Efficient verifiable transaction protocol using witnessing on top of ipfs

Michael Simkin
simsim314@gmail.com

August 2019

Abstract

Validation of transaction in decentralized data structure is a well known problem. We propose an algorithm that prevents attacks with very high probability using witnessing protocol. Unlike gossip or pow the system is (deterministic) randomly shuffling verification nodes to validate operations and transactions done by other nodes. We use only locally connected nodes and randomly connected graph to allow strict validation of all operations, and punish attackers. The system can recognize attackers with very high probability, as the system is node centric, attackers are removed from the swarm which achieves very high consistency due to node centricity and previously trusted parties verifiable consensuses. Trust and status is earned in the system with time and useful computational effort - to decrease the probability of attacks even further. The proposed protocol is converting static ipfs into dynamically verifiable super computer using oracle like approaches. The validation is done on relatively small amount verifiably random trusted nodes, which sign any operations done inside the system - making it highly scalable and trustworthy.

1 Overview

Lets assume A and B made some transaction between them. Then A claims the transaction was x while B claims it was y. As long as we have node centric system we would say they both right as there is no "objective" way to know who's right. The simplest way to make sure the transaction was x or y is to call someone random we can verify doesn't know A nor B, to write x in his transaction catalog. Every transaction should come with deterministically yet randomly defined witnesses, signing and verifying it. So when A and B come to the "court" we can check out the witnesses protocols, make sure the witnesses were chosen by the shuffling algorithm, and track back as long as needed to make sure who exactly is lying. As the system is suspicious by definition, and requires some amount of participation and time to gain trust and has very harsh punishment protocol (excluding the liars from the swarm), people will prefer

to avoid attacking it. Because every node knows his own history well, and can validate every transaction regarding his own state, anyone who will try to attack him, will be provably discarded. As the system can build itself from stage 0, any attack can be rejected with enough validation effort, and the initial "valid" swarm will always be verifiably intact. As we use ipfs and local private secure data storage per node, attacks on the whole system are virtually impossible. Any specific node that was attacked can also be restored using the witnessing history of his state. The system stores copy of every wallet and its history at least X16 times, and every wallet that was online once will be remembered and simulated further on (up to some degree) even when offline. When wallet is discarded due to inactivity, all his witnessing conformations are copies to active wallets. Yet the wallet itself although the information about it continues to be kept inside the active nodes, if they will be discarded as well due to inactivity will not be able to restore itself completely. This is why wallets needs to be continuously operational, or frozen and some amount of assets are invested into keeping the wallet alive in this state. People who make transaction and then suddenly disappear for unknown amount of time - without any effort to keep their wallet, will eventually be erased from the system, thus we avoid spam to high extent. Obviously if some assets were kept inside this wallet, the system will use the assets before erasing the wallet.

2 The data structure

At any given point in time, each node has 16 patrons (the nodes that validate his operations by witnessing) and he's witnessing 16 other nodes.

Each node has two documents (hosted in his ipfs node and backed up by all witnesses):
1. History of his states and operations changing his state (or others).
2. History of his witnesses to other nodes.

Those documents are not full and might contain links to other documents (also hosted on ipfs and pinned by other patron nodes).

When shuffling is occurring (see next chapter), each node is opening a communication channel to 16 new patrons. Each patron is copying only the current state of the node, to his witnessing document. The communication is just a new file with password, used by the node and mutated by deterministic function to generate the password using the state of the new patron. Then the password is secretly shared to the patron in read only format, so only he could read the messages sent by the node. The patron is listening to the messages asking him to verify different actions done by the node. The patron is validating the actions and writes the results in his witnessing document (encrypted by the password).

After the cycle is finished (after 6hr) the password is published, and the ver-

ification document is linked to immutable location and pinned/saved locally by both the node and the patron. Thus every operation done by the node is saved 16 times, and is immutable and freely available to verification. The immutable state of the patron and of the node are also added to the shared document so that history retrieval of that point can be done easily.

