Title page

Voorwoord

Stuff.

Inhoud

<auto generated>

Introduction

Distributed and in particular peer-to-peer systems are working examples a fundamental design principle of the internet, peers interacting on a basis of equality. Not only are these systems usefull to overcome virtual restrictions, but due to annonimization they can be of great practical help to journalists, dissidents, whistle blowers and uncensoring political opposition. Peer to peer systems are not only academically relevant, they actually help people with real world problems.

However many peer to peer systems are subject to the tragedy of the commons where people consume resources based on their individual need, as opposed to consuming resources based on what is sustainable <lit>. Thankfully the human nature is not that depressing and there are many examples 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.

Of particular interrest is the BitTorrent peer to peer protocol, currently the dominant file sharing protocol. It's 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 multichain, a block chain inspired accounting mechanism. It is intended to supllant 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. <lit>

<structure>?

Background

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.

BitTorrent

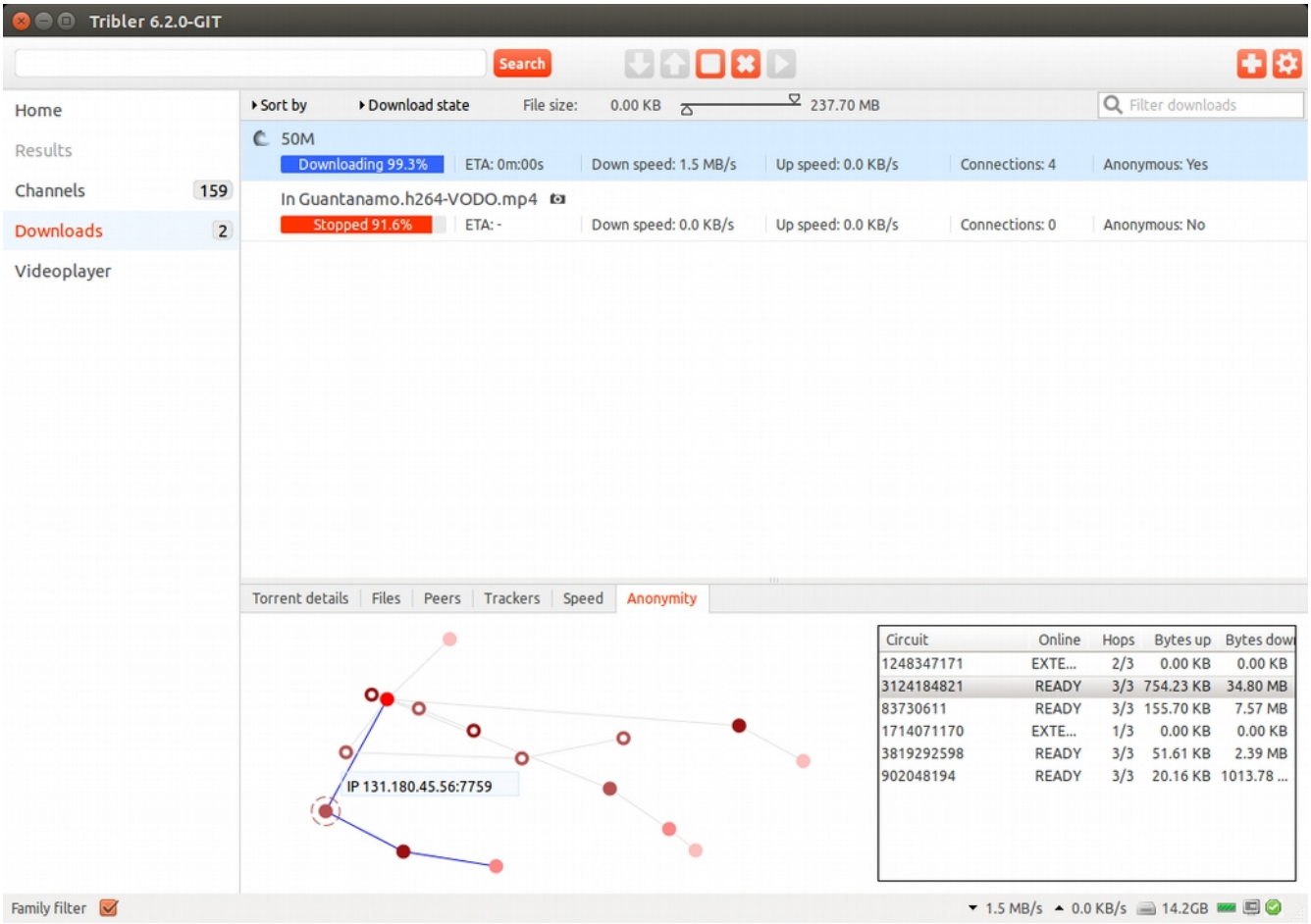
BitTorrent is a peer to peer system that aims to share files between users in a fast and reliable way. It splits a file into many small chunks. The original uploader of the file and peers that have completed the download are known as seeders and provide chunks to peers that don't have all the chunck 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.

