catalyst

Catalyst Network

Technical White Paper

Catalyst Network - Technical White Paper (DRAFT - V1)

Joseph Kearney*- Atlas City

November 21, 2019

Abstract

Distributed Ledger Technologies (DLTs) over the last decade have grown at an extraordinary rate, with new technologies being announced, developed and released on a daily basis. A majority of these technologies have not progressed from the original concepts set out by Satoshi Nakamoto in his seminal paper [1] with regards to speed, security, scalability and decentralisation. There has however been a clear demonstration for the appetite and innovation for wider markets to adopt these technologies. Catalyst marries together many existing technologies as well as some novel ones in order to create a framework which give a fast, scalable and fair ledger.

This document is a work in progress and is subject to change.

*joseph.kearney@atlascity.io

Contents

1 Catalyst Consensus Mechanism

Distributed networks have no single point of trust to determine the validity of transactions, therefore concurrency must be ensured by other methods. Typically this requires a majority of the network's participants to agree on a particular update of the ledger and the account balances / holdings held within. Blockchain technologies generally employ Proof-of-Work (PoW) and occasionally Proof-of-Stake (PoS) mechanisms in order to gain consensus across a network. However, these methods are prone to increasing centralization at scale as well as in the case of PoW high energy consumption. Other networks employ a small amount of trusted nodes that ensure the validity of transactions, however this is highly centralised and almost as fallible as the single point of failure systems that DLT endeavors to avoid.

Catalyst integrates a newly designed consensus mechanism, based on Probabilistic Byzantine Fault Tolerance (PBFT). This is a collaborative rather than competitive protocol, meaning that all honest work performed by nodes on the network benefits the security of the network and that all participating nodes are rewarded equally. For each ledger cycle a random selection of worker nodes are selected, the nodes become the producers for a cycle or number of cycles. The producer nodes perform work in the form of compiling and validating transaction thereby extracting a ledger state change for that cycle.

The protocol is split into four distinct phases:

- Construction Phase - Producer nodes that have been selected create what they believe to be the correct update of the ledger. They then distribute this proposed ledger update in the form of it hash digest.
- Campaigning Phase - Producer nodes designate and declare what they propose to be the Ledger State Update as determined by the values it collects from other producer nodes in the producer pool.
- Voting Phase - Producer nodes vote for what they believe to be the most popular Ledger State Update as determined by the producer pool according to the candidates for the most popular update from the other producer nodes.
- Synchronisation Phase - In this phase the producers who have computed the correct ledger update can broadcast this update the rest of the network.

This section is based on the work set out in [4], where the original research into the creation of a new consensus mechanism are laid out.

The various parameters and thresholds mentioned in this chapter and their impact on the levels of security and confidence in the successful production of a ledger state update are discussed in the paper found here [8].

1.1 Producer Node Selection

Original work for the Catalyst consensus mechanism states that the peers are selected with relation to the PID and the hash of the previous data. As the PID can be manipulated by a user and thereby weight in their favor of selection to become a producer this is not usable.

Research into a RANDAO provides a viable alternative [5][6]. Generation of a process by which each user creates their own random value and then by combining these random numbers across the network you gain distributed generated pseudo-random numbers. The larger the network the more random the number will be. The proposed process works as follows:

- Define p as the maximum integer that the hashing function used can represent.
- Each node n in the worker pool N generates a random number r in the range $[0,\ p]$.
- To this random number the hash of the previous ledger state, D, must be added mod p.

- n then creates a Blake-2b hash of the combined random number $H(r + D)$.
- Each n must then send their value r to the contract.
- If they do not send their r value they are not eligible to become a producer node.
- Each n in N sends their $H(r + D)$ to a hard coded smart contract. This creates the global random value R. This is done through addition of all the random numbers mod p.
- The smart contract must determine:
  - Whether n did submit all correct information. i.e. r, and $H(r + D)$.
  - That n did in fact use the D value when generating the random number. This is done by taking the r value submitted by the user and hashing with D.
  - Ensuring n has paid a sufficient stake to take part in the selection process.
  - Validating that each worker n has only distributed one random number to the smart contract.
- Failure of any one of these points means the smart contract will not accept the submission of a random number from r and and stake made will be lost.
- This global random can the be used to determine the producers for the next cycle(s).
- This is done by determining the nodes that have a $H(r + D)$ closest to the R value.

