Proof of Trust Protocol

For Blockchain Consensus and a Generalized Trust Network

Swapnil Pawar, ASQI Investment Managers

v1: March 15, 2022; v2: October 15, 2022

Abstract

The paper presents 'Proof of Trust' (PoT) consensus protocol that uses trust score of network participants informed by their community and historical behaviour. Current consensus protocols used by most blockchains continue to struggle with either scalability, owing to the energy-hungry processes or reliance on a single protocol currency for creation of 'stake'. Also, both proof-of-work and proof-of-stake protocols remain open to the risk of creeping centralization. In comparison, PoT is inherently non-concentrating and energy lite. It also has higher byzantine fault tolerance and greater resistance to denial-of-service attacks, 51% attacks and Sybil attacks than existing consensus protocols.

Introduction

Public blockchains require a consensus protocol that is Byzantine Fault Tolerant (BFT). Bitcoin's proof of work protocol[1] was a novel approach that worked well until the rewards for mining became large enough to led to a computing power arms race amongst the miners and the consequent energy wastage. An adverse effect of the same market forces was also the creation of 'mining pools' which went against the very idea of decentralization[2]!

There are several consensus protocols being used by the current public blockchains – including proof of work, proof of stake, proof of elapsed time, delegated proof of stake etc[3]. Almost all of these protocols work with the assumption of anonymous nodes and a memoryless system as regards their behaviour. This misses out on useful information that network participants may have about each other. This is akin to decentralized trustworthiness assessment of each node, by their peers. We build on this idea to create a protocol that uses such information along with updates based on actual behaviour of participants over time, in terms of honesty or lack of it in block creation, confirmation etc.

We introduce a new consensus protocol called 'Proof of Trust' – which uses the trust scores of participants as the primary input to arrive at consensus in a byzantine fault tolerant manner. We have borrowed some concepts from a well-known proof of stake protocol Tendermint[4]. However, we

replace the pure financial stake envisaged in it with a trust score of network participants. This approach enables a richer assessment of the probability of a given node being dishonest. It also creates a stronger disincentive over time for a given node to turn dishonest, because trust scores are harder to earn than to lose. Also, in PoT, unlike current public blockchains, individual nodes are required to be run by real persons with verifiable identity. These nodes participate in validating transactions and adding blocks - earning platform tokens and a higher probability of future selection for block addition for honest contributions. Conversely, for bad contributions or malicious behaviour, they get a sharp reduction in probability of future selection for block addition and also lose part of their deposited tokens. PoT has superior Byzantine Fault Tolerance because of its two-step confirmation of a block - first by a randomly selected committee and second by the rest of the network. Also, the validation in either case is not just based on simple majority but on trust score weighted assessment.

Trust scores are derived from a combination of social inputs (through an algorithm similar to Google' PageRank) and refinements based on honest or dishonest block creation and voting by network participants. A public blockchain that currently deploys Proof of Trust consensus protocol is Newrl, initially set up by ASQI Investment Managers and now decentralized in its operation and governance. In the rest of the paper, this implementation is referred to for specificity. Newrl generalizes trust scores beyond block creation and validation. All transactions carried out on it can potentially alter the trust scores of the parties involved. Also, inactions like failing to honour a contractual financial obligation can also lead to reduction of trust score. Lastly, to deal with real world physical verification requirements in specific context, auditor participants can also submit transactions that modify trust scores, backed by their observations (e.g. incorrect or missing title with property registrar for a real estate token).

Proof of Trust has the beneficial effect of bringing informal social capital into a formal context[5]. This paper focuses on the specific implementation of PoT in the context of achieving Practical Byzantine Fault Tolerance in a public blockchain context. A generalized implementation of trust scores for other uses like asset tokenization, borrowing, fund-raise etc through the Newrl blockchain is covered in a separate lite paper[6].