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.

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.

Blockchain Technology

The well known Bitcoin crypto currency is based on a blockchain. A structure of cryptographically signed blocks, being created in sequence, at a somewhat predictable rate. Each such block contains one or more transactions and most importantly a reference to the previous block, leading to a characteristic chain structure.

An important characteristic of the bitcoin blockchain is that currently a full history of all transactions is needed to verify the last block in the chain. This complete global state is a huge drag on scalabillity. In addition to this

issue, bitcoin has a fixed average block creation rate and a maximum to the number of transactions per block. So the total transaction throughput is limited.

MultiChain

The multichain as used in Tribler is inspired by classical blockchains, but has several different characteristics and goals.

The wastefull CPU intensive mining is not needed. Instead both parties agree on some transaction by both placing a signature.

It does not require a peer to have full global state, only its own chain and the records from other chains that proove the double signature.

There are no limits on transaction throughput. Since there are no global operations involved, the network as a whole is not limited in the number of concurrent transactions.

Eventual detection of double spend and correction. The traditional blockchain peer can accept or reject a new transaction based on the global state it has. Since multichain purposfully lacks this global state, it is not immediately clear if a counter peer is not cheating. However the intertwined nature of the multichain ensures that fraud will eventually be detected. It is even possible to rollback all transactions of a fraudulent node.

Problem Description

Eco-system for sustained cooperation, which is beyond tit-for-tat, <voor inspiratie zie http://resolver.tudelft.nl/uuid:d2a72d3f-d28b-4d9d-998a-b030c2f28aed, ch 2>

Multichain deploy & crawl, prototype code → production code (software engineering metrics and practices)

Ask Ratio

Parity for contributions, seeding & relaying

Sharing ratio (tracking, enforcement)

When downloading in a BitTorrent (BT) 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 a swarm. The only practical real world solution so far is the centralized tracking of a peer sharing ratio that happens in private communities. This is too high a barrier of entry for the average user. So there is a need for a mechanic that promotes sustained cooperation that is not bound by tit-for-tat's limitations.

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.

Looking at the bigger picture, it is not possible, in a fully decentralized setting, to track a peer's participation across multiple BT swarms and the annonimization layer. Therefor habitual free-riders are not shunned becase they are not identified. The recently introduced MultiChain system can be used to track a peer's sharing ratio over it's lifetime, so this can potentially be used to weed out the free-riders. With MultiChain, peers cryptographically sign transaction blocks, which in the case of Tribler contain lifetime sharing ratios. After a successful interaction and MultiChain block creation peers will be more likely to interact again. Such repeated interactions leads to a local self-reinforcing trust between peers.

The MultiChain also creates a limited transitive trust between peers that have never interacted, they trust one another because 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 nodes. The MultiChain induces an ordering on the known peers, and thus a basis for deciding who to provide service to.

To annonymize P2P-traffic the Tribler network needs nodes that relay traffic, but there is currently no strong motivation for nodes to provide this service apart from the intrinsic reciprocity. The Tor network has a similar situation and its throughput is limited because it lacks a motivation for proxy nodes. <lit> To remedy this situation, the recently developed MultiChain should help communities to detect and shun non-cooperating nodes. In order to incentivize proxy nodes, multiple schemes can be considered based on the MultiChain double

signature primitive. Moreover the application of MultiChain to a real life problem also provides insights into further refinements of the MultiChain model itself.

Background & Literature

- "KARMA : A Secure Economic Framework for Peer-to-Peer Resource Sharing"

https://www.cs.cornell.edu/people/egs/papers/karma.pdf

- Papers about Tor incentivizing

- Norberhuis

- Pim V

- Pim O

- Delayed payment processing

Terminology

^ if I need this , it's wrong

A proxy node relays traffic.