This method is secure from manipulation as hashing algorithms are one way functions meaning that there is no provably efficient method to inverse a hashing function i.e. retrieving a message m from a digest $H(m)$. If a node does not input a value into the smart contract then they are not eligible for selection for becoming a producer for that cycle(s).

Addition of the D value is necessary as this will prevent a producer from using known random values to create a desired digest that gives them an advantage when being selected. The value for D must fulfill two rules, firstly it must always be the same for all nodes in the worker pool, secondly it must change with each draw of a random number when determining the random selection of producer nodes. This prevents a user creating a hash using known input and digest combinations to gain an advantage when being selected. Furthermore if the hash of the previous ledger state is used as D is ensures that a prospective producer node knows the current ledger state. If they do not then their random number will be invalid as $H(r+D_{prod}) \neq H(r+D)$.

As described in RANDAO, this can be further extended to implement staking. Nominal fees to contribute can be added in order to prevent DDOS. This is done in such a way that a nominal fee is added as to not dissuade users from legitimately wanting to become producers while dissuading malicious entities from attempting to perform a Sybil attack against the network in order to gain majority control over a producer pool for a cycle or multiple cycle. There is also the additional benefit of simplification of the overall consensus mechanism as it removes the need for a queuing mechanism as well as producing a verifiable method of keeping track of what nodes are registering to be workers for any given cycle. It also thereby in turn provides evidence to other nodes on the network who the producers for any given cycle are as the process will be verifiable.

This scheme will provide the Catalyst consensus mechanism with a verifiable, reliable and secure mechanism to generate a pool of producers. By randomly assigning which workers get to participate in the next ledger cycle, fairness is ensured.

1.2 Notation

A dictionary for the variables used in the proceeding description of the consensus mechanism can now be described for clarity:

- p - Describes a single producer that has been selected randomly to perform work for the network for a given cycle.

- P - The pool of producers who have been selected through the previously described producer selection mechanism for the particular cycle being described. It is made up of a set of producers p
- T - The mempool of any given producer p. It contains a group of transactions that will be used to for the update for a given cycle.
- t - A single transaction that is found within the mempool T.
- u - The local ledger state update generated by a single producer node. This is what that particular node believes to be the correct update for the given cycle.
- σ - A pseudorandomly generated number.
- E - The list of entries found within a single transaction t. A producer will producer will producer one of these lists for each valid transaction in T.
- d - Hash tree containing all the signatures extracted from the transactions in T.
- H - Hash of each list E combined with the salt σ.
- L - Alphanumerically ordered list the pair E, H, sorted according to H.
- G - The list of proposed updates u that producer p collects from other nodes in producer pool P. This is used to determine the most popular update.
- $\mathcal{L}(prod)$ - The list of producers from a particular producers set G that sent the most popular ledger state update.
- L_{CE} - A producers list of compensation entries i.e. who a producer believes did the correct work for the network and thereby should be rewarded.
- $\mathcal{L}(vote)$ - The list of producers a specific producer found to have forwarded the correct vote for the most popular ledger state update.
- LSU - Ledger State Update, this is the final ledger state update proposed by a producer node. It consists of the list of entries used to create the update, the hash tree containing the signatures and the list of compensation entries
- V - A producers vote for the correct LSU formed of the hash of the LSU, L_{vote} and the producers Id.
- $\mathbf{o}$ - The final producer output that is declared to the network.