Overall Architecture of a Public Blockchain for Implementing Proof of Trust

The PoT protocol uses a state-based approach. For validating any transaction, having the latest state is sufficient and each valid transaction updates the state.

This state should have following components, which can be maintained as tables in a database.

1. Blocks and transactions tables
2. Identity tables: wallets, personids
3. Trust scores by personid combinations (including network trust score where source personid corresponds to the network trust manager smart contract)
4. Tokens
5. Balances by wallet-token combinations
6. Contracts
7. Distributed organizations
8. Nodes and their liveness status
9. Stake ledger

The state can be changed by execution of transactions by each node locally. Transactions are of following types.

Type	Function	Signed by
1	Identity management – create new personid and wallet, create a linked wallet etc	Introducer
2	Token management – create a new token, issue more of an existing token, destroy token etc	Asset custodian / contract manager
3	Smart contract execution – set up a smart contract from template, deploy it, execute specific functions in it, destroy it (if applicable)	Various
4	Two-way token transfer – sender1 sends token1 to sender 2 and sender2 sends token2 to sender1	Sender1 and Sender2
5	One-way token transfer – sender1 senders token1 to sender 2	Sender1
6	Alter trust score – person1 alters the trust score of connection directed from person1 to person2	Person1
7	Update network liveness of a node	Node owner
8	Smart contract internal transaction	None

A transaction in Newrl has the standard format as below.

{“transaction”: “data”, “signatures”: [{sign1}, {sign2}]}

Validation of a transaction is for its signatures as well as economics e.g. for a transfer transaction, the sender needs to have adequate balance.

“Proof of Trust” Consensus Protocol (PoT)

There are three types of computations carried out in a typical public blockchain[3] - transaction validation, block creation and verification of a block upon its broadcast and reception by other nodes.

Transaction handling

In PoT, any node can broadcast a transaction to its peers. Receiving peers ignore transactions that they already have (checked using transaction id). For valid transactions, each node stores them in the local memory pool and transmits it onwards through a gossip protocol using the transport layer (by default, this is the libp2p protocol).

Selection of block creation committee and block minting node

PoT involves block creation by a chosen node and verified by a randomly selected committee. The minting block is selected randomly inside the committee, where the selection probability is proportional to its trust score. Additionally, each node is required to deposit a fixed number of valuable tokens of specified types (Newrl token or USD stablecoins) in a dedicated smart contract that governs the protocol.

For arriving at consensus, the network undertakes the following steps in each round for a new block creation. The below process begins for a given block immediately upon the inclusion of the previous block.

We assume the following at the end of the state update after the latest block inclusion.

N = Number of total nodes in the network (from the nodes table)

e_i = Liveness score of the i^{th} node (from the nodes table, currently set as 1 for all entries)

s_i = Network trust score of the i^{th} node (derived from the trust score table)

$H_{committee}$ = Threshold of trust score for inclusion in the block creation committee (set at 0 at genesis)

H_{live} = Threshold of liveness score for inclusion in the block creation committee (set at 1 at genesis)

C = Size of the block creation committee (set at 10 at genesis)

The selection space S for block creation group is created is follows.

$$node_i \in S \; if \; s_i \geq H_{committee} \; and \; e_i \geq H_{live}$$

$$S = sort(S, ascending, peerid)$$

Since the state is same for all participants, S will be identical for them.

We select a total of C nodes in the committee from S.

$$S_1 = S$$

$$d_1 = \; random(seed = hash_{latest_{block}}) \times size(S_1)$$

$$node_j = node \; at \; d_j^{th} place \; in \; S_j$$

$$S_j = S_{j-1} - node_j$$

$$d_j = random(seed = d_{j-1}) \times size(S_j)$$

When *j* = *C*, the selection process halts. *C* nodes are now selected for the committee.

The block minting node is always selected from the committee itself through a random selection with the previous block hash as the seed. Since everyone is starting with the same state and using the same hash as the first seed, the choices of all committee members as well as the minting node for a block will be same for all.

