Leveraging blockchains to enforce cooperation in distributed networks

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

18th January 2017

Author
Pim Veldhuisen

Title
Leveraging blockchains to enforce cooperation in distributed networks

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
18th January 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. One situation where each individuals 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 [3]. This is known as the tragedy of the commons.

Early Internet applications that attempted to stimulate cooperation between Internet users were also affected by these 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 [1]. A similar experiment performed on the eDonkey network in 2004 identified 68% of the users as free riders [5]. 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 extend 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 mechanism [2]. 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 [4]. 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 [9]. 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. Furthermore, the solution might not be resistant to users tampering with their score, which is self-reported.

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 [6]. 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 [7]. 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 [8]. 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 describes two important steps on the road to the creation an all-round blockchain based incentive system that improves network performance in the real world. The first step consists of reliable record creation. While reliable record creation is the foundation of the Multichain, it is not sufficient to create a tamper-proof reputation system. Nodes must be able to discover records through third party nodes, to verify the reputation of peers the could interact with. Discovery of records also enables peers to determine a reputation based on interactions in the wider network. Record discovery thus constitutes the second step.

Challenges and properties of the problems the two steps form are explored in chapter 2. This chapter also takes into consideration prior work in this field. Conceptual and implementation improvements upon the existing record creation protocol are described in chapter 3. Furthermore, this chapter lays out different options for the design of a discovery protocol. This thesis not only focuses on theoretical properties of the system, but also on experimental validation and real world performance. Chapter 4 then describes how both protocols are evaluated using various simulations and experiments, including deployment to Tribler users. Finally, chapter 5 summarizes the results of the research in a conclusion, and surveys future steps to the path of 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 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 2.1.

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 part of the system before everything is done. This allows improvement of the software and inclusion of new insights on the go. It also allows 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

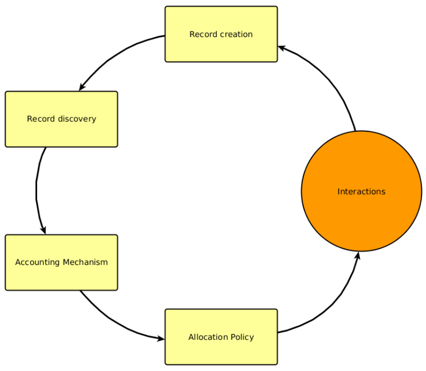

Figure 2.1: The different steps taken by the multichain system to enforce cooperation of peers.

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.

As mentioned in the previous chapter this thesis will focus on the first two steps. Previous work exists on the first step [8], 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 networks functioning.

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 condition 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 principle 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.

Chapter 3

Design

3.1 Record creation

The first step consists of creating records of interactions between peers that have occurred in the network. These interaction 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.

3.1.1 Asynchronicity

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 gridlock arises. 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 be non-blocking, and able to have multiple outstanding requests at the same time. The hashes used to chain blocks together however make this hard. 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

3.1.2 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 posses 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 posses 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 3.1.2

	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

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 same amount of bytes, the initiator will be decided based on the alphabetical ordering of the public keys.

3.1.3 Protocol

Data-structure - Datastructure -¿ fields used in database and wire packet

Message sequence - Message sequence

Asynchronicity - Asynchronous for performance: consequences for hashes

3.1.4 Implementation

Tribler community Tribler consists of different communities which share things among a subsection of peers. Some communities share content based on communal intrest, but others share functional data. The multichain system is implemented in Tribler as a community, where the particiapating peers share data about each others blockchains. The communnity is used as a wraper in which the peers communicated and send requests for new interactions and can check histrocal data. - As a Tribler Community

Dispersy Like all Tribler communities, the multichain community uses the Dispersy software to exchange messages among peers. This means that there exists a strong entaglement between Dispersy and the multichain. However, the multichain is not fundamentally dependent on Dispersy. It does not use the advanced features of Dispersy, instead relying on point to point messages. Furthermore, the database for the multichain block is seperated - Dispersy dependency / independency

SQLite - SQLite

3.1.5 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 are aquired, this 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 allong the peers in the network. Peers that are found lying should immedatly be refused any service for which the multichain is used, both because no reputation can be reliably assertained form the multichain, and to discourage lying in general.