Up and down a tunnel, the path from one end of the annonymization process to the other end, with one to many proxy nodes in between.

To “pay” with multichain. A mutation of up & down counters, the inverse of which is applied to the peer's multichain.

Architecture and Protocol Design

True Halves

Figure 1: Norberhuis Multichain

The implementation of multichain as created by Norberhuis[cite] 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 Figure 1). The identity is the hash of the response to the last transaction. After the requester sends a signed 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 limits the transaction rate, which is contrary to the stated design goal of multichain to be scalable in nodes and transaction volume.

Velduizen[cite] revised the design of multichain to rely on half blocks as a default. 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 block request to complete before it can continue making requests. The downside of this revised multichain is that blocks have multiple identities. As depected in Figure 2, 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. Moreover, when the responder has signed, every node with knowledge of the requesters multichain will require an update to complete the block that is now signed. This datastructure is not immutable or easly cachable in a distributed system. The major contribution of Veldhuizen was to default to halfsigned blocks in order to eliminate the chain locking behaviour of the previous version. However this insight was only partly utilized and resulted in an asymmetric and mutating datastructure.

Figure 2: Veldhuizen Multichain is asymmetric

Code cleanup through ego-centric viewpoint.

Figure 3: Symmetric Multichain

When using half-signed blocks as default a node is infact saying “I claim that ...”. The other party can either affirm this by adding its signature, or it can reject the claim by ignoring it. The key difference is that it is no longer a “Can you sign this?” style question with request-response mechanic. It is a statement that once made allows a node to immediately continue making other claims. Notice that the responder makes a claim too. Thus the design of multichain can be brought closer to true

Multichain can be revised to eliminate these defects by making the responder use a half-block claim to “sign” a request. This makes the datastructure symmetric, reduces messaging to only a single type of packet and since blocks are immutable they can be cached and distributed without limit.

- All attacks described by Norberhuis reduce to detecting double signed sequence numbers from the requester, or double countersigns of the same request block.

HOBBY HORSE: How hard is it to "hide" a fraudulent MultiChain block?

Say that node M has comitted fraud, it split its chain and thus it must have double signed a sequence number M.seq'. Because of the previous hash pointers in multichain blocks, this split is propagated to all further sequence numbers. This means ongoing fraud if M wishes to advance both branches of the split multichain, since it would need to double sign any sequence number after M.seq', once for each branch. So M will want to make the branch as short as possible, spread one of the blocks far and wide as its “true” chain, and limit the spread of the branched block that would expose its fraud.

A simple detection scheme could be to obtain the last x multichain blocks of any node that gets introduced and check them for fraud against what is known locally.

Assume M wishes to conduct further business and has to select a node (A) to interact with. Node A will obtain a copy of the last x multichain blocks that M has signed. This ofcouse includes M's public copy of the fraude block and not the block it wishes to keep private. Suppose M made the split in it's chain such that only node B has obtained the private fraud block of M. Then M must select A in such a way as to minimize the chance that A has interacted with B in the time from M's fraud upto x transactions later on the multichain of B. If A did interactwith B within this timeframe, A will have the block that M wishes to keep private as part of obtaining x transactions from B. To completely hide the fraud from the simple scheme, M will have to successfully have x of these transactions. After those, even the x nodes that interacted with B immediately after the fraud will no longer obtain the public fraud block from M.

Thus in this scheme there are 2 variables affecting the chance of discovering fraud.

- The selection probability s, the chance that a node gets selected for interaction. Worst case this is uniform over the whole community and thus depends on the number of nodes in the network. In the best case, some locality or random walking will give nodes closer to M a higher probability to be selected. In such a scheme nodes further from M will rank M as an almost new node and are thus not likely to provide M with service.

- The crawlback distance x, increasing this value leads to nodes keeping an almost complete global state, while lower values make it far easier to hide fraud. The higher value will also force M to have more successfully interactions after the fraud, before the fraud is really hidden.

<do math modeling to make it into a formula/predictions>

A problem with this simple scheme is that M can have x sybils sign some fake interaction on it's public multichain to hide the fraud on its public chain. Next interactions will then not go back further than x and are thus not in a position to detect fraud. Thus a hardcoded x will not work.