Block creation by the minting node and broadcast to committee

The minting node is expected to create a block using the below-mentioned transactions.

1. Standard transactions

   The node minting a new block will incorporate as many transactions in its local memory pool as fit a single block - ordered in terms of their timestamp. It also transmits these transactions to the committee members, with the expectation that the committee members either already have these transactions or can update their local memory pool with them if they don't.
2. Block reward transaction

   The block reward transaction in each block creation automatically, with the block creating node as the recipient.
3. Network trust score transaction

   The minting node calls the network trust score manager smart contract with a selected set of receipts from its memory pool, which refer to previous blocks (not necessarily just the latest one). The smart contract consumes the receipts and updates network trust scores. Each block has exactly one transaction like this. It is further described in the 'repercussions of voting' section below.

The block reward and network trust score transactions are not separately signed but are deemed signed by the minting node through its signature on the overall block. This removes the need for validating these transactions by other nodes since these are not transmitted ahead of time anyway.
The selected node also updates its local state. The transaction fees from all included transactions in the block are transferred to the treasury address of Newrl. The treasury uses its balance as a liquidity pool of the tokens allowed for the transaction fees.
The mining node broadcasts the block to the rest of the committee members with its own signature and receipt as follows.

```
{ block_data : {}
  signature : {}
  receipt : {} }
```

Block format

Each block in Newrl has the following format.

block_index	An integer
proof	Set at 0 for regular block and 42 for empty block
Status	Set at 1 for regular block, -1 for empty block created because invalid submission by block creator and -2 for empty block created because of time-out
creator_wallet	The actual creator of that block – sentinel node for empty blocks
expected_miner	Expected creator of the block at that index
committee	Committee of nodes that will signed off on its validity
timestamp	Time in seconds since the UNIX epoch
previous_hash	Hash of the previous block
transactions_hash	Root hash of the transactions
transactions_data	{standard transactions, network_trust_score_update_transaction}

"Transactions data" includes only the transaction id for standard transactions. The actual transaction data is not sent and instead is taken locally by each node using the transaction id in the received block. This avoids any malicious modifications in the transaction by the minting node and hence reduces the need to validate the transaction again by the receiving node.

Block receiving and validation by other nodes in the committee

When a committee member receives a block from the minting node, it first validates the block as shown in the box below.

```
if block_index != current_latest_block_index + 1:
    return (Status = False)
if previous_block_hash != hash(latest_block):
    return (Status = False)
if not valid(block_signature) or signer!= minting_node:
    return (Status = False)
if SHA256_hash(block_data)[0: DIFFICULTY_LEVEL] != "0"*DIFFICULTY_LEVEL:
    return (Status = False)
synchronize(memory_pool, peers)
for transaction_id in transaction_ids:
    if transaction_id not in memory_pool:
        return (Status = False)
if not valid(network_trust_score_transaction):
    return (Status = False)
return (Status = True)
```

This process leads to creation of a receipt, which looks as follows.

{“receipt_data”: {“block_index”:<index>,
“block_hash”: <hash>,
“vote”: 1, -1, -2, 5},
“address”:<address>,
“signature”:<signature>}

Normally, a vote of 1 reflects the True status as per earlier steps above and vote of -1 reflects False status. However, a node is at liberty to send a vote of -2 that indicates that it has insufficient information, or it does not want to vote. This is useful in avoiding the other committee members as well as the wider network waiting for a given member’s vote if it wants to abstain. Allowing it to vote -2 reduces the confusion about whether a node is inactive or abstaining.

Each committee members transmits its receipt to all other committee members and also transmits received receipts to others, in case they haven’t received those from the original sender for whatever reasons. It is expected that at the end of the block_creation_timeout, all live committee members will have a sufficient number of receipts to conclude on block validity. Special empty blocks created by the sentinel node due to time-outs and in lieu of dishonest blocks have a vote 5 indicating their special status.

