Distributed consensus networks

Peter P. Rohde*

Blockchains rely on distributed consensus algorithms to decide whether a proposed transaction is valid and should be added to the blockchain. The purpose of consensus is to act as an independent arbiter for transactions, robust against adversarial manipulation. This can be achieved by choosing random subsets of nodes to form consensus sets. In an economy where consensus is the commodity, consensus must be secure, computationally efficient, fast and cheap. Most current blockchains operate in the context of open networks, where algorithms such as proof-of-work are highly inefficient and resource-intensive, presenting long-term scalability issues. Inefficient solutions to allocating consensus sets equates to high transaction costs and slow transaction times. Closed networks of known nodes afford more efficient and robust solutions. We describe a secure distributed algorithm for solving the random subset problem in networks of known nodes, bidding to participate in consensus for which they are rewarded, where the randomness of set allocation cannot be compromised unless all nodes collude. Unlike proof-of-work, the algorithm allocates all nodes to consensus sets, ensuring full resource utilisation, and is highly efficient. While the protocol is synchronous, a staking mechanism ensures self-synchronisation with no dependence on an external reference. The protocol follows self-enforcing where adversarial behaviour only results in self-exclusion. Signature-sets produced by the protocol act as timestamped, cryptographic proofs-of-consensus, a commodity with market-determined value. The protocol is highly strategically robust against collusive adversaries, affording subnetworks defence against denial-of-service and majority takeover.

CONTENTS

* dr.rohde@gmail.com

I. INTRODUCTION

II. NOTATION

$\mathcal{P} = \mathcal{P}_C \cup \mathcal{P}_V$ denotes the pool of all nodes.

$\mathcal{P}_C$ and $\mathcal{P}_V$ denote pools of known consensus and validator nodes.

$\mathcal{B}$ denotes a common broadcast channel, observed by all nodes, into which all nodes may announce messages.

$\mathcal{C}$ denotes a consensus set.

$\mathcal{V}$ denotes a validator set.

III. CONSENSUS

The purpose of consensus is to provide a decentralised mechanism for a set of parties to collectively act as an independent arbiter in judging and signing off on the validity of a statement. In the context of blockchains, this is employed to determine whether a newly submitted block is legitimate and should be added to the blockchain.

In an environment where a minority subset of parties are dishonest and conspiring to form false consensus, the purpose of consensus protocols is to always reflect the will of the majority, independent of the behaviour of dishonest parties. Since dishonest parties are assumed to be

in the minority, this can always be achieved by ensuring that all parties are involved in consensus. However, this is highly resource-intensive and consensus needn't involve all parties. A subset of parties suffices to form consensus if their decision reflects the will of the majority.

Choosing a subset of parties to form consensus there is some probability of dishonest parties forming a false majority by chance. Consensus sets should therefore be chosen to upper-bound this probability, independent of the strategy employed by dishonest parties.

Consensus is formed by majority vote on the validity of some `statement` by members of a consensus set $\mathcal{C} \subseteq \mathcal{P}_C$ chosen from a pool of consensus nodes. The MajorityVote$(\cdot)$ function is then defined probabilistically,

$$\textsc{MajorityVote}(\mathcal{C}, \texttt{statement}) = \begin{cases} \Pr(1 - P_M), & \textsc{MajorityVote}(\mathcal{P}_C, \texttt{statement}) \\ \Pr(P_M), & \neg\textsc{MajorityVote}(\mathcal{P}_C, \texttt{statement}) \end{cases},$$

where P_M is the probability of $\mathcal{C}$ comprising a majority of dishonest parties who are attempting to subvert majority rule.

For $P_M = 0$, where an honest majority is guaranteed, this function is deterministic rather than probabilistic and $\textsc{MajorityVote}(\mathcal{C}, \cdot) = \textsc{MajorityVote}(\mathcal{P}_C, \cdot)$. The purpose of distributed consensus algorithms is to choose consensus sets $\mathcal{C}$ which guarantee operation in this regime.