1.3 Construction Phase

The first phase of the Catalyst consensus algorithm is the Construction Phase. Within which the selected producer nodes P calculate their proposed ledger state update or their local ledger state update. This is done by aggregating and validating all transactions that have occurred during a set time period. These transactions assuming their validity are integrated into the producers local ledger state update. From which they can create a hash of the update. This hash digest represents what they believe to be the correct update and is broadcast to the other producer nodes during for that cycle. Assuming the collision free nature of hash functions, the only mechanism for multiple producer nodes to have the same local ledger state update is by both using the same set of transactions.

Each producer p in the set of producers P follows the same protocol. The construction phase begins with producer p beginning their construction phase by flushing their mempool. This mempool T is made up of n transactions t where n is the number of transactions that have been broadcast to the network and have been stored by p. These transaction are used to create p's local ledger state update u.

The producer at this point also creates a hash trees d, this is to store the the signatures that are extracted from each transaction in T. A salt σ is created utilizing a pseudo-random number generator using the previous ledger state update $U - 1$ as its seed. p then follows the following steps:

1. Producer p verifies that each transaction in T is valid following the rules set out in [7]. From each of the n transactions in T the entries E that constitute the transaction t are extracted to form a list $E = e_1, ..., e_m$ for m entries in t. The producer should therefore end up with n lists for E from T. each signature from the transactions are also extracted and added to d.
2. p for each E it created then creates a corresponding hash digest as:
$$H = \mathcal{H}[E \mid\mid \sigma]$$
Each pair (E, H) is added to a list L.
3. p then sorts list L into lexicographical order according the hash values H.
4. The producer p then extracts the transaction fee value from each transaction in T to create v which is the total sum of all transaction fees.
5. The local ledger state update u for producer p can then be calculated. Firstly the list L is concatenated (denoted by $||$) with the hashtree d and a hash digest is created as:
$$u = \mathcal{H}(L \mid\mid d)$$
u is then concatenated with p's unique peer identifier Id to create:
$$h = u \mid\mid Id$$
6. h is then broadcast to the other producer nodes on the network.

Producer p also collects other producers partial ledger update values. At most they will collect $P - 1$ values. Optimally every producer in P will receive the same set of transactions therefore for every p in P will have the same partial ledger update u. However this is unlikely due to all transactions not being received by a small group of nodes. Equally they may not hold G where $G = h_1, ..., h_P$, meaning they may not receive a proposed update from all candidates.

1.4 Campaigning Phase

The second phase of the consensus mechanism is where producer p generates and proposes a candidate, which it calculates to be the most popular ledger state update.

Beginning this phase producer p has a set of partial ledger updates G that it has received from other producer nodes. Each h within G contains a producers hash of the proposed update (u and a peer identifier (Id). The most popular u value can be found, this gives us u^{maj} from there the subset G_{maj} can be created, which is the amount of votes for the most popular update. Two thresholds must be considered first is G_{min} this is the minimum amount of updates it has received from other producers in order to generate a valid candidate. The second is G_{thresh} this is the threshold value for which a minimum number of votes must be in favor of G_{maj} which the most popular vote found within G. So in order to proceed with declaring a candidate $G > G_{min}$ and $G_{maj} > G_{thresh}$.

Beginning this phase producer p has a set of partial ledger updates G that it has received from other producer nodes. Each h within G contains a producers hash of the proposed update (u and a peer identifier (Id). The most popular u value can be found, this gives us u^{maj} from there the subset G_{maj} can be created, which is the amount of votes for the most popular update. Two thresholds must be considered first is G_{min} this is the minimum amount of updates it has received from other producers in order to generate a valid candidate. The second is G_{thresh} this is the threshold value for which a minimum number of votes must be in favor of G_{maj} which the most popular vote found within G. So in order to proceed with declaring a candidate $G > G_{min}$ and $G_{maj} > G_{thresh}$.

If the thresholds are met the following can take place:

1. p creates a list $\mathcal{L}(prod)$. To this list p appends the identifier of any producer that correctly sent the u value that equals u^{maj}. If p's u value is also the same as u^{maj} then they should append their own Id.