Using receipts to validate a block

We assume the following variables.

p_i = f(s_i)

v_i = vote of i^{th} committee member

P = number of 1 votes

G = number of -1 votes

Z = number of -2 votes

The following steps are used to validate a block based on its receipts.

$$V^+ = \prod_{i=1}^{P+G} prob_i$$

Where,

$$prob_i = p_i \ (if \ v_i = 1) \ and \ [1 - p_i](if \ v_i = -1)$$

$$V^- = \prod_{i=1}^{P+G} prob_i$$

Where,

$$prob_i = p_i \ (if \ v_i = -1) \ and \ [1 - p_i](if \ v_i = 1)$$

$$Y = C - (P + G + Z)$$

If $Y > 0$, there are pending receipts.

If $V^+ > V^-$, assume all the pending receipts are -1

If $V^+ < V^-$, assume all the pending receipts are 1

Calculate Q+ and Q- as follows.

$$Q^+ = \prod_{i=1}^{P+G+Y} prob_i$$

Where,

$$prob_i = p_i \ (if \ v_i = 1) \ and \ [1 - p_i](if \ v_i = -1)$$

$$Q^- = \prod_{i=1}^{P+G+Y} prob_i$$

Where,

$$prob_i = p_i \ (if \ v_i = -1) \ and \ [1 - p_i](if \ v_i = 1)$$

If $V^+ > V^-$,

If $Q^+ > V^-$, consider the block valid, else consider the outcome indeterminate.

If $V^+ < V^-$

If $Q^- > V^+$, consider the block invalid, else consider the outcome indeterminate.

The above algorithm ensures that even with partial receipts a node can make a decision about the validity of a block. The algorithm also ensures consistency of decision with arrival of new valid receipts. An “indeterminate” decision either changes to valid or invalid or remains indeterminate. A “valid” or “invalid” decision remains the same with new receipts as well.

<u>Committee member action after confirming the validity/invalidity of a block post receipts</u>

Each committee member confirms the status of the block in light of the receipts from other members, as described in the earlier section.

A. Indeterminate status:

If the status is indeterminate for all live members (which is possible if only a minority of the committee members are active) after a time-out limit described below, the committee members mint an empty block (with timestamp of 1 second after that of the previous block) without any signature, create receipts of "1" locally and broadcast it within the committee. Then they move to next step of broadcasting the empty block (validated with an inadequate number of receipts.) In this case, they do not transmit the valid non-empty block created by the minting node.

B. Valid status:

   If the status is valid, the members broadcast the block data, minting node's signature for it and the receipts they have locally to their respective peers.

C. Invalid status:

   If the status is invalid, they broadcast the invalid block with the accompanying receipts. Then, similar to 1 above, they mint an empty block locally (with timestamp of 1 second after that of the previous block) with no signature and broadcast it amongst themselves to create receipts for it. This empty block with adequate receipts is then broadcast.

After this, the committee members update their state locally in case of valid block addition. For empty blocks, there are no state changes other than addition to the blocks table.

Time-outs inside the committee

It may happen that the node everyone considers chosen does not conclude so because it has a wrong or non-updated version of the state. It may also happen that the node chosen is simply inactive.

Also, while a valid block might have been minted, only an insufficient number of committee members may be active.

In the above instances, it is necessary for the committee to conclude that they cannot wait any longer and must move forward with minting of an empty block. This time period is a protocol level constant and is referred to as t_1. This is the time trigger for the indeterminacy referred to in the previous section.

New block verification by rest of the network

After committee members broadcast any block as described above to their respective peers and then onwards to the wider network, the block verification and inclusion/rejection by the rest of the network (i.e. other than the committee) is as follows. The first part is to validate the block using receipts alone. This is done as follows.

1. Validate receipts

    a. Must be from someone who is a committee member
    b. Must be signed correctly
2. Check for the block referred to in the receipt - check if the broadcasted block's hash matches that in the receipt
3. Try to validate the block using the received receipts as described earlier above.
4. If validity status is indeterminate, wait for more receipts.

