Trustchain based descision policies

E. M. Bongers, BSc

Delft University of Technology

Trustchain based descision policies

Master's Thesis in Computer Science

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

E. M. Bongers, BSc

17th January 2019

Author
E. M. Bongers

Title
TODO TITLE

MSc presentation
2019-02-07

Graduation Committee

TODO GRADUATION COMMITTEE	Delft University of Technology
prof. ir. dr. D. H. J. Epema	Delft University of Technology
ir. dr. J. Pouwelse	Delft University of Technology

Abstract

TODO ABSTRACT
Select policies on Triblerchain that ensure fairness, forgiveness and reward positive contributions. With contributions to the half blocks design and validation of triblerchain. Introduce novel way of calculating Netflow

Preface

TODO MOTIVATION FOR RESEARCH TOPIC I have always cared for fairness. When there is no authority that enforces fairnes, only transparency and accountability can influence peers to behave fairly. In the case of Tribler this is compounded by the desire to remain annonymous. This seemingly contradictory problem attracts me.

TODO ACKNOWLEDGEMENTS Many thanks to Johan for being so patient with my slow progress.

E. M. Bongers, BSc

Delft, The Netherlands
17th January 2019

Contents

Preface v

1 Introduction 1

2 Background 3
- 2.1 Distributed Systems and Central Components . . . 3
- 2.2 BitTorrent . . . 3
- 2.3 Tribler . . . 4
- 2.4 Blockchain Technology . . . 4
- 2.5 MultiChain . . . 5
- 2.6 Literature . . . 7

3 Problem Description 9

4 TrustChain Engineering 11
- 4.1 Improving Halves . . . 11
- 4.2 Block Validation . . . 15
- 4.3 TriblerChain Specifics . . . 16
- 4.4 Gumby Experimental Framework Enhancements . . . 17
- 4.5 Proxy Peer Selection . . . 18

5 Experiments 21
- 5.1 Minimal Experiment . . . 21
- 5.2 Trustchain Based Policy . . . 21
- 5.3 Better Trustchain Based Policy . . . 22
- 5.4 Full Validation Experiment . . . 22

6 Conclusions and Future Work 23
- 6.1 Conclusions . . . 23
- 6.2 Future Work . . . 23

Chapter 1

Introduction

Distributed and in particular peer-to-peer systems are working examples a fundamental design principle of a democratic society, peers interacting on a basis of equality. These systems can annonimize journalists, dissidents, whistle blowers and uncensor the political opposition. In short peer to peer systems support the democratic process and are not just academical play things. Peer to peer systems actually help people with real world problems.

However many peer to peer systems are subject to the tragedy of the commons[1] where people consume resources based on their individual desires, as opposed to consuming resources based on what is sustainable ¡lit¿. However there are many examples ¡lit¿ of communities that successfully and sustainably maintain a shared resource. In the case of open-access resources it is invariably the peers which hold each other accountable, that form the basis for a sustainable use of the resource. This is not a traditional punishment stick nor a rewarding carrot, but more of a self-imposed discipline based on reputation. In the case of a distributed systems it has proven difficult to realize a functional reputation system.

Of particular interrest is the BitTorrent peer to peer protocol, currently the dominant peer to peer file sharing protocol on the internet. Its tit-for-tat mechanism ensures that, atleast for a single download-swarm, the system is sustainable. However it does not weed out habitual free-riders. Tribler, the TU Delft BitTorrent client, was recently enhanced with ~~DoubleEntry~~ ~~MultiChain~~ TrustChain, a blockchain inspired accounting mechanism. It is intended to augment the tit-for-tat mechanism, weed out the habitual free-riders and thus ensure a more sustainable peer to peer system. All without relying on centralized components like other solutions do.

TODO ORGANISATIONAL DESCRIPTION OF THESIS

[1]Which is in itself a tragic misrepresentation of history

Chapter 2

Background

2.1 Distributed Systems and Central Components

Distributed systems refer to a group of computers where each computer does a part of the work (computing/storage/communication/etc.) in service to the group in order to achieve some end goal. Many examples exist in the world with http (the world wide web) being a prime example and the domain name system (DNS) another. However these distributed systems rely on centralized components (such as central servers), and such centralized components are a weak point for a distributed system.