Algorithm 1 Consensus methods.

```
procedure Consensus(C, statement) → {0, 1}
    return MajorityVote(C, statement)

procedure ConsensusTime(C) → ℝ
    return Median(ReportedTimes(C))

procedure ConsensusParticipants(C, S) → P
    P = {}                                   ▷ Participant set
    for i ∈ S do
        if MajorityVote(C, S_i ∈ P) then
            P ← P ∪ S_i
    return P
```

IV. MODEL

We consider a network of known, trusted nodes, comprising a pool of consensus nodes ($\mathcal{P}_C$) and validator nodes ($\mathcal{P}_V$), with access to a shared, public broadcast channel ($\mathcal{B}$) into which nodes may Announce messages for all to see. The combined pool of consensus and validator nodes, $\mathcal{P} = \mathcal{P}_C \cup \mathcal{P}_V$, is known to and accepted by all nodes.

Nodes bid to participate in the protocol by announcing a salted hash of the previous block header and staking a wager, which is recovered upon completing and remaining in compliance with the protocol. Non-compliance results in loss of stake.

Validator nodes are responsible for forming consensus on which nodes from the pool have placed valid bids to participate, $\bar{\mathcal{P}} \subseteq \mathcal{P}$, and for observing the broadcast channel and timestamping all announcements.

Consensus nodes are allocated to consensus sets via a distributed implementation of the random subset problem.

V. THE RANDOM SUBSET PROBLEM

The random subset problem is to choose a random subset A uniformly from B, where $A \subseteq B$, of which there are,

$$\binom{|B|}{|A|}, \tag{5.1}$$

combinations. If the proportion of parties acting dishonestly and collusively is r, the probability that a random subset of size N contains a majority of dishonest players is,

$$\begin{aligned} P_M(N, r) &= \sum_{n=\lfloor N/2 \rfloor + 1}^{N} \binom{N}{n} r^n (1-r)^{N-n} \\ &= 1 - I_{1-r}(\lfloor N/2 \rfloor + 1, \lfloor N/2 \rfloor + 2) \end{aligned} \tag{5.2}$$

where $I_p(a, b)$ is the regularised incomplete beta function. Choosing subsets uniformly at random ensures the probabilities of set members being dishonest are independently and identically sampled with probability r. Hence, P_M is independent of how dishonest nodes are distributed and what strategies they might employ.

To ensure the integrity of majority voting, we require the likelihood of a minority gaining the ability to form consensus by chance to be upper-bounded by an accepted threshold $\varepsilon \ll 1$, an arbitrary, exponentially small approximation for zero,

$$P_M(N, r) \leq \varepsilon. \tag{5.3}$$

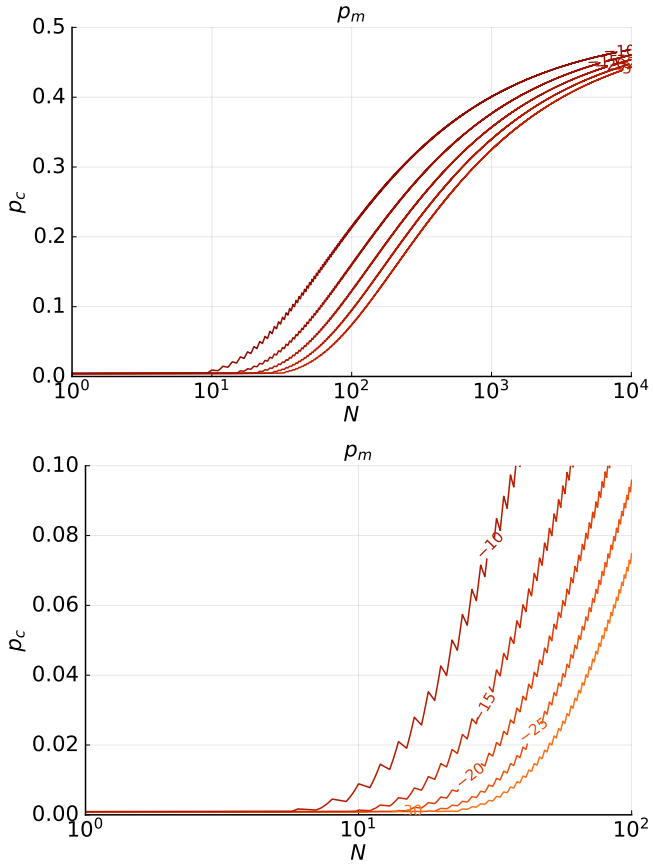

Figure 1: Probability ε of dishonest parties from a random consensus set of size N_c forming a false majority when the proportion of dishonest nodes is r.