<further plans to investigate, x with a geometric distribution starting at 200. Or keep the full multichain of a few select nodes? Keeping them moving is another option, see section “moving blocks”>

Local Flooding

Would it be usefull to broadcast/flood a sucessfull transaction?

A limited dispersal of each sucessfull transaction would make it harder if not very hard for a node to claim what their "head" of the chain is. This would be another way of mitigating the attacks currently possible on multichain. An attacker would have to wait out the expiry/TTL on their last announced transaction before they could branch again, or do any other interaction. This would slow an attack down to the point of it being infeasable. If there are hetrogenous expiry/ttl values, the wait time between attack steps becomes even higher.

Money or Rank

An important aspect of multichain is its semantic interpretation. Is it money (or credits) that can be used to "buy" things for a price? Or is it more usefull because it induces a total ordering over the community (i.e. nodes can rank each other for service)? It is easy to slip into the money category since it is familiar. When viewed as a currency, multichain expresses a physically limited resource, namely bandwidth used. This is also a problem since total network bandwith is ever increasing and real world bandwith growth is probably super linear. This leads to an inflating economy and devaluation of any built up credit. Thus I will not view multichain as a credit system, but as a ranking system.

Payout & proxy incentivation

A strong incentive to relay traffic means that a node can *prove* to any member of the community that it has relayed. MultiChain is a cryptographic counting system that can be used to provide such proof.

A naïve scheme could be to compensate relays for their bandwith usage, both up and down where the downloader at the end of the proxy chain has to provide this compensation. Such a scheme is doomed to fail for the following reasons.

First of all, because of the annonymization process a seeder can always pretend to be just another proxy relaying for another ultimate seeder. As such it can request payment for one or more proxy steps and thus request more than what it is entitled to. This is an inherent seeder fraud incentive. Moreover, even if seeds are honest, they are then exposed as seeders because they request a payment very close to the circuit total. So that is a second seeder fraud incentive.

Secondly, the direct linear payment scheme requires that the seeder initiates the payment process and then each proxy in turn adds to the total for the downloader. Everyone has to wait for the payment request to eventually reach the downloader and for the payment to propagate back along the circuit. Again because of the annonymization process, proxies cannot initiate the payment process since they don't know their place along the circuit, and thus do not know what factor to multiply the circuit total with. The result is that the payment process depends on a faultless, honest and timely operation of all nodes along the circuit. A rather precarious assumption to have to make.

Thirdly, the downloader pays for the annonimity decision of the seeder. This is because the seeder controlls the

number of proxies it uses for its part of the circuit. And thus the seeder can enjoy annonimity at the expense of the downloader, who will only discover the cost after the bandwidth has been consumed.

One could consider the payment scheme with payment factor of <1, where the downloader initiates payment of the consumed amount and each node gets a percentage of what trickles down the chain. However this suffers from the same afflictions, but now the downloader is exposed and incentivized to pretend it is just a proxy and pay less than it should. It does however solve the waiting problem partly, since the dowloader initiates payment. And the seeder now gets payed less when the downloader has a long circuit part. It also incentivizes relaying over seeding since the payout is higher.

There are a few opportunities to mittigate the issues with these payment schemes. For example if the proxy nodes know their place on the circuit, then they can calculate the amount to request from the next node. This also solves the waiting problem. However this is directly contrairy to the design of the annonimity system that relies on not knowing the difference between a seeder starting a circuit or a proxy extending a circuit. Thus it also leaks the identity of the seeder and downloader.

Obviously any linear payment scheme (where payment factor >1 or <1) is not feasable nor elegant and leaks the identity of the seeder and downloader if they are honest. However setting the payment factor = 1 has some attractive qualities. Foremost, the payments become transparent and independent. Since all the nodes in the circuit know how much the cost will be, without knowing their place in the circuit, they can request payment independently. They do not have to wait for the payment to propagate back and forth. Secondly there is no increase of risk along the payment path. Thirdly the annonymous path length is no longer a factor in the cost.

It is not obvious how setting payment factor = 1 would incentivize relays, since nodes get paid equal for seeding or relaying. This is however a feature, since both are needed to keep the community healthy. It is expected that a large enough supply of seeders will provide enough seed capacity to meet the download demand. Once that demand is saturated, nodes will spend more bandwidth to relay since that is the only avenue to increase their ranking. On the other hand, relaying also has a lower barrier of entry than seeding. Seeding requires knowledge of a torrent, it's swarm and having a portion of the content, where as relaying is possible without any knowledge. All in all, an equal valuation of seeding and relaying might be exactly what is needed.