Centralized components have three aspects that make them undesirable in any distributed system. First of al central components are a single point of failure, so part of the service cannot be provided if they are unreachable or unresponsive. Secondly, they are inevitably controlled by a single entity, and any entity is influencable by private actors and governments. The worst case being manipulation of the service provided by the central component. This is generally impossible to detect and can severely affect the functioning of a distributed system. Thirdly, central components also have a hosting cost associated with them and the distributed system user community has to provide for this in some way.

Often the use of central components is a design compromise. It is often easy to have a (part of a) process execute in a controlled environment. This simplifies designs greatly. A good example of this is distributed gaming where the actual game simulation happenes in a controlled environment on a server. However distributed gaming without a central server has yet to be perfected.

So while central components can be used to great effect, they are to be eschewed in distributed systems that wish to operate even when under attack.

2.2 BitTorrent

BitTorrent is a peer to peer system that aims to share files between users in a fast and reliable way. It splits a file into many small chunks. The original uploader of the file and peers that have completed the download are known as seeders and provide chunks to peers that don't have all the chunck yet (leecher). All peers will exchange information about what chunks they have and want inorder to exchange chunks. This system has proven to be very reliable and fast, it can deal with large influxes of new peers, mass exoduses and has very low protocol overhead.

The BitTorrent protocol assumes a tit-for-tat reciprocity. This means that if a peer uploads a chunk which the counter party requests, the peer expects a chunk in kind. If the counter party does not reciprocate then the peer will, for a time, refuse further service (choking). There are however two effects that free riders can exploit to still get service while providing very little service back to peers.

First of all, seeders do not use a tit-for-tat mechanism since the leecher has no chunks that the seeder wants. Instead seeders simply spread the love and give equally to any peer. So a free-rider can simply wait for service from seeders in order to obtain all chunks. This could make the download take more time compared to a cooperating peer, but if there are enough seeders to saturate the free riders bandwidth, there will be no difference.

Alternatively, if the ratio of leechers to chunks is large the free-rider can simply try with many peers, and if it gets choked it simply moves to the next peer. For example in a download with 4,000 other leechers and 200 chunks the free rider has 20 peers to ask for each chunk. By the time the free rider has exhaused all peers, the first peers it tried will have unchoked the free rider again, and new peers will have started the download.

2.3 Tribler

Tribler is a BitTorrent client created by the Distributed Systems department of the TU Delft. For over a decade Tribler has enabled researchers to investigate a variety of peer-to-peer topics including distributed content indexing and searching, video streaming and anonymized donwloading. What makes Tribler unique as a research platform is the fact that Tribler is used by thousands of real world end users. New features get put to the test in the real world, not just academic benchmarks.

¡picture here¿

The long term goal of Tribler is to create a network of self-reinforcing trust on which to base all interactions with peers. A first step towards this goal has been the development of Multichain, a blockchain inspired accounting mechanism. When deployed in the real world this should weed out the free riders and potentially guard against sybil attacks. Future iterations can then focus on creating an iterative trust consensus model.

2.4 Blockchain Technology

he well known Bitcoin crypto currency is based on a blockchain. A structure of cryptographically signed blocks, being created in sequence. Each such block contains one or more transactions and most importantly a reference to the previous block, leading to a characteristic chain structure. Each block has an added proof-of-work value that gives the block special properties with regard to its hash value. Finding this proof-of-work value is very compute intensive and its primary function is to regulate the speed at which blocks are created. On average one block every 10 minutes. The secondary function of the proof-of-work calculation is to seed new bitcoins in the system with an associated real world cost. ¡pic of blockchain or bitcoin logo?¿

An important characteristic of the bitcoin blockchain is that a global state is needed to verify the last block in the chain. In the current system this global state is simply the history of all transactions, ever. These blocks are now several gigabytes, a huge drag on scalabillity. In addition to this issue, bitcoin has a fixed average block creation rate and a maximum number of transactions per block. So the total transaction throughput is limited. Currently about 7 transactions per second. Even though this limit can be streched, Bitcoin is very resource intensive by design and scales poorly ¡lit¿.

While numerous researches have not found weaknesses in the blockchain structure itself, there are several protocol and procedural flaws in bitcoin. The original design assumed that an attacker would have to control 51possible strategic manipulation of the mechanics of the bitcoin protocol ¡lit¿ this means that the largest hashpool operators are already in a position to attack bitcoin. Largely because bitcoin depends on a distribution of global state.