We define the consensus set size, N_C, as the smallest set size satisfying this inequality,

$$N_C(r, \varepsilon) = \arg\min_N (P_M(N, r) \leq \varepsilon). \tag{5.4}$$

For the network to maintain ε-confidence in the integrity of majority votes, consensus sets should never be smaller than N_C. Simultaneously, choosing consensus sets larger than N_C is unnecessary and wasteful. We will therefore assume all consensus sets are of size $|\mathcal{C}| = N_C$.

Tradeoff curves between N_C, r and ε are shown in Fig. 1.

A. Centralised algorithm

The random subset problem has a trivial centralised solution, shown in Alg. 2. We assign unique random bitstrings to each node, order them by their random number, and partition the ordered list piecewise into units of length N_C. These partitions form non-intersecting random subsets, each defining an independent consensus set.

The challenge is to securely implement this algorithm in a decentralised environment, such that nodes are unable to compromise the randomness of the assignments of consensus groups.

Algorithm 2 Centralised algorithm for assigning a set of nodes $\mathcal{S}$ to a set of independent random subsets $\{\mathcal{C}\}$ of a given `size`.

procedure RANDOMSUBSETS($\mathcal{S}$, `size`) $\to \{\mathcal{C}\}$
  ▷ *Assign a random number to each node* ◁
  for $i \in \mathcal{S}$ **do**
    $\texttt{random}_i \leftarrow$ RANDOM($\{0,1\}^n$)
  ▷ *Sort nodes by their random number* ◁
  `ordered` = SORT($\mathcal{S}$, `random`)
  ▷ *Partition the list into consensus sets* ◁
  $\{\mathcal{C}\}$ = PARTITION(`ordered`, `size`)
  return $\{\mathcal{C}\}$

B. Proof-of-work

Proof-of-work has been widely employed in blockchains such as Bitcoin as a distributed protocol for choosing random subsets of nodes. Here, nodes compete to find satisfying inputs to hash functions whose outputs satisfy a constraint dictating the likelihood of success. This distributed algorithm effectively asks nodes to find solutions to randomised problems with very low probability of success, $p_{\text{mine}} \ll 1$, such that winners are randomly allocated across nodes in the network.

Since hash functions are pseudo-random and exhibit pre-image resistance, the only viable approach to finding such solutions is via brute-force, repeatedly hashing random bit-strings until a satisfying input is found. Winning nodes are hence allocated at random and cannot be spoofed. The distributed algorithm for proof-of-work is shown in Alg. 3.

Algorithm 3 Random subsets via proof-of-work. Nodes $\mathcal{S}$ hash random bitstrings, salted by a problem instance specified by the previous block `header`, where the per-hash success rate is p_{mine}.

procedure PROOFOFWORK($\mathcal{S}$, `header`, `size`) $\to \mathcal{C}$
  $\mathcal{C} = \{\}$ ▷ *Consensus set*
  while $|\mathcal{C}| <$ `size` **do**
    for $i \in \mathcal{S}$ **do** ▷ *In parallel*
      $\texttt{random}_i =$ RANDOM($\{0,1\}^n$)
      $\texttt{output}_i =$ HASH(`header`|$\texttt{random}_i$)
      if VALID($\texttt{output}_i$) **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{S}_i$
  return $\mathcal{C}$

procedure VALID($x \in \{0,1\}^n$) $\to \{0,1\}$
  if $x/2^n \leq p_{\text{mine}}$ **then**
    return 1
  else
    return 0

Proof-of-work has no requirement that nodes be known

or trusted, providing a very general protocol suited to open networks in which anyone can participate. However, while affording a distributed solution to the random subset problem, this approach is inherently wasteful. The expected mining rate is,

$$R_{\text{mine}} = p_{\text{mine}} \cdot R_{\text{hash}}, \tag{5.5}$$

where R_{hash} is the network's net hash-rate. To inhibit the formation of multiple, simultaneous consensus sets, potentially manifesting itself as forks in the blockchain, proof-of-work systems may algorithmically adjust the difficulty parameter to ensure consistent mining times. To achieve constant mining time, difficulty must scale with cumulative network hash-rate, making the distributed algorithm less efficient and more wasteful as network size grows.

C. Closed networks

An environment comprising a closed network of known nodes affords a more efficient solution to the random subset problem. The key observation is to utilise secure shared randomness to randomly permute the set of nodes and partition them as per Alg. 2. The shared randomness should be secure, in the sense that its randomness is robust against manipulation by dishonest nodes.