Another though might be to add a “relay” metric to multichain, so in addition to the total up/down there would be a total relay up/down counter set. Notice that this scheme also runs into problems because of the annonymization process. By design it would leak the identity of the seeder and downloader. These would in turn be forced to pretend to be a relay since that keeps their role hidden. Everything would then be counted on the “relay” counter, effectively degrading this scheme to using a single counter.

In conclusion, to incentivize relays they would need to be identified, which implicitly leaks the identity of the seeder and downloader. The only recourse therefor, is to depend on the absolute value of the upload metric as ranking, since this counts the total contribution to the community, either as a seed or as a relay. If the download/upload ratio is larger than 1 (plus a bit for smudge), then that node should be shunned from obtaining services at all since it is about to start free riding.

Hidden multichain

- The tunnel initiator spawns new random multichain identities like the XYZ identity at a certain interval (which can itself be random).

- Each identity has a lifetime (by either a despawn probability every time tick or expiry time).

- At the end of an identities lifetime it starts to sign over it's positive standing to newer identities. So the tunnel initiator ends up with a set of identities that collectively contain its multichain standing.

- After each crypto setup step in tunnel extension, the tunnel initiator sends a message along the lines of "this tunnel will be payed by multichain identity ABC"

- At the end of each circuit, the tunnel initiator can pay each proxy along the way, through the tunnel.

This allows multichain transactions through (hidden) tunnels. The achillies heel being the signing over of multichain "value", this is not directly possible in multichain, and at some point these transactions would be very large relative to a regular tunnel session. So they would stand out eventually, burning the expiring identity but also the next identity.

Hidden payments

With the True Halves, we don't need to have the full counter party public key, a hash of that key suffices. Doing this saves some bytes but also it obfusticates the counterparty. Only when the other party counter signs is its identity revealed.

Fraud Rollbacks

If there exists a cryptographic fraud proof, then any node in possesion of this proof can rollback the effects of the fraudulent node on it's multichain.

Non countersigning

We can suspend interaction with a node until it counter signs our request. But if nodes do not persist counters (disk full, crash before flush, etc) it will never counter sign our interaction. Should the interaction suspension time out after some time? (Relative to the size of the risk absorbed?)

Moving blocks

Nodes can decide what their history is. Since we ask them nicely for their chain. The current attacks (except sybil) are possible because a node can make it's own claims about what it's chain is composed of. However if the network where to store the chain (for example in a DHT), then this would no longer be possible. The sybil attack would be slightly mitigated because the nodes need to actually exist in some way. A pair of blocks that confirm an instance of fraud will continue to move through the network because of node churn and eventually both fraud

blocks will end up on the same node. As such a node storing blocks can detect fraud and report it. This does depend on the network storage not forgetting about blocks, and guarding against active purging of fraudulent records.

Risk Management

How much traffic should a node be altruistic about (Alt_size)? It is a risk management problem <look for lit on this> and also a security issue. It is an important parameter with large influences on the resulting system. It is especially important for "new" hosts, that have not been interacted with before. My thoughts on this:

- We could try to get it constant-ish over the whole network. (Requires estimation of network size)

- There is something to be said for differentiation, raise the limit for nodes which we have successfully interacted with. (up to some max?)

- Change it to the time domain. How long to wait before wanting a signature.

- Make it randomized within certain bounds, get signatures when you want but also obfusticate interactions.

Maximum signing cap

Are transfer-all style trasactions usefull or harmfull?

Do my alterations preserve security & annonimity?

Make sure my stuff doesn't leak or cock-up in any security way.

Implementation

- True half blocks, symmetrical data structure, immutable messages & blocks are usefull for dispersy. Improved efficiency. Simplified code (almost halved), esp. database code.

- Improved gumby framework to control multiple communities during experiments

- (Local) Validator, check blocks with respect to what is already known.

- Crawler improvements, crawl in batches to reduce messaging overhead.

- (Relay incentivizing)

Impl. TODO:

- Walker

- Actively sample multichain blocks from other nodes

- Fraud detection (and exposure if detected)

- Base “service request? Yes/No” decisions on multichain status.

Experiments

Analysis?

Conclusion