False request - False requests - How does it work? - Solution: check validity

Deny requests - Don't sign downloads - How does it work? - Solution: don't interact with peers that don't sign

Hiding blocks

Branching - Branching: - How does it work? - Risk of discovery - Solution: make the reputation layer such that this is not worthwhile

Trick into branching - Trick into branching - How does it work? - Solution: Responder has a responsibility to check previous requests

Denial of service

Implementation-based attacks SQL injection Hash collisions

3.2 Record discovery

A commonly employed strategy to explore a graph is that of a random walk []. 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.

3.2.1 Unconditional Random walk vs reputation-directed walk

This enables a secondary feedback loop
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 explaination: 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.

3.2.2 Statefull vs stateless walk

A random walk usually consists of a single path, where each step the next nodes is contacted. In dispersy however, the exploration is strongly linked to the keeping alive of other nodes. To achieve this, multiple nodes are considered active, and each step consists of walking to a new node from one of the active nodes. While it is convenient to combine these two functions, walking in this manner might have slightly different properties.

3.2.3 Algorithms

The above options result in a total of four possible options. Each is described in more detail below:

Undirected statefull Most vannila walk, keep alive seperate from walking

Undirected stateless Currently implemented in dispersy

Reputation-directed statefull keep alive separate from walking

Reputation-directed stateless A personal favorite

Chapter 4

Evaluation

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.

4.1 Record creation

4.1.1 Block creation performance

Experiment 0: Block creation block creation performance graph

4.1.2 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 initial idea was to put this threshold at 1 MB, but due to the amount of blocks this would create, it was soon increased to 5 MB. A graph of a blockchain that this protocol could result in is shown in graph 4.1.
This graph was produced by running the code in a truly distributed way, while in a controlled environment. We can see that as a result of the policy long sections where to chains are intertwined exists. This represents sequential interactions between the same two peers, which is a result of the fact that an interaction often encompasses more than 5 MB of data transfer. This results in a situation where one logical interaction is covered by multiple multichain blocks.
Experiment 1: Signing policy Variant A: Every 5 MB: block graph
a shitload of blocks
Variant B: On tunnel close block graph
Fewer blocks = better

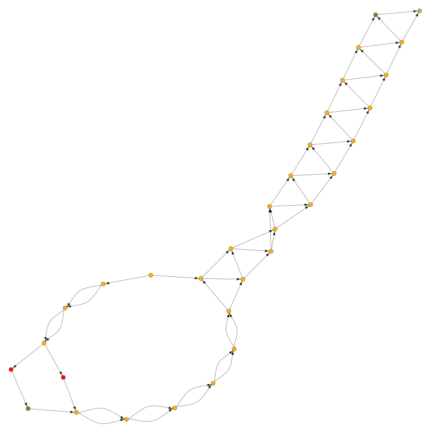

Figure 4.1: An excerpt from an early multichain block graph, highlighting internal inconsistencies.

4.1.3 Scalability

Experiment 2: Scalability Al lot of nodes block graph
Still works

4.1.4 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'.
- Optional updates - How do we enable useful measurements? - Forced update: No backwards compatibility

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. An excerpt from the block graph is shown in figure 4.2. A 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. The experiment helped to detect, identify and resolve the bug, and provided a criterium to test future graphs against. The chain itself was discarded after the bug was fixed. This was not a problems since this was only an internal release.

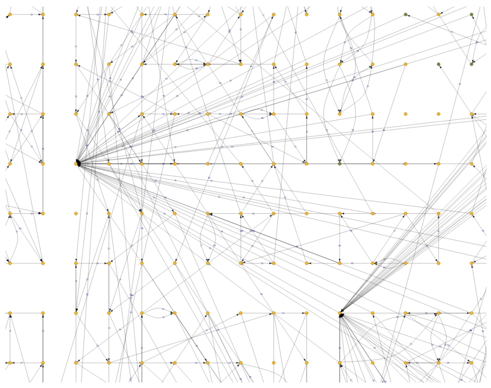

Figure 4.2: An excerpt from an early multichain block graph, highlighting internal inconsistencies.

The discovery of this bug lead to a series of checks that can be done on every database to verify its integretity. These checks can be found in table 4.1.4

Metric	Actual Value	Expected Value
Number of blocks	4823	-
Number of distinct identities	111	-
Reuse of key, sequence number pair		0
Reuse of previous hash		0
Average MB up in a block		
Average MB down in a block		

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

Release Experiment 3: Real world block graph
interaction graph

4.2 Record discovery

The performance of the different algorithms as described in 3.2 and their 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 4.1.4. 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.