1. Hash-based random subsets

Secure shared randomness can be achieved by first requiring all nodes to commit and reveal unique random numbers,

$$H_i = \textsc{Hash}(\texttt{transaction}|\texttt{salt}_i), \tag{5.6}$$

salted hashes of a `transaction` bundle they are bidding to participate in, where $\texttt{salt}_i = \textsc{Random}(\{0,1\}^n)$ are random bit-strings. Once all H_i have been revealed a global key is defined as,

$$K = \textsc{Hash}(\bigcup_i H_i), \tag{5.7}$$

our shared random source. Individual keys are then assigned to each node,

$$K_i = \textsc{Hash}(H_i|K). \tag{5.8}$$

Ordering nodes according to $\{K_i\}$ and partitioning the list assigns them to consensus sets as per Alg. 2. Note that the randomness of the global key, K, cannot be compromised unless all H_i are controlled. So long as a single H_i is honest the assignment of consensus sets will be random.

This approach necessarily assumes a closed network as the global key K cannot be established for an undefined set, nor can an undefined set be ordered.

2. Quantum-random subsets

The randomness in the assignment of subsets is inherited from that of the shared random source. While contemporary hash functions are well-studied and considered cryptographically strong, they are nonetheless deterministic functions and not sources of true randomness. If instead of announcing unique hashes nodes announce quantum random numbers we may define our global key as,

$$K = \bigoplus_i H_i, \tag{5.9}$$

and permuted individual keys as,

$$K_i = H_i \oplus K, \tag{5.10}$$

where $\oplus$ denotes the bitwise XOR operation.

Since uniformly distributed quantum random numbers exhibit maximum entropy, K will be quantum-random if at least one H_i was, requiring all nodes to be compromised if the randomness in set allocation is to be manipulated.

VI. DISTRIBUTED CONSENSUS NETWORKS

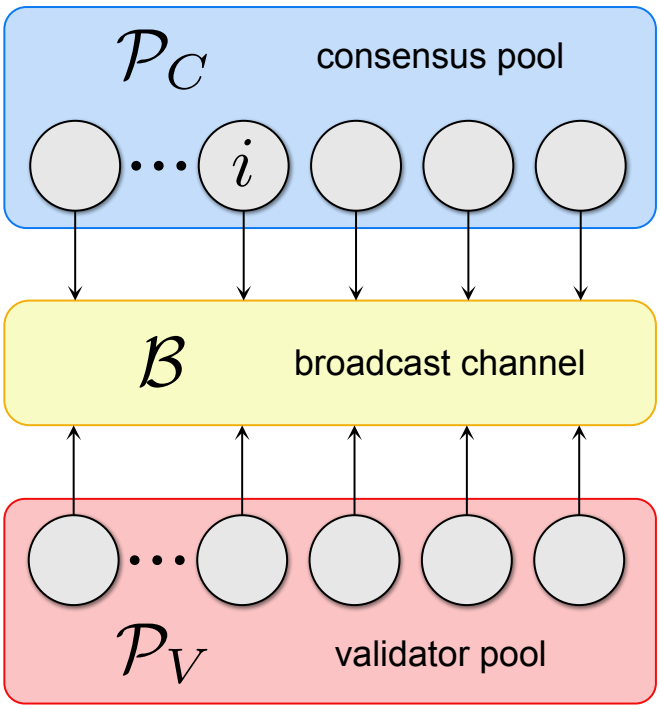

Figure 2

A distributed consensus network comprises a set of known nodes accessing a shared broadcast channel, $\mathcal{B}$. There are two types of nodes, consensus ($\mathcal{C}$) and validator ($\mathcal{V}$) nodes, both bidding to participate in consensus

on a bundle of transactions, for which they are offered a reward upon completion.

All nodes possess the same list of public signatures of every node in the network, defining the consensus and validator pools. Nodes are allocated to consensus sets via distributed implementation of the random subset problem. Consensus nodes $\mathcal{C}$ are allocated to forming consensus on statements presented by the market, while validator nodes $\mathcal{V}$ are responsible for enforcing compliance of consensus nodes.

To incentivise compliance with the protocol, nodes initially stake a deposit to participate, returned upon completing the protocol.

A. Communications primitives

The communications primitives our model relies upon are shown in Alg. 4.