2.5 MultiChain

nspired by the ideas behind the bitcoin blockchain, the Tribler team has developed the MultiChain ¡lit¿. When applied to Tribler the MultiChain aims to keep a balance of all the work done for and by a peer over its whole existence. Using the MultiChain balance of peers in the neighborhood, a peer may compute a ¡PimRank¿ score for its connected peers. A PimRank can be used to rank service request from peers and improve on BitTorrent's tit-for-tat system. After each successful interaction peers will be more likely to interact again. Such repeated interactions leads to a local self-reinforcing trust between peers.

In MultiChain a request peer makes a claim of having some total balance after having exchanged some amount of bytes up and down with a responder peer. The requester signs this claim to irrevokably link the claim to the requesters identity. If the responder agrees that the numbers are correct, it will add its own balance to the claim and sign everything to complete the transaction. For each of the peers the claims also contain a hash link to a previous claim. The result of repeated claims and signatures is a large graph of interwoven chains.

This scheme has several advantages. First of all the use of a global state is avoided, each peer only needs its own chain and the claims from the responders. Secondly, this ensures that transactions can be handled directly between peers, and as such there is no explicit limit on the transaction throughput. Third, the wastefull proof-of-work computation is not needed since there is no global state to regulate.

For these advantages there is a tradeoff. MultiChain is a softer coin than bitcoin. MultiChain is designed around eventual detection of fraud whereas bitcoin stops fraud from entering the blockchain. The traditional blockchain peer can accept or reject a new transaction based on the global state it has. Since MultiChain peers have no such global state, it is not immediately clear if any peer is cheating or not. However the interwoven nature of the MultiChain ensures that fraud will eventually be detected. It is even possible to rollback all transactions of a fraudulent peer. This all makes the value of a MultiChain claim a bit less solid than having a bitcoin in a wallet.

The Tribler implementation of MultiChain counts bytes sent and received, and because it is impossible to send or receive negative numbers of bytes, the MultiChain counters only monotonically increase. As such it is not directly possible to transfer value between MultiChains. Because of this inabillity to transfer value directly and the softer character of MultiChain compared to bitcoin, it would be a misnomer to call MultiChain a currency. The Tribler MultiChain only has value when compared to others. It can be used to rank peers and their requests for service, and ultimately to weed out the habitual free-riders. As soon as an other "value" is linked to the Tribler MultiChain counters, such as democratic voting power or discount at a retailer, peers might make seemingly irrational transactions. This will cause a basic premise of MultiChain to collapse, namely that an attacker would need to provide as much bandwidth as it can consume. It might be rational to provide such bandwith, if the reward is not expressable in economic terms.

The first implementation of MultiChain as created by Norberhuis¡lit¿ relies on a synchronized interaction between two nodes. The requester sends a request that contains the identity of the last transaction of the requester (as in fig x). This identity is the hash of the completed last transaction. After the requester sends a signature request, it has to wait for the responder to answer since it depends on the response to create its next block. This blocking wait is undesirable since it can cause deadlock and limits the transaction rate, which is contrary to the design goals of MultiChain ¡lit¿.

Velduizen¡lit¿ revised the design of MultiChain to rely on the identity of half blocks, not the fully completed transaction. In this scheme, a block has two identities. First a hash of the request part of the block and second a hash of the full block as produced by the responder. By allowing transactions to be based on either of these identities, the requester does not have to wait for a signature request to complete before it can continue making requests. The downside of this revised MultiChain is that blocks have multiple identities. As depected in fig y, the structure becomes asymmetric. So even though the requester and responder execute almost exactly the same steps to produce a block, the resulting datastructure is asymmetric. This datastructure requires updates and is not cachable. Lastly the signature response cannot exist separated from the signature request, since only the signature request contains the delta for the counters. The major contribution of Veldhuizen was to default to half signed blocks in order to eliminate the chain locking behaviour of the Norberhuis version. However this insight was only partly utilized and resulted in an asymmetric and mutating datastructure.