The validation of block from first principles is same as that described earlier in the context of committee members.

Following are the next steps locally taken for each node upon receiving a block broadcast.

Type of broadcast	Action	Further broadcast?
Standalone valid receipts	Store in memory pool	Yes
Voted invalid block i.e. one with adequate "-1" receipts	Do not act on the block, store receipts and block in memory pool	Yes
Voted valid block i.e. one with adequate "1" receipts	Move to next step of block validity from first principles	Yes if valid
Voted empty block with adequate number of receipts.	Move to next step of block validity from first principles	Yes if valid
Empty block with inadequate number of receipts, within time-out period.	Wait for a time-out period to check if another block broadcast is received.	No
Empty block with signature of sentinel node, after time-out period.	If there has been no valid block with adequate number of receipts up to that point, move to next step of block validity from first principles.	Yes if valid and not ignored
Empty block with inadequate number of receipts, after time-out period.	Query a few randomly selected nodes from the network for a new block. If they send a block with adequate number of receipts, refer to above cases. If they send an empty one with sentinel node signature, refer to the above cases.	No
Non-empty block with correct signature but an inadequate number of receipts, within time-out period	If there is an empty block with adequate number of receipts, already received, ignore this non-empty block. Else, wait to receive remaining receipts. If adequate number of receipts is received, go to appropriate step in the above. Do not act on the block in either case, in terms of appending it to local chain and updating state, if adequate number of receipts not received.	No
Non-empty block with correct signature but an inadequate number of receipts, after the time-out period	Query a few randomly selected nodes for a new block. If a valid non-empty or empty block with adequate number of receipts or an empty block with sentinel node signature is received, go to the appropriate next step from above.	No

No block received at all even after time-out period.	Query a few randomly selected nodes for a new block. If a valid non-empty or empty block with adequate number of receipts or an empty block with sentinel node signature is received, go to the appropriate next step from above.	No

Time-outs in the wider network, outside the committee

There is a network-wide time-out window, represented by t_2 (> t_1 mentioned earlier), which is the outer limit of time for which a node is required to wait for receiving a valid block with adequate receipts from the block creation committee (even if it's empty).

After this time-out, to avoid the network getting stuck (e.g. in the instances of indeterminate empty blocks from a committee), specifically chosen sentinel nodes (typically with the highest trust scores) broadcast an empty block with their signature. If received after the network-wide time-out period and in absence of an adequate-receipt valid block until then, the recipients in the network broadcast this block onwards. This is unlike the indeterminate blocks which are not broadcast onwards as a rule.

This will not create confusion or forking because empty blocks are all the same, irrespective of whether committee sends them or a specific node. The purpose of this step is only to avoid network being stuck for want of a valid block (even if empty) to add. To avoid dishonest nodes from sending empty blocks at random, this action is restricted to a few chosen sentinel nodes which have very high trust scores. In any case, nodes can choose to locally add an empty block as well, after the time-out period. Hence there is no additional information in this action other than a confirmation that this block is meant to be empty. Also, empty blocks sent in addition to valid blocks with adequate signatures are ignored anyway and the sentinel nodes sending such blocks are removed from the sentinel list for subsequent rounds.

Post-validation process by a node

When a receiving node concludes that it is ready to process a received block, as per the previous section's method, it proceeds as follows.

1. It validates the block for its data from first principles, as described earlier. This is not based on receipts but on the actual transactions included in the block. This is same as the validation carried out inside the committee.
2. If the block is found to be invalid despite the receipts suggesting otherwise, the node rejects the block and does not add anything for the time being. It does not broadcast the received

message forward either. Its expectation here is that the full network will arrive at consensus based on the validity of the block anyway and if the node itself has mistakenly considered the received block to be invalid, it can catch up with the rest of the network in subsequent rounds by querying its peers for the updated blocks.