4.2.1 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 4.3 for the random walk and figure 4.3 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 timestep, 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.

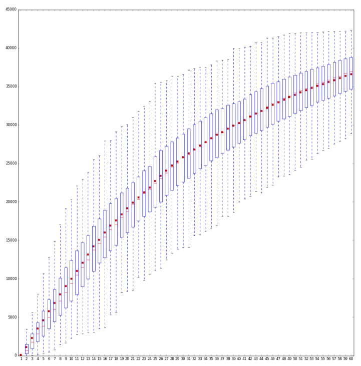

Figure 4.3: The collection process of blocks using the random walking algorithm.

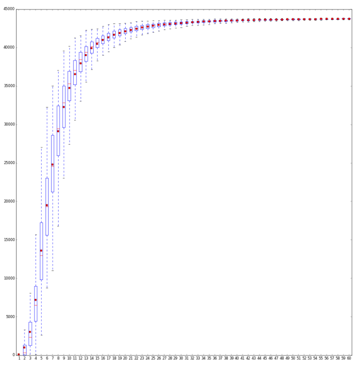

Figure 4.4: The collection process of blocks using the biased walking algorithm.

4.2.2 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 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.

4.2.3 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.

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

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

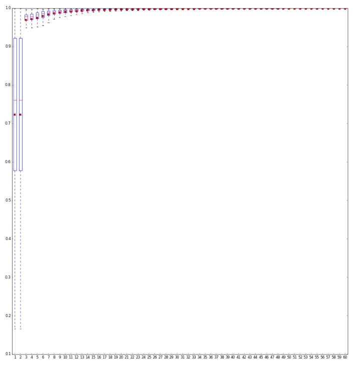

Figure 4.5: The convergence of the ranking similarity using the random walking algorithm.

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_nod
```

In the simulation churn is taken into account by turning some nodes of during certain times. NAT connectability is taken into account by making some nodes NAT restricted.

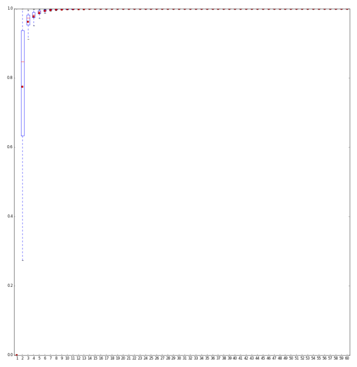

Figure 4.6: The convergence of the ranking similarity using the biased walking algorithm.

Chapter 5

Conclusions and Future Work

BIG TODO: design of churn resilience, ping on live edges
discuss policy: only incoming, outgoing, or mixed?
who to keep contact with?
of past 100+ connections, random 10 strategy
Then BigExitNodeProblem
if you introduce 1 peer for 10 minutes to all visotors, you DDoS that peer !!!! :-)
limited introduction or random peer selection as a solution.
Informativeness and replicas - people start sharing and spreading records of ”near” peers - describe policies, random, 1-hop, top-N, etc. -show that we can now do superior crawls
todo experiments storage requirements multichain records Processing time to create and insert sign incoming requests and store

5.1 Conclusions

TODO CONCLUSIONS

5.2 Future Work

TODO FUTURE WORK

Bibliography

[1] Eytan Adar and Bernardo A Huberman. Free riding on gnutella. *First monday*, 5(10), 2000.

[2] Bram Cohen. Incentives build robustness in bittorrent. In *Workshop on Economics of Peer-to-Peer systems*, volume 6, pages 68–72, 2003.

[3] Garrett Hardin. The tragedy of the commons. *science*, 162(3859):1243–1248, 1968.

[4] Seung Jun and Mustaque Ahamad. Incentives in bittorrent induce free riding. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Economics of Peer-to-peer Systems*, P2PECON '05, pages 116–121, New York, NY, USA, 2005. ACM.

[5] 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, pages 217–226, Berlin, Heidelberg, 2004. Springer-Verlag.

[6] 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*, pages 1–8, May 2009.

[7] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System.

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

[9] Mao Yang, Zheng Zhang, Xiaoming Li, and Yafei Dai. An empirical study of free-riding behavior in the maze p2p file-sharing system. In *Peer-to-Peer Systems IV*, pages 182–192. Springer, 2005.