2.6 Literature

- ”KARMA : A Secure Economic Framework for Peer-to-Peer Resource Sharing” https://www.cs.cornell.edu/people/egs/papers/karma.pdf - Rep on the block? - Norberhuis - Pim V - Pim O - Delayed payment processing

Chapter 3

Problem Description

When downloading in a swarm the tit-for-tat mechanism ensures that every peer has to participate. This simplistic method has proven effective in practical situations, however there are still opportunities for free-riding in individual swarms. The only practical real world solution so far is sharing ratio enforcement through centralized tracking of a peer's sharing ratio. Many private bittorrent communities do exactly that and keep a record of each peer's sharing ratio over all downloads. This is a high a barrier of entry for the average user, and requires the private community to operate a central system. There is no general solution that is decentralized, unbounded by tit-for-tat's limitations, promotes sustained cooperation and has a low barrier of entry (just works/out of the box).

Similar to BT's swarm free riders, there are also free-riders in the annonimization layer of Tribler. To annonymize P2P-traffic the Tribler network needs peers that relay traffic, but there is currently no strong incentive for peers to provide this service apart from the intrinsic reciprocity. The Tor network has a similar situation and its throughput is limited because it lacks an incentive for peers to proxy connections. ¡lit¿ All of the proposed solutions for Tor ¡lit here¿ include a centralized component. So as with bittorrent free-riding the same need for a solution is encountered in annonimization free-riding.

The recently introduced MultiChain system can in theory be used to track a peer's sharing ratio over it's lifetime, so this should be able to weed out the free-riders. MultiChain creates a limited transitive trust between peers that have never interacted, they trust one another because of, and only as far as, they trust the peers that connect them. In the case of Tribler this means that peers have a limited faith that interaction with an unknown peer will be successful. That is assuming there is at least one path along the MultiChain connecting the peers. The MultiChain induces an ordering on the known peers, and thus a basis for deciding who to provide service to. There are several elephants in the room when translating this high-level concept to an actual implementation.

1. Foremost, MultiChain's state of engineering is not ready to be deployed. For example, the current implementation will sign any request, does not validate

messages and does not request records when they are needed. Ask Ratio, (software engineering metrics and practices)

2. PimRank is not yet integrated in MultiChain and Tribler, nor any other metric. Sharing ratio (tracking)

3. How to make service decisions based on PimRank while avoiding “social collapse”? How forgiving to be towards new nodes? Was that just a packet drop or adversarial manipulation? Risk Management, Parity for contributions, seeding and relaying

Chapter 4

TrustChain Engineering

Both the original TrustChain by Norberhuis and the revised TrustChain by Veldhuizen where Proof-of-Concept implementations and not fit for real world deployment. A considerable design and software engineering effort was made to prepare for the deployment of TrustChain and to gather data from real usage. This chapter starts with the conceptual redesign of the TrustChain to fully embrace the the half blocks in section 4.1, and works towards the description of a specific implementation known as TriblerChain in section 4.3. Validation of blocks and transaction contents is discussed in section 4.2. Finally section 4.5 is dedicated to the proxy hop selection problem faced by Tribler.

4.1 Improving Halves

The TrustChain as revised by Veldhuizen ¡lit¿ allows the use of just the signature *request* as previous block pointer, as opposed to the Norberhuis design where only the signed response is a valid previous block pointer. Fundamentally this changed the TrustChain from a synchronized mutual agreement system to a desynchronized compatible claim system. However Veldhuizen only used the change to remove the need for a blocking wait before starting another transaction. That was a missed opportunity, since fully embracing the fundamental change to a mutual claim system can bring additional benefits.

The detailed design of this revision of TrustChain starts with the half blocks. Conceptually each half block represents a signed claim by a peer that a transaction has occured. However, if transactions occur between multiple peers, each peer that is involved in the transaction signs such a half block. If all the half blocks together present a consistent whole the blocks are compatible and the transaction is completed. If there are blocks missing or if there are inconsistencies, the transaction has not completed. Note however that due to the desynchronized nature of this scheme transactions are not forced to complete within a set timeframe. Each half block has to contain the following information:

1. The identification of the peer creating the half block.

2. The sequence number of the half block on the TrustChain of the signing peer.

3. Previous half block hash, this will contain the hash of the previous half block on the chain of the signing peer.

4. A link to connect the chains of the transaction participants.

5. A cryptographic signature over all the data in the half block.

6. Implementation dependent data for the transaction that is claimed in this half block.

The identification of the peer creating the half block is used for two reasons. First it is needed as part of the primary key of the block. Secondly, it is used to validate the cryptographic signature. In the Norberhuis implementation of TrustChain, the identification used was the sha-1 hash of the the peer's public key. This is efficient since few bytes are need compared to the full public key. However when crawling the TrustChain of other peers there will be signatures by unknown peers. Because of the hashing, it is impossible to verify these blocks, since the required public key cannot be recovered from the hashed public key. Therefore, to make blocks self verifiable, the full public key of a peer must be included as identification of that peer.