It may yet happen that the committee and minting node were dishonest, and they send an invalid block. In such cases, majority of the honest nodes will not update their local state and keep waiting. The sentinel nodes can send an empty block to get the network restarted.

3. If found valid, it appends the new block to its own chain. It also transmits the whole message onwards to its peers, as stated in the previous section.
4. After a successful update as above, a node also updates its local states according to the transactions in the new block.

In future, the block addition broadcast will also include changes in the state so that receiving nodes can also double check their state updates.

Repercussions of voting

Each process of validation of a block by the committee members results in the creation of a vote receipt by that member about the block being valid or not. This receipt has a reference to a block index and the hash of the block being voted on besides the actual vote and the signature of the voter.

These receipts are broadcast at first by their creator and then onwards like standard transactions by the others in the network i.e. each recipient confirms the validity of the signature and includes in the memory pool as well broadcasts onwards if it is valid. This way, most of the participants in the network have valid receipts of votes by committee members on a given block. While these receipts are used by the committee as well as the rest of the network for validating the current block, they are retained in the memory pool for subsequent processing in network trust score update transaction as follows.

At the time of new block addition, the minting node incorporates as many receipts as possible, with a preference to older receipts, inside the “network trust score update” transaction referred to earlier, calling the network trust score manager smart contract.

The call input reads as follows.

$data = \{receipt_1, receipt_2 \ldots receipt_K\}$

The network score update algorithm uses the receipts with the following logic.

1. Validate the signature on the receipt again. If valid, check the block index referred to in it.
2. Refer to the archive of the right committee members for that block index.
3. If the receipt is not from any of the committee members, update the trust score of the sender of the receipt downward. This is an instance of someone attempting to pose as a committee member without being one.
4. If the receipt is from someone in the correct committee list, check the actual hash of the block index referred to in the receipt from the local chain copy. Follow a different process for an empty block and a non-empty block.
5. For a non-empty block, match this actual hash with the hash of the block referred to in the receipt.
6. If the hashes match, and the vote is “1”, update the trust score of the sender of the receipt upwards as specified. If the vote is “-1”, update the trust score downwards.
7. If the hashes do not match, and the vote is “1”, update the trust score of the sender of the receipt downwards as specified. If the vote is “-1”, update the trust score upwards.
8. In a special case of an empty block, the archives may have one discarded block with inadequate “1” votes and either adequate or inadequate “-1” votes. This block is evaluated for its validity. The receipts that voted for this block as “-1” lead to a trust score reduction of those nodes. The receipts that voted “1” for this block lead to a trust score increase of those nodes. This is the instance of indeterminate valid vote or minority being honest but majority in the committee forcing its way to an empty block.

This process is carried out by the rest of the network as well when that newly added block is considered valid by it. This way, the network uniformly updates the trust scores of all the voting committee members as well as any other nodes that may have tried to pose as committee members. It also updates for attempts at denial of service in the rare instances where the majority of the block creation committee happens to be dishonest, leading to addition of an empty block (as described in 8 above). The catch-up with the current block may vary as per number of receipts pending. However, the protocol is adjusted to incorporate more receipts during the periods of fewer other transactions. (This is analogous to how a human brain updates memories during sleep!)

Network Contribution Trust score

Each node on Newrl network is linked to a specific personId. The transactions and blocks broadcasted by the node are signed with one of the wallets mapped to this personId. The selection probability of each node for minting the next block is proportional to its network trust score.

Assume a network size of *N* nodes. We define a variable Δ = 0.2 by default. It can be reset by the Newrl community.

Every new joining node has a trust score of 10.

The trust score s_i of i^{th} node is modified by the trust score manager, using the receipts provided to it, as follows.