2. Producer p then creates their candidate for the ledger update c which is calculated as $c = u^{maj} \mid\mid \#(\mathcal{L}(prod)) \mid\mid Id$
3. Producer p will then broadcast their preferred update c to the other producers.

p during this phase will be collecting the c values from other producers. At the end of this phase of the cycle p will hold a set of C candidates.

1.5 Voting Phase

The third phase of the ledger cycle is the Voting Phase within which a producer p from the C candidates it has received decides on what it believes should be the global ledger state update i.e. the update that should be applied to the ledger for that cycle.

1. p verifies that the same first hash value u^{maj} is embedded in a majority of producer candidates.
2. p is required at this point to have created the majority ledger update. As only with this partial update can they participate effectively. Therefore their $u == u^{maj}$

If each producer collects the first hash value generated by every producer, any two producers would build the same list of identifiers $\mathcal{L}(prod)$. However even in highly efficient distributed network it is unlikely that all producers will retrieve all information. Therefore it must be assumed that the list of identifiers held by p is incomplete, p however must ensure that they hold enough information to confidently issue a vote on the update to the ledger. The identifier of a producer is appended to the hash they distribute because:

- To verify that p is a producer node and that they in fact were one of the winners of the random draw described in **??**.
- To evaluate and track the quality of the work performed by p.
- To ensure that the correct producers withing P are rewarded for their work maintaining the ledger.

New transaction entries (compensation entries) are created using the list's $\mathcal{L}(prod)$ produced by the producers p. This allows reward and fees to be paid to the correct users. These new transaction entries will be appended to the final ledger state update for that cycle. Therefore a complete ledger state update should consist of:

- The list of transaction entries integrated into the new ledger state all E lists.
- The transaction signatures from T held in d.
- Compensation entries rewarding the producers from $\mathcal{L}(prod)$.

The voting process thereby must confirm the correct list of identifiers involved in producing the correct ledger state updates. The final list for $\mathcal{L}(prod)$ is generated by merging the producers lists together. In order for a producer p to have their Id added to the final $\mathcal{L}(prod)$ the must appear in at least 50% of the $\mathcal{L}(prod)$ lists distributed by P. This ensures that no producer can just append their own Id to their update whether that update was correct or not and gain a reward for it.

This process ensures that only producers that have provided the correct update may receive this part of the reward, thereby preventing unethical behavior from producers by declaring that they got the correct update when in fact they did not. However, it does introduce the problem of producers that have created the wrong ledger update being disincentiveise from proceeding in the consensus process further. Therefore producers that vote for the correct update whether or not they produced the correct update should also receive a portion of the reward. This is determined by the list $\mathcal{L}(vote)$ that each p creates to store the Id's of the producers which the gained votes from that voted for the correct update.

p follows a series of steps:

1. p generates a list $\mathcal{L}(vote)$ it then appends the Id of any other p that has forwarded a C candidate that satisfies $u == u^{maj}$.
2. p creates the combined list for $\mathcal{L}(prod)$ of any p from P that appears in $P/2$ lists.
3. Then p creates a list L_{CE}. This contains the contribution entries. This is for any user that is included in the combined $\mathcal{L}(prod)$. Each of these producers contained within this list will receive x tokens. If it is considered that X is the total number of tokens that are injected into the network at each cycle for the pool of P producers. The amount of entries into L_{CE} will be less than or equal to P. R is to be considered the total amount of reward that is to be split among all the producers that found the correct ledger update $R = f_{prod}X + v$ where f_{prod} is the fraction of the injected tokens that are given to users found in the combined $\mathcal{L}(prod)$. While $(1 - f_{prod})X$ is the value that is distributed to other nodes that have performed work.
4. Producer p can then create the candidate ledger state update for this cycle. This includes the reward allocations for producers according to their contributions. This ledger state update LSU can be defined as:

   $$\mathbf{LSU} = \mathbf{L} \mid\mid \mathbf{d} \mid\mid \mathbf{L_{CE}}$$

   p then computes its vote (or *producer vote*):

   $$\boxed{V = \mathcal{H}(LSU) \mid\mid \#(\mathcal{L}(vote)) \mid\mid Id}$$

5. p then forwards their V to the other producers and collects the producer votes V issued by its peers.