The sequence number of the half block also serves multiple purposes. First it is the second part of the primary key of a half block. This complete identification of a block is used in compatible claim half blocks to refer back to a single and specific transaction initiation half block. Secondly, the sequence number guards against a fork in the TrustChain of a peer, since this would require using a sequence number in multiple half blocks. Third, the sequence number ensures that there are no obvious gaps in the TrustChain of a peer.

To form a consistent transaction the half blocks from multiple TrustChains must relate to each other in some way. Any peer that is part of the transaction must be identified. So atleast the public keys of all participants must be included. By adding the chain sequence numbers the primary key for each half block is known so the links between blocks can exactly identify a half block. However the initiating peer does not know the sequence numbers that the transaction participant peers will use to create their half block, such knowledge could be outdated by the time it is communicated. So the initiator cannot exactly link to half blocks on the chain of the transaction participants. The initiator is allowed to leave the linked sequence numbers blank. Such a blank part of the link is infact the only way to identify half blocks that initiated a transaction.

Since there already is a sequence number it is not immediately obvious why a previous half block pointer would be needed. However consider a peer that is fraudulent and signs the same sequence number for multiple transactions. Without a previous half block hash these branches would merge when the next sequence number is used. There is nothing forcing the branches to stay separated. By including the hash of the previous half block these branches are kept separated, requiring the fraudulent peer to perpetuate the fork and thus increasing risk of detection.

Observant readers might have noted that if the hash of a block suffices to identify it, then the half block primary key tuple (peer public key, sequence number) could also be replaced by simply the half block hash. Indeed this is true for the link connecting two half blocks to a full block, and all chain integrity checks would still be possible However completely removing the sequence number introduces the possibility of infinite chain lengths. Consider the design without the sequence number and a peer initiating a transaction. The other transaction participants would then try to obtain a copy of the initiator's TrustChain, starting from the genesis block. The initiating peer can generate half blocks as desired without disclosing how many blocks remain until the initiated transaction is reached, leading to a Denial-of-Service attack. Given enough computing power, the initiating peer might even hope to create a hash collision eventually to actually legitimize the half block that initiated the transaction.

TODO: it is also possible to add a linked-last-seen-hash field. Such that each party is aware of the knowledge of the others at the time each half block was created/signed. Perhaps that makes iterative consensus easier?

The removal of the temporal constraints of TrustChain transactions as compared to the Norberhuis version is not without sacrifice. A major downside is that it is possible for transactions to complete very long after they where initiated. As such functions, like aggregate sums and products, over all the transactions in a peers TrustChain are not always reliable. For example, a peer might have a positive balance of some value, but if a large transaction from long ago suddenly completes, its new balance might be very negative.

In essence this is the problem that the transaction initiator cannot reliably have up to date knowledge of the state of another TrustChain. In addition to implementation dependent bounds on transaction validity this problem can be mitigated by adopting a double agreement scheme. Assume an initiator starts a transaction and all other transaction participants then sign their half block. After exchanging all the half blocks every transaction participant can verify that nothing unexpected happened on the TrustChains of the other transaction participants up to the point of the transaction. Lastly all parties sign a secondary half block that signifies they where satisfied during the verification phase. If all secondary half blocks are signed and exchanged, the transaction is complete and everybody is in agreement over what was signed, and what the state was of every TrustChain upto the point where the transaction happened.

The last hurdle from a theoretical standpoint is how to interpret transactions that do not present all expected half blocks or transactions that do not contain consistent implementation dependent transaction data. For the case of missing half blocks, if there is no implementation dependent reason why transactions cannot complete in the future, these transactions are to be considered pending. It is possible that the missing blocks exist but where not (yet) communicated, or that a peer is suppressing selected half blocks, etc. Implementations might also chose to include rejections in the transaction data of a half block. From a TrustChain point of view these transactions are complete since all half blocks are known, but the rejection marks the transaction as a whole as failed.

TrustChain has been refactored to fully embrace the half blocks design. This brings the following benefits as opposed to the Veldhuizen implementation of TrustChain:

- Half blocks are self verifiable.