1. Each valid block creation by that node increases its trust score by a value as follows.

   $$s_i = s_i + 10\Delta$$

2. Each invalid block creation by that node decreases its trust score by a value as follows.

   $$s_i = s_i - 100\Delta$$

3. Honest receipts update the trust score as follows.

   $$s_i = s_i + \Delta$$

4. Dishonest receipts update the trust score as follows.

   $$s_i = s_i - 10\Delta$$

The value of s_i is capped at 100 and floored at -100. In state database, the value is stored with 4 decimals, so a score of 25 reads 250000.

The trust score change is asymmetric. For instance, it would take a node just 50 dishonest receipts to get from a score of 100 to 0 while it would take more than 500 honest receipts to climb back up from 0 to 100. In terms of blocks the same numbers are 5 dishonest blocks to go down from 100 to 0 and 50 honest blocks to make the journey back up.

Rehabilitating a node

It is possible that a node misbehaves and gets past the lower bound of 0 trust score to be included in any future committee. This would make it impossible for a node to be rehabilitated if it wanted to prove its honesty.

To deal with this situation, a rehabilitation smart contract can be called by at least *C* persons with trust scores in the top decile of the network and with a collective deposit of at least *K*C* times that of the standard node deposit (K set at 100 to start with). This transaction if picked up by a minting node will reset the trust score of the rehabilitating node at 1. The community can decide to use the deposit submitted here to compensate any loss from malicious behaviour of the rehabilitated node in future.

Byzantine Fault Tolerance and other security considerations in Newrl

Unlike the current public blockchains, the nodes participating in the network of Newrl are not anonymous. Secondly, their behaviour history is retained in the form of the network trust score. More specifically, a node creating a wrong block is immediately recognized before the block is broadcast, by the other members of the block creation committee. Since this committee is randomly chosen, the probability of malicious actors being able to reliably control its outcome is vanishingly small.

Assuming the proportion of dishonest nodes is *d*, even in absence of trust scores, the raw probability of a committee of C parties having a majority of dishonest nodes is given by the following.

$$p_{dishonest\_committee} = d^{(C/2)}$$

For d=0.33 and C=20, this probability is 0.002%.

Within a committee, since trust scores are used to arrive at validation of a block by its members, the trust_score informed votes have a vanishingly small probability of being manipulated by even a majority of dishonest nodes. If the honest nodes with high scores vote against a new block, the dishonest nodes' votes for the block can still be negated.

Even after a block is broadcast by a committee, it is open to audit by others in the network. Since others too additionally validate a broadcasted block and ignore it if found invalid, the attacker's work is not finished even after being able to add an incorrect block through majority control of a committee.

An attempt to vote maliciously against a valid block, with the objective of simply creating hurdles in the regular operations of the network (and prompting addition of an empty block) also gets caught fast enough as empty blocks prompt the network to review the discarded block and update trust scores - punishing majority if the block were valid and minority if it were indeed invalid.

The most reliable defence against malicious actors is that the community of Newrl participants is built organically with gradual addition of persons known to someone or other in the existing network. A new person cannot join the network without being invited by someone who can vouch for them. This promotes honesty without centralizing control of the network.

Lastly, since all wallets are KYC verified, an attacker even after succeeding to steal tokens will not be able to use the stolen tokens for anything without revealing their identity. It is akin to stealing money into one's typical KYC-ed bank account (as against the numbered bank account of the Swiss Banks variety, which are analogous to the current blockchains' addresses).

A Sybil attack is extremely difficult to employ on Newrl, for the following reasons.

a. Each node on Newrl has an associated personId and a KYC wallet. A sybil attacker will need to impersonate several persons along with their valid documents.
b. Each wallet on Newrl is added upon another person/institution vouching for that person. This reduces the speed with which multiple new accounts can be added by a malicious actor.
c. A new node does not have a very high probability of being selected for minting a new block. It has to patiently build good behaviour history over time.
d. Each new block added is still verified by others in the block addition committee which is chosen randomly for each block.

The denial-of-service attack in Newrl is avoided by throttling the number of transactions a node can receive from another node.

