," 1992.

[13] H. Dobbertin, "The status of md5 after a recent attack," *CryptoBytes*, vol. 2, no. 2, 1996.

[14] 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.

[15] X. Wang, Y. L. Yin, and H. Yu, "Collision search attacks on sha1," 2005.

[16] "Announcing the first sha1 collision." `https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html`, 2 2017.

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

[18] "Sqlite database engine." `https://www.sqlite.org/`.

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

[20] "Multichain walker simulation." `https://github.com/pimveldhuisen/multichain-walker-simulation`.

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

[22] 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.

[23] 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.

[24] J. Mazurek, "Evaluation of ranking similarity in ordinal ranking problems," *Acta academica karviniensia*, pp. 119–128, 2011.

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