- Half blocks are by design immutable. This makes it possible to distribute, cache and store half blocks without restriction. A desirable property in distributed systems. It also adds to the robustness, since the signing parties don't strictly need direct contact.

- Simplification of the messaging protocol. The messaging protocol goes from 6 message types to only 2 types of messages. One message type to request blocks from another node, and one message type to transport half blocks. If a node recieves a half block, it can deduce if the half block is signed, if the half block needs to be stored, or even if it should sign a compatible half block. There is full local autonomy

- The previously used request/response mechanic from Dispersy can be discarded. There is no longer a need to keep track of outstanding requests and which requests have had responses, this makes the messaging protocol stateless.

- Reduced overhead on messaging, since the half blocks are signed and tamper proof, the Dispersy messaging mechanic no longer needs to verify signatures on half block messages.

- The structure of half blocks is identical for all transaction participants. Because of this, much code can be shared between the requester and responder code paths.

With all these concepts applied to the TrustChain code, total a reduction of 40

There are still several strategies that an adverserial peer can employ:

- Double sign a sequence number. This is akin to double spending in a classical blockchain. In this case a dishonest peer will reuse a sequence number for which it already has a half block on its chain. This creates a fork in in the TrustChain of the dishonest peer, and because of the previous half block hash the dishonest peer can never merge the forked chain. This strategy is not trivial to detect, however eventually a peer will obtain the two fraudulent half blocks, and expose the fraud to the world.

- Double link to an initiating block. In this case an honest peer creates a half block. The dishonest peer then signs multiple half blocks that link back to the same transaction initiating half block. This is orthogonal to double signing a sequence number where a horizontal fork was created by the transaction initiator, this case is a vertical fork created by the transaction participant(s). This strategy is trivial to detect, since an examination of the full TrustChain of a peer will reveal multiple linked blocks to one original transaction.
- Strategic supression of half blocks. When examining the TrustChain of a peer, that peer might decide not to share the linked half block of an unfavourable transaction. In the case of a half block that links back to a transaction initiating half block, it is obvious that something is with held. However for blocks that initiate a transaction the linked block might genuinely not exist.
- sybils. Sybil defence is implementation dependent.
- block creation DoS

Both the double sign of a sequence number and the strategic suppression of half blocks are primarily possible because peers report about their own chain and associated blocks. If the storage and retrieval of a peers TrustChain half blocks where to be separated from said peer this could aid in the detection of fraud or manipulation. Especially if such a separated storage, like a DHT, moves blocks about or gossips about blocks. That would increase the chance that two blocks that expose fraud encounter eachother. Future work on TrustChain should investigate such a backup storage or local gossip strategy as a way to mittigate the known attacks.

4.2 Block Validation

One substantial element that was missing from the previous TrustChain versions was a validation check on blocks. Previous versions would just sign any transaction. This is obviously not acceptable for a real world deployment. A validator was implemented that checks the basic TrustChain properties. In addition it executes callbacks to validate the implementation dependent transaction data. Only if blocks pass validation are they processed further and stored in the database.

As part of the validation process the hash of the previous block is also computed and checked against the previous block hash in the block under validation. It can happen that the local block database does not hold the previous block of the involved peer, in such cases the validator passes the validation but reports that the validation was only partial. It is up to the application to decide what to do in that case. Note however that it is not possible to hide fraudulent blocks by sneaking them through as a partial validation. When an eventual full check of the TrustChain of the involved peer occurs, the validation will fail for blocks adjacent to a fraudulent block. This means that the block database can never contain a fully connected fraudulent block. This renders the TrustChain of a fraudulent peer unprocessable, depriving that peer from further service.

If a peer recieves a transaction initiation half block and would like to accept the transaction, then a partial validation result is not good enough for this half block. TrustChain will request that the initiating peer send the block immediately preceding the transaction initiation half block. If a partial validation where allowed for transaction initiating blocks then a fraudulent peer could trick another peer into a transaction on an unconnected block. While this is not fraud on the part of the honest peer, it does indicate something strange is happening and thus should be avoided.

Because it is not possible to recover a TrustChains after it has issued a block that fails validation, it is very important to check if newly created and signed half blocks are valid before communicating them to the world. Thus the validator is also used to validate a peers own new half blocks before they are sent out.