Overall, would-be attackers have to patiently build at least moderate trust scores through valid contributions over time and valid KYC at the start, to be even in the reckoning for being able to influence the outcome of any voting. Even then their attacks are thwarted by the multiple checkpoints on validity of transactions, blocks and receipts.

Way forward

Proof of Trust is a stabler consensus protocol than proof of stake and significantly lower on resource wastage than proof of work. More importantly, it incorporates valuable information about node behaviour that is currently ignored. Newrl generalizes the trust scores beyond network management to wider array of transactions amongst real persons. It has several features that enable a more effective execution of the proposed objective of decentralized social finance - including identity, trust network, atomic tokenization, templatized smart contracts and decentralized organizations.

The current blockchains have established the foundational technology of trustless transactions in a decentralized ledger. Newrl aims to be the next-generation blockchain that builds on these foundations to enhance the varying levels of trust that already exists amongst participants in a real economy, using smart contracts that make contract formation and enforcement frictionless and a trust network that deters against wilful misconduct while rewarding honest behaviour.

References

[1] Nakamoto, Satoshi, Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf, retrieved 14th January 2022

[2] Reiff, Nathan, Why Centralized Cryptocurrency Mining Is a Growing Problem, https://www.investopedia.com/investing/why-centralized-crypto-mining-growing-problem/, retrieved 2nd January 2022

[3] Zhang, Shijie, Lee, Jong-Hyouk, Analysis of the main consensus protocols of blockchain, ScienceDirect, ICT Express 6 (2020) 93-97

[4] Buchman, Ethan, Kwon, Jae, Milosevic, Zarko, The latest gossip on BFT consensus, https://arxiv.org/pdf/1807.04938.pdf, September 24, 2018

[5] Putnam, Robert, Social Capital: Measurement and Consequences, OECD, https://www.oecd.org/innovation/research/1825848.pdf, retrieved 31st December 2021

[6] Pawar, Swapnil, Newrl – The Trust Network, February 1, 2022

Appendix 1: Deriving the network trust score of a node

While any two pairs of participants can have a trust score for each other, there is a specific score used for determining the probability of selection for block creation and in validation committee. This is referred to as the network trust score of a node and is stored in the trust score table with the node as person2 and network trust score manager smart contract personId as person1.

At the beginning of each block creation round, there is a specific state that the network has reached a consensus on, based on the information available up to the previous block. This includes the unilateral trust scores given by network participants to each other as well as the network trust score up to the previous round. Through the current block's transactions one or more inter-node trust scores may have changed. Also, there are receipts from previous rounds of block creation and committee-voting by some nodes.

During the block creation, the minting node includes a call to the network trust score manager smart contract with zero or more receipts. This smart contract revises the trust scores to incorporate the effect of changed inter-node trust scores as well as receipts from voting and block creation.
It executes the following steps.

A. The community trust score is derived as follows.
Start with community trust score estimate of 1.0 for all nodes.

$$C_i = 1.0$$

Carry out iterations (ITER times) as follows. ITER is a protocol constant and is set to 100 by default.

1. Start with the node that has the highest trust score in the current state. For equal scores, arrange nodes alphabetically in the order of their personids.
2. For the i^{th} node with scores s_{ji} (score of node *i* in the assessment of node *j*), the community trust score is,

$$C_i = \frac{\sum_{j \neq i} C_j s_{ji}}{\sum_{j \neq i} C_j}$$

3. One iteration is finished when C_i for the last node is updated.

Calculate the scores with 4 decimals of accuracy. After ITER iterations, round the C_i values to two decimals and multiply the same by 100 to arrive at an integer value of the score.

B. This score i.e. C_i is updated through the network activity as described in the main paper, using receipts of voting included in the call to the contract by the minting node. There is no iteration in this step. The update affects only the specific nodes whose receipts are included in the call to the contract.

The updated value for C_i is the network trust score of the node *i* at the end of the given round of block addition.