Verification is done in such a way that only limited set of operations is available and each operation is associated with static ipfs file, hard coded into the running system. Thus if the verification was falsified by all the parties (say the node convinced all the patrons to lie), anyone can check there was a lie done by the nodes - and reveal the liars at any point in time.

The state include:
1. Dictionary from currency to amount in the node wallet.
2. Personal word that can be changed from shuffle to shuffle.
3. Node id.
4. Random seed (changes every call to random)
5. List of promises to B_i of currency amount X_c till time t_i
6. Pointer to list of trusted verification functions.
7. Status per verification protocol.
8. Provides offline simulation service or not.
9. Validation protocol.
10. ID of node added me to the system.
11. IDs of all wallets I've added to the system.

There are only 6 operations which require verification:
1. Choose "random" patron. (for redirecting request).
2. Close amount X_c for B_i to period t_i.
3. Close amount X_c for offline simulation.
4. Close amount X_c for trading by y_c by price.
5. Retrieve all promised X_{c_i} from B.
6. Retrieve value for simulation activity.
7. Register new word for the shuffling.
8. Register new user.
9. Vote to change transaction coin amount.
10. Commit trade.

Everything that changes state is validated by all of the patrons.

Notice every payment is done in two steps:
1. The node is promising value to someone.
2. This someone calls the node and asks to give him the promise.

While the promise requires verification from the patrons of the node only, the transaction to a new node is verified by both patrons, of the sender and of the receiver patrons. Only after verification by all of the patrons the receiver can access the new amount of currency to his wallet.

Transaction is done by closing some amount of currency into some side document. When payment is received the paying node state does not change. The patrons swarm are managing this in between state. This kind of protocol allows safe swaps as well as one sided safe payments. Because the validation of activity is done by independent witnesses which validate the transaction.

3 Shuffling protocol

Shuffling is done every 6 hours. Every patron is sending every node he witnessing to some of his patrons. This random permutation is ordered alphabetically by id, and used deterministically. The patron redirection is also verified and is the last verifiable operation done by the node. After that the documents are published with their static immutable path.

Each node can define a new word, which comes into action between the closure of the first cycle and before the second cycle comes into place. This can be done automatically by calling some private function that mutates the word, or by hand. The node is also doing a work of random walk to find solution to some puzzle, which includes all the new words from all the nodes (specific well defined algorithm is stopping only when some criteria is met which can't be computed beforehand). The new word is secret up until everything from the previous cycle is openly published. The new random state is a function of all the new words from all the new patrons - thus making it harder to guess what will be the redirection action (for high quality transaction verification) of the node would be in the new cycle. This makes sure the conspiracy will not succeed because everyone's random state is secretly changes - and even one random person who's not participating in the conspiracy changes all the states of everyone (for safer security one can make another patron swap after the random shuffle - thus every 6hr jumping two connected nodes, the second jump is very unpredictable).

Notice in case of disagreement - we don't continue with majority, we make investigation and disqualify nodes that were lying. We require complete honesty (unless some good reason was found for misalignment in case of function of time for example, or online/offline switches). This is all done automatically - depending on the lie, and the ability of the minority to defend itself by providing proofs (for example online status change recorded in the node itself).

4 Safety, status and proof of useful work

Our safety is reached from well documented witnessing protocol, this is why the system is not enforced to do any work - and the liars black list can be generated

from each node using its own history and connecting to all trusted parties from this node. For example if some line is disputed 50/50 we can check ourselves the results, and black list the 50 liars. As everything is documented from the start and each node documents its own history, each node is capable to recognize the liars from his "node centric" approach. As we achieve perfect safety by blacklisting attackers, we don't need any proof of work at all. We can even use 3 witnesses in normal circumstances and it will still be safe system. Yet attacks might occur, hackers can take control over parties previously trusted. Unless hackers take 100% of the network we can always redeem all the parts of it using our own data and trusted parties which should be all in complete synchronicity. When they don't - especially when conspiracy are introduced, all the trusted previously parties should lie together. When this happens (very unlikely) we still can confirm the lies if any node is acting by the rules of the protocol.