While care has been taken to methodically think about all the information in a half block and how it could be valid or incorrect, it is impossible to prove that the validator is perfect.

TODO: should this go before or after the TriblerChain section? The TriblerChain section does deal with concepts of valid and invalid, so perhaps it would be best to introduce the concept of valid first?

4.3 TriblerChain Specifics

While the previous section deals with the TrustChain as an abstract concept, this section presents TriblerChain. A specific implementation of TrustChain as used in the Tribler bittorrent client.

The goal of the TriblerChain is to enable detection of free riders over multiple bittorrent swarms. To this end, whenever two Tribler peers exchange data the TriblerChain counts bytes uploaded and downloaded. The TriblerChain uses the TrustChain design as outlined in the previous section and adds four counters for each transaction.

- Up & Down: The total data bytes exchanged between two peers.
- Total Up & Total Down: The sum of the traffic counters for the peer creating the half block over all transactions in its TriblerChain.

In addition to the half block linking rules that the TrustChain imposes, two TriblerChain half blocks only contain a consistent transaction if the up counter of each half block is equal to the down counter of the other half block. Additionally, the Totals must add up with respect to the previous TriblerChain half block.

TODO: tell about keeping up/down session state state?

The previous versions of TrustChain by Norberhuis and Veldhuizen where also versions of TriblerChain. However they did not include any decisions based on the transaction history. Their experiments deal with peers helping eachother uniformly. Once decisions are based on the counters in the TriblerChain this effects what peers will recieve service and uniformity will be lost. Every peer induces its own ordering on all other peers. Those with a higher ranking will be preferred for further cooperation.

Since the feedback is positive it will reinforce existing relations, creating an almost static network. The exit nodes will be the top of the ecosystem since they move all traffic in and out of the Tribler network. Everybody must go through them. Peers that have a high TrustChain rating, in the oppinion of the exit, will become key figures in relaying access to the exits. Up to the point where the exits are exclusively accessible to this clique. This clique must pass along the majority of this bandwidth to maintain a high TrustChain rating. However if they decide to use the exits exclusively for a while, or even collude to do so, they can and will block/starve the rest of the network. A large fan out (or high degree of each peer in the graph) can mitigate this effect somewhat. Alternatively, exits could decide to ignore TrustChain status alltogether. Another effect is that new peers will be the bottom feeders of the ecosystem. If they select to use multiple hops annonimity they might not even make it to an exit, forcing them to use more hops. It is uncertain if they can climb out of this position. The self reinforcing TrustChain network can only work if there is an oversupply of bandwidth.

Over many cycles of cooperation this feedback mechanism thus leads to the stratification of the Tribler community, if not the outright formation of local groups/cliques that shun outsiders. It is unclear if such a feedback mechanism would remain stable, reinforce good behaviour and be accessible to newcomers. This is the main goal of investigation in chapter **??**.

TODO: relay incentivization, dont pay a lineair sum

4.4 Gumby Experimental Framework Enhancements

For running experiments the Gumby Experimentation Framework is used. It already has support for Tribler and its underlying middleware, and a scenario scripting facility.

However support for Tribler has its limitations. Namely that it is impossible to write a scenario that controls both the TriblerChain and annonymous tunnels simultaniously. This deficiency was addressed but required a far reaching refactoring of gumby. Combined with fixing technical debt accumulated over the years, it can now be said that the Gumby code has been gold plated[1].

During the experimentation phase it was discovered that Triblers annonymization layer would not work reliably during experiments. This was traced back to the use of the real world DHT to connect hidden seeders and downloaders. While this works for a single run, the next run would hit stale data in the real world DHT and be unable to find the right peers in subsequent runs. Eventually a gumby module

[1] See `https://blog.codinghorror.com/gold-plating/`

was created that can isolate DHT nodes and thus produce a DHT that is limited to just the experiment peers.

4.5 Proxy Peer Selection

The last piece of engineering is to enable policies on annonymization circuits based on information from TriblerChain. An important question in this is who decides on what peers to use to build a circuit, either the tunnel originator, or the peer at the end of the tunnel.

Originally the tunnel originator gets a list of suggestions for next peers when a tunnel section is established. In Tribler's implementation the originator gets to name the next peer by public key and socket address in the extend message. Tribler selects one of the suggested next peers, except when a specific exit (or rendevous point) is required.