During the voting phase, the producer p collects the producer votes V broadcast by its peers. At the end of the voting phase , the producer p holds U producer votes V in its cache. This will be equal to or less than the number of producers who formed the correct update.

1.6 Synchronisation Phase

The Synchronisaction phase allows those nodes that have produced the correct ledger update to then pass this update onto the rest of the network, thereby integrating the changes held within upon the overall state of the ledger.

During the synchronisation phase p executes the following steps:

1. Producer p selects from the producer votes V that it recieved the most popular unique V. The number of most popular votes received is defined as V_{max}. This V_{max} value must exceed a specified threshold V_{thresh}. So $V_{max} \geq V_{thresh}$.
2. p creates a list $\mathcal{L}_{final}(vote)$. To this list p appends the identifier of any producer that their given $V == V_{max}$ *i.e.* they voted among the majority and only if they appear in at least half the $\mathcal{L}(vote)$ values from the V values p collected.
3. If the producer p also created the correct ledger state update LSU then the they can write it to the DFS with the content address CID [**?**].
4. Producer p then creates the final producer output that can be declared to the network. This output is:

   $$\boxed{\mathbf{o} = CID \mid\mid \#(\mathcal{L}_{final}(vote)) \mid\mid Id}$$

   The producer then broadcasts $\mathbf{o}$ to the network.

References

[1] S. Nakamoto *et al.*, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] A. Back *et al.*, “Hashcash-a denial of service counter-measure,” 2002.

[3] S. Hackernoon, “The ethereum-blockchain size has exceeded 1tb, and yes, it’s an issue.” https://hackernoon.com/the-ethereum-blockchain-size-has-exceeded-1tb-and-yes-its-an-issue-2b650b5f4f62, 19/05/2018.

[4] P. Bernat, J. Kearney, and F. Sage-Ling, “Catalyst network research: a new consensus protocol.” https://atlascitywebsitedocs.blob.core.windows.net/websitefiles/catalyst-consensus-paper.pdf, 07/11/2019.

[5] B. Skvorc, “Two point oh: Randomness.” https://our.status.im/two-point-oh-randomness/, 07/05/2019.

[6] randao, “randao.” https://github.com/randao/randao/, 26/03/2019.

[7] catalyst network, “Transactionvalidator.cs.” https://github.com/catalyst-network/Catalyst.Framework/blob/f23089d75ce876008ae1400f8044f538e07a3100/src/Catalyst.Core.Lib/Validators/TransactionValidator.cs, 07/11/2019.

[8] P. Bernat, J. Kearney, and F. Sage-Ling, “Catalyst network: Security considerations.” https://atlascitywebsitedocs.blob.core.windows.net/websitefiles/Catalyst%20Network%20Security%20Considerations.pdf, 07/11/2019.

[9] catalyst network, “Catalyst.core.modules.dfs.” https://github.com/catalyst-network/Catalyst.Framework/tree/develop/src/Catalyst.Core.Modules.Dfs, 07/11/2019.

[10] J. Benet, “Ipfs - content addressed, versioned, p2p file system,” *arXiv preprint arXiv:1407.3561*, 2014.

[11] catalyst network, “Bytesextensions.cs.” https://github.com/catalyst-network/Catalyst.Framework/blob/b761153e5e30a65b843d470cf60175256bd43276/src/Catalyst.Core.Lib/Extensions/BytesExtensions.cs, 07/11/2019.

[12] catalyst network, “Catalyst.core.modules.kvm.” https://github.com/catalyst-network/Catalyst.Framework/tree/develop/src/Catalyst.Core.Modules.Kvm, 07/11/2019.