To make the conspiracies and high amount of nodes attack we use some proof of work. This is done as immune for the whole swarm. Traditionally coins are associated with computational work - and we see why not to use this mental association to add extra protection to our network. To incentivize it, we add status to people who're securing their transaction with some extra work. Anyone can choose the work that is being done. If an extra work is asked from the patron but the patron is not willing to do this work - as it's not trusted script - he can transfer his work to his patron which is willing to do this work.

The work is simply a request to run some function located in js file on the ipfs. The files can be uploaded by anyone - and trusted parties can validate the "cosher" verification functions. Each individual can trust his own trusted parties to run verifications. Attacks from this direction can create verifications black lists. In general one should not run not trusted verification function, yet we can ensure that anyone who wants some computational effort to be done needs to implement some interface in form of function list, and the ipfs based witnessing supercomputer will make computational work to contribute. This work can include many useful applications, and the supercomputer is generating high quality trustworthy results. As someone is asking to verify by some protocol, he is paying some compensation (defined by the protocol and the market) and can gain status points from it. The relation between verification and status is open per each specific verification protocol implementation. Although the status is verification centered the interface provides rough estimation of how much work is done, and total status score from all the verifications protocol provides also single per user "global" status. Obviously one can compute the status of others using only his trusted list, thus the status is node centric as well.

Sensitive verifications can be done not only by direct patrons but with several parties randomly shuffled of patron branching. This reduces the probability of attacks on specific record.

If some patron frozen the wallet and left the system, we must provide evi-

dence for that event and skip a patron to the next one (we continue to simulate its state from virtual node - see chapter 7).

We don't use any public data - everything is local, thus no attacks can be done not from the local (connected) env. on the node. Every smart contract is run by individual and his requests (generally we're currently focused on implementing transaction of value, but the same idea can implement any algorithms run - we provide list of trusted contracts to execute and generate trusted coin). We can support two peers transactions of states, as well as any transaction of states of smart contract, which always manged as list of states of individuals (each contract is assuming each individual lived in its own personal "bubble").

There is also an option of single patron attack. We can ensure that someone is incapable to make all nodes to be of his choosing, but if safety a word is chosen properly and no patrons changed their safety word (highly improbable but possible), then one can find such a word that fits some of the patrons to his favor using brute force search. As the system is making random shuffling using cryptography - there is no way to find a word that fits to all 16 patrons.

5 Value and incentives

As rule of thumb every work done inside the system is associated with some coin, and this is out of the box protocol. We don't try to make single coin, but to have a wide array of coins and a market place where their value can be estimated. Every useful action is incentivized by weekly generated new coin. Thus we have on one hand a constant supply of the coin from the past and on the other hand the coins are traded, and their value is used to backup new operations. We don't use gas instead each verification protocol might incentivize the nodes not only by status but also with some coin. In the future we can also ask for further development of the system or some specific adaption of the system by using some DAO coin, and the most profitable actions will be naturally implemented by developers who want to implement the feature asked by the DAO for the promised price.

One central coin is introduced to provide stability to the market and everyone can trade relative to it. The coin is launched with the system, and early adapters earn it per operation, except of the usual incentives done by the system. This is only done in the beginning to incentivize adaption.

Every useful operation and specifically witnessing of the transactions are incentivized by native coin (weekly updated version). Each week there is constant amount of coins generated, and they can be used to provide backup for the next weeks operations requests for useful work. Harsh inflation is avoided by using only coins which can be seen as IRC20 tokens divided in the community

by contribution value. Thus each coin can change its price without influencing other coins - the natural market forces will guide the system to an equilibrium (there will be a lot of coins - but each coin amount is constant and each coin can completely lose its value).