This is a problem for TriblerChain because specifying a target peer breaks the assumption of local autonomy. Imagine the following scenario:

- Peer A starts a tunnel to a peer with a slightly better TriblerChain rating. - Peer A continues extending the tunnel up the social ladder. Upto peer T, the Top node. - Peer A extends the tunnel from T to peer A' a sybil of peer A with a very low TriblerChain rating. Normally T would not interact with A' because of its low rating, but due to the mechanics of tunnel building it is forced to do so.

If the originator has the final say in what peers to use this could lead to awkward situations at the end of a tunnel. This is something that needs to be addressed since this is also how rendevous points work.

If each peer decides itself what peer to use for tunnel extension, it becomes impossible to steer the circuit towards an exit or rendevous point. And if the tunnel happens to extend into a malicious peer, it can certainly decide the next peer to be a sybil, ad infinitum. A black hole of malicious sybils from which there is no escape. This is a fundamental no go, the tunnel end peer may never decide the next peer. The tunnel end peer should not even be able to influence the selection of the next peer.

A solution could be to allow tunnel extension requests to be rejected. This means that an extending circuit might need to backtrack and select another peer for extension. Even giving up on building a circuit all together if all peers are rejected. However notice that "rejecting everything except a few peers" is actually a way the last peer can influence the next peer selection. As discussed previously this is not allowable, so any solution that allows rejections on tunnel extensions is susceptible to a sybil black hole.

TODO: If the origin is to generate/select all the hops than it is likely that the tunnel will use nodes close to the origin, since those are known. Drawn in a graph the tunnel will sort of wrap around the originator and only jump to the exit on the last node. This doesn't put much "distance" between the originator and the exit.

TODO: if the network and ecosystem that forms is rather static, won't it be easier to detect likely paths for circuits, and thus leak data / worsen security?

Chapter 5

Experiments

The goal is to experimentally verify that Trustchain based policies reject free riders over multiple downloads and do not introduce problems.

5.1 Minimal Experiment

The minimal experiment is a setup to serve as a baseline for further experiments.

The setup is 4 peers. One is an naked seeder. Another two are exits, and the last peer is an annonymous downloader. The download completion is tracked on the annonymous downloader. The results for this experiment are plotted in fig 5.1.

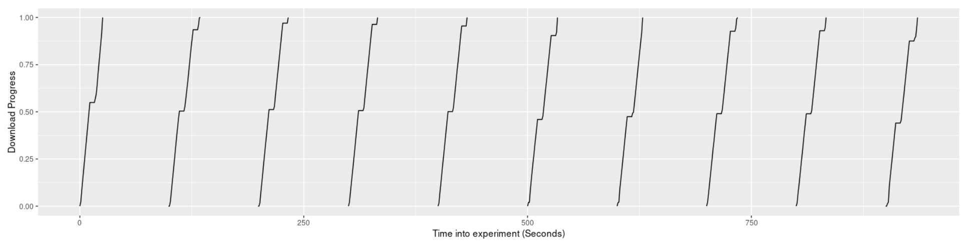

Figure 5.1: Minimal experiment results

TODO: figure of triblerchain values?

5.2 Trustchain Based Policy

Similar to the minimal experiment but with the default TrustChain based policy enabled.

figure(s)

Discuss result

Come up with ”better” alternative

5.3 Better Trustchain Based Policy

Similar to the minimal experiment but with a Better TrustChain based policy enabled.

figure(s)

Discuss result. Bash default policy.

5.4 Full Validation Experiment

The Full Validation Experiment is a setup to verify that the better Trustchain based policy will work in a more realistic real world scenario.

Start 20 peers in 4 groups: 9 peers will download and seed upto ratio 1.9, 4 free ride low (ratio 0.8), 4 free ride medium (ratio 0.5) and 3 free ride heavy (ratio 0.0). Sequentially perform 15 downloads of 20MB. Plot download completion over time for each peer, colour by peer grouping. If TrustChain works as advertized, we should see the download completion of the 3 free rider groups converging towards their respective sharing ratios or lower. Importantly we should see some of the 15 downloads no longer completing. At the same time the other peers should see a modest increase in throughput. If TrustChain is disabled (or does not work as advertized) we should see all downloads completing, and all peers getting roughly equal throughput.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

TODO CONCLUSIONS

6.2 Future Work

TODO FUTURE WORK