The Stick And The Carrot

About methods to incentivize Tribler peers

Voorwoord

Stuff.

Inhoud

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 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. 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 blockchain inspired accounting mechanism. It is intended to supplant 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.

<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.

Blockchain Technology

The 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 51% of the compute capacity in the whole network.<lit> Recent research has reduced this to 33% <lit>. Combined with possible 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.

MultiChain

Inspired 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.

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

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 by keeping 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.

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

MultiChain Engineering

<story here about what is to come in this section, mainly engineering stuff that has been done to get multichain out in the field, or be ready for it atleast.>

Towards True Halves

The MultiChain as revised by Veldhuizen <lit> allows the use of just the signature request as previous block, as opposed to the Norberhuis design where only the signed response is valid. Fundamentally this changed the MultiChain from a synchronized mutual agreement to an independent mutual claim. But Veldhuizen only used the change to remove the need for a blocking wait before starting another transaction. That was a missed opportunity, since the fundamental change to a mutual claim system can bring more benefits.

MultiChain has now been refactored to fully embrace the independent mutual claim system. In this version each transaction (full block) is composed of a claim (half block) made by an initiator and compatible claims made by counter parties. Transactions that miss half blocks do not add up to a full block and are thus considered invalid. The half blocks are compatibile when they provide a consistent link to eachother and may include furthere implementation dependent restrictions. In the Tribler case transactions only happen between 2 peers that exhange data, and a half block is compatible if the up counter of either half block is equal to the down counter of the other half block. This brings the following benefits:

- Each half block is a claim of what the signing peer believes to be true. This provides an ego-centric viewpoint when programming, obliviating the need to continually check if the peer is a requester or responder.
- It vastly simplifies the persistence layer which can now store each half block independently. Previously each search for a given public key would have to search the table for a requester with that public key, and search the table for a responder with that public key and union the two together. After which the processing code had to figure out if a record was returned because of a match on the requester or responder side.
- Half blocks are by design inmutable. This makes it possible to distribute, cache and store them without restriction. A desirable property in distributed systems. It allso adds to the robustness, since the signing parties don't even 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 it is valid, if it needs to be stored, or even if it needs to create and sign a compatible half block.
- The request/response mechanic from Dispersy can be discarded. Since there is no need to keep track of outstanding requests and wich 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. That has to be done by MultiChain code anyway.

The detailed design of this revision of the Tribler MultiChain started with the half blocks. In each half block there has to be the following information:

- Up & Down: The total amount of this transaction. The linked half block will have the values reversed, since upload for one is download for the other and vice versa.
- Total Up & Total Down: The total traffic counters for the peer creating the half block.
- The identification of the peer creating the half block
- The sequence number of the half block on the MultiChain of the signing peer
- Previous pointer, this will point to the previous block on the chain of the signing peer.
- A link to connect the chains of the transaction participants.
- A cryptographic signature over all the data in the half block.

There are several choices to make here.

- What is the identification of a peer composed of? In the Norberhuis implementation of the Tribler MultiChain, the identification used was the sha-1 hash of a public key. This is efficient since few bytes need to be transmitted compared to the full public key, and also the Dispersy middleware makes extensive use of this hashed public key. However when crawling the MultiChain of other peers there will be signatures by unknown public keys. Because of the hashing, it is suddenly impossible to verify a block, 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.
- The public key and sequence number together are the identification of every half block. But so is the hash of a half block. (usually... hash collisions *are* possible) So does one identification scheme have an advantage over the other? Not really, so there is no reason to commit to any one scheme except to make things consitent for other developers. Actually both are needed.
- What is the previous pointer composed of? Or, since there is already a sequence number, doesn't that imply what the previous block is? Consider a peer that is fraudulent, it will sign the same sequence number for two (or more) transactions. Without a previous pointer both these branches would automatically merge again 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 lie.
- How are half blocks linked to eachother to form a full block? What is the link to the chain of transaction participants composed of? Any peer that is part of the transaction must be identified. So atleast the public keys of all participants must be included. The initiator does not know the sequence number that the participant peers will use to create their half block (that is impossible), so the initiator cannot exactly identify a block on the chain of the participants that will be the compatible half block. However

the other participants can all link their half blocks back to the exact public key and sequence number used by the initiator.

Possible to use linked-last-seen-hash field. Perhaps that makes iterative consensus easier?

There are roughly 4 attacks possible:

- Fudging the counters

- Double link to a initiating half block

- Double sign a sequence number

- sybils.

The first two are trivial to detect because the initiator must provide the previous blocks to ensure that the previous hash link checks out and that there will be no holes in the chain. Using the previous blocks it is easy to verify that the counters add up, and that there are no double links to an initiating half block. Sybil defence on the MultiChain has been proposed by Otte<lit>. That leaves the double sequence number signing, this attack is detectable if two or more blocks with the same public key and sequence number are known to a single node.

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 s. 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 since it would need to double sign any sequence number after s, 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 business after the branch at sequence number s 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 fraud 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 the private fraud block is only spread to the peers of the next x transactions of B. Thus 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 interact with 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 p, 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 successfull interactions after the fraud, before the fraud is really hidden.

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 this scheme with a hardcoded x will not work.

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. (Experiment: database improvement stats)

- Improved gumby framework to control multiple communities during experiments

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

- Fraud detection

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

- (Relay incentivizing)

- Eliminated sign anything, actually crawl the requester if not enough information is known.

Impl. TODO:

- Walker

- Fraud exposure

- Actively sample MultiChain blocks from other nodes

- Base "service request? Yes/No" decisions on MultiChain status.

Experiments

Analysis?

Conclusion

Bibliography

@misc{cryptoeprint:2015:578,

author = {Arthur Gervais and Hubert Ritzdorf and Ghassan O. Karame and Srdjan Capkun},

title = {Tampering with the Delivery of Blocks and Transactions in Bitcoin},

howpublished = {Cryptology ePrint Archive, Report 2015/578},

year = {2015},

note = {\url{http://eprint.iacr.org/2015/578}},

}