To prevent transaction spamming (transactions done to only generate activity in the system), we introduce transaction coin. Each individual in the beginning of a week gets 1000 transaction coins, and the coin is tradable (we want to ensure transactions and therefore can't let the patrons decide if they agree to perform transaction validation). Transaction coin is tradable so if someone wants to make more transactions he can always buy more transaction coins on the market (most people don't make 1000 transactions in a week, so abundance of transaction coins is probable unless spammers try to take over - then the coin price is raised). Transaction coin is deleted when transaction is made. Because after the closure of the week the transaction coins is constant, people will avoid making empty transaction instead of keeping the coin for later trade.

6 Wallet disconnect: offline, freeze, restored, removed.

Wallets that go offline continue to be simulated by the swarm by action called virtualization of the wallet. As everything is done by witnesses, the witnessing swarm can also register all relevant operations to a separate document. Virtualization is requiring others to make work for you - and therefore costs value. An hour of offline simulation is offered as coin. Each provider of the service can limit amount of simulation work he's willing to allow on his node (but must provide minimal amount of offline work - at least 20 hours of offline nodes simulations per hour - if he wants to be regarded as offline supporting node). Each node gets offline coins for his promise, but must provide the work per offline coins provided to him. The coin disappears when simulation offline takes place.

Wallets that often go offline can be patronized only by wallets that can simulate offline wallets. Wallets that go offline often can't promise to simulate other wallets when they go offline. Yet a wallet can make promise for next hour.

Wallet that didn't keep its promise and went offline - is penalized by the offline coins are taken in rate 200:1 per missing minute.

While offline is natural state of affairs some nodes can simply disappear from the radar, while kept simulating. When the offline simulation backing resource is depleted, the wallet starts to overgo discarding protocol steps. He stops to be simulated, all his witnessing actions are being backed up by the system and

other nodes. This work is distributed randomly among the active nodes. If then he wants to come back to his previous state, and the information about his state is still kept in the swarm, for a price of restoration special algorithm is devised to restore his state. This is highly penalized.

Restoring lost wallet can be also done by attaching email/tel to the wallet. Thus even when the wallet is lost - email verification can be sent to restore its state. Restoring is basically walking from witness to witness and retract the last state recorded by the witness. As disappearing is rare in the system, one should usually be able to resurrect himself.

Special freeze state is provided to freeze the wallet. He stops to be simulated but his internal state and everything he witness is kept by the swarm. He is out of the game - but freeze coin is specially payed to freeze keeping service providers. The nodes that provide freeze service can go offline, but should not disappear. Like in case of offline the freeze coin is freely available in the market place, everyone can buy the coin and trade it for the freezing services. Freeze state requires 6hr wait to enter the new cycle of work, while the offline node can come online and make transactions immediately.

Wallets are recommended to buy and close amount of offline coin as well as freeze coin. When the offline coin is depleted, the freeze coin is taking place automatically. Without closing any amount of offline coin - going offline will automatically delete the wallet.

7 New wallets trust and status

In the beginning new wallets are welcomed and are incentivized by the central system coin. Yet the system does not trust new wallets witnessing functionality at first. One should provide reliable witnessing for a while, having trusted witnesses that the witnessing services provided by the new wallets are valid. As the wallet gain status and trust (status is provided by PoUW the wallet is engaging, and trust is simply linear to the time the wallet was active - frozen wallets don't gain trust points).

New wallets are born with some amount of offline coin it can offer when it goes offline (but can't trade it). Also after some work done by the wallet, he can buy new offline points. In the beginning he can immediately can trade his early adapter coins to offline coin (but not recommended).

8 Conflict resolution and validations

Because everything is recorded and kept in the witnessing documents, every action can be validated. In case of disagreements or conflict a verification protocol for verification coin is introduced. For example I claim I had 100 coins but now I have 30. When I look into history some invalid transaction is being introduced to my records. I can then walk to witnesses and ask them if they have the same transaction as well. If the transaction is present in all witness private historical immutable catalogs protected each by its own private key - then there is nothing else that can be done. The transaction is valid.

To protect itself from attacks like this, each wallet can decide what kind of useful work and what amount he is requiring for his transactions. Each record of transaction which violates this protocol is automatically considered an attack. There is no limit of work or witness requests - but some of there requests can fail and the value for work spent is not returned.

Special kind of attack can be done by simulating parallel swarm and using same algorithm. This is why each node keeps the adding wallets in his state. While the parallel swarm might be valid its root node would be different. Anything else that tries to mix will be force to violate some witnessing, and will be discarded or added to black list as fraud.

Other than that in the later stages we will implement active verification protocol - a nodes that will go over the whole swarm and make sure no attacks are made. If someone is being attacked and we verify it - we start to actively search for attacking source. Some kind of immune system is introduced. Just like in other coin cases one can sign in to immunity verification, by buying immunity coin on the market. Everyone who wants to provide immunity can generate the coin by providing actions vs. the presented coins payed by the customers.

Attacking wallets are added to the black lists of the system. The wallets can appeal and claim being hacked. In this case they're restored for some price - and a mark is done of being suspicious (trust points are deduced). Low trust wallets that made attacks are black listed forever, and their assets are confiscated by the system.

9 Full node

We implement the coin design on top of ipfs. We have two version of the product - in browser and in node. The full nodes will probably prefer to run inside a node env. Full nodes provide a whole pack of the services mentioned above. They simulate offline wallets, freeze wallets, make a lot of useful work, and monitor for attacks. They also collect data for faster node to node communication, as

they collect the tree of all wallets. Full node is not something special it's just a node that provides a lot of services.

We use low level ipfs functionality and everything is done by simple yet robust algorithms with witnessing validation as explained. We basically provide ipfs nodes with pinned document by the whole swarm to produce verifiable well scalable super computer on top of ipfs.

10 Appendix A: General smart contract

We use verifiable trusted function f which is also pure i.e. without side effect. It has only input state and output state and execution time. Execution time is execution cycles inside assembly code. The node that requires witnessing is asking the validators swarm (patrons branch) to verify and sign the result. They run the code with the same input - and write the output state hash in their witnessing document. The model is the same - each one gets CPU coin for the promise to run validations, and CPU coin is burned when action is taken (payed by the node - like for gas). The validations static document address of the patron validations is recorded in the transaction together with pointer to the new state of the contract.

The format:

(Contract address), (function call), (input), (prev state) $\rightarrow$ ($newstate$).
$validationid1$
$validationid2$
$validationid3$
$validationid4$
.
.

This is created each time call to contract is validated. This then referenced inside the transactions list of IPNS of the user, and the smart contract IPNS is pointing to the new state.

11 Appendix B: Market

Market place is inherently needed for this application. Market actions swaps are installed as part of the initial implementation. Markets are using atomic swaps to provide services. Market swap costs two transaction coins the lower amount is paying for both parties. Atomic swaps are providing friction-less market - as everything is done by agreement the prices are optimized for both parties. (See friction-less market document).

12 Appendix C: Privacy

We are starting from open trades (bitcoin ethereum open design) but intend to improve the system by zkp very fast. This will make it harder to restore the lost data, but will provide privacy. Once full smart contracts support will be introduced, zkp coins will be implemented as part of the design.

13 Appendix D: Compatibility

Our witness based protocol allows to remove/close some amount of value in one system verifiably by smart contract and witnesses and then transfer it to our system. For example one can close some amount of ethereum provide a key and witnesses can see he closed some value in smart contract so he can get this same amount in the current system. The same trick can be done with fiat money - one can make a video where he burns some cash required by witnessing protocol (body language can be attached to hashes and words of his patrons), this will allow him to verifiable transfer fiat representation to our system. This action can be done only once in 6hr.