Algorithm 4 Communications primitives, where $\mathcal{B}$ denotes the shared broadcast channel.

procedure ANNOUNCE(node, statement)
  message = SIGN(node, statement, LOCALTIME(node))
  $\mathcal{B} \leftarrow \mathcal{B} \cup$ message

procedure COMMIT(node, statement)
  salt = RANDOM($\{0,1\}^n$)
  message = HASH(statement|salt)
  ANNOUNCE(node, message)

procedure REVEAL(node, statement)
  ANNOUNCE(node, salt(statement))

B. Protocol

A synchronous protocol for solving the random subset problem is shown in Alg. 5, where numbers indicate the synchronous steps.

Nodes must remain compliant with the protocol to secure return of their stake. This requires nodes to be in agreement with the majority on all votes. For validator nodes this additionally requires their timestamps of messages in the broadcast channel be consistent with consensus time, where all validator nodes are required to timestamp all messages in the broadcast channel.

The consensus time for a given message is taken to be the median of all reported timestamps. The median exhibits the property that if a majority of timestamps are within δ of the median, no action by an adversarial minority is able to undermine this. Hence, consensus time is dictated by the majority. In a setting where non-compliance is penalised, this incentivises consensus time towards accuracy, allowing the protocol to self-synchronise, independent of an external time reference.

Algorithm 5 Distributed synchronous protocol for random subsets, where numbers denote synchronised steps.

procedure CONSENSUSSETS($\mathcal{P}_C$, $\mathcal{P}_V$, header) $\rightarrow \{\mathcal{C}\}$
  ▷ *1. Nodes bid to participate* ◁
  for $i \in \mathcal{P}_C \cup \mathcal{P}_V$ **do**
    ▷ *Announce salted hash of* header ◁
    $H_i \leftarrow$ COMMIT(i, header)
  ▷ *2. Validators commit recognised bids* ◁
  for $j \in \mathcal{S}_V$ **do**
    $B_j \leftarrow$ COMMIT(j, RECOGNISEDBIDS(j))
  ▷ *3. Validators form consensus on participants* ◁
  $A \leftarrow$ CONSENSUSPARTICIPANTS($\mathcal{P}_V, \mathcal{P}_C \cup \mathcal{P}_V$)
  ▷ *4. Assign consensus sets* ◁
  $K =$ HASH($\bigcup_{i \in A} H_i$) ▷ *Shared randomness*
  for $i \in A$ **do**
    $K_i \leftarrow$ HASH($H_i|K$) ▷ *Randomised local keys*
  sorted $\leftarrow$ SORT($A, \{K_i\}$) ▷ *Sort nodes by local keys*
  $\mathcal{C} \leftarrow$ PARTITION(sorted, N_C) ▷ *Partition into sets*
  return $\mathcal{C}$

Algorithm 6 Compliance checking for consensus and validator nodes.

procedure COMPLIANCE($\mathcal{S}$) $\rightarrow \tilde{\mathcal{S}}$
  $\tilde{\mathcal{V}} \leftarrow \{\}$ ▷ *Compliant validators*
  for $j \in \mathcal{V}$ **do**
    compliant $\leftarrow$ true
    ▷ *Validators must not ignore bidders* ◁
    $\mathcal{C} \leftarrow$ CONSENSUSPARTICIPANTS($\mathcal{V}$)
    $\mathcal{C}_j \leftarrow$ RECOGNISEDPARTICIPANTS(j)
    if $\mathcal{C} \neq \mathcal{C}_j$ **then**
      compliant $\leftarrow$ false
    ▷ *Validators must conform with consensus time* ◁
    for message $\in$ ALLMESSAGES **do**
      $t =$ TIMESTAMP(j, message)
      if $|t -$ CONSENSUSTIME($\mathcal{V}$, message)$| \geq \delta$ **then**
        compliant $\leftarrow$ false
    if compliant **then**
      $\tilde{\mathcal{V}} \leftarrow \tilde{\mathcal{V}} \cup j$
  return $\tilde{\mathcal{V}}$

VII. DISTRIBUTED QUANTUM CONSENSUS NETWORKS

* Same as above, but now the reported measurement outcomes are utilised as the random source, enabling quantum-random subsets.

A. Interactive proofs of quantumness

An interactive proof of quantumness () comprises two parties, a *prover* and a *verifier*, where the goal is for the prover to prove to the verifier that they have honestly executed a quantum implementation of some function $f(\cdot)$.

The verifier has only classical resources and both parties may classically communicate.

While such protocols are not known in general for arbitrary $f(\cdot)$, they have been described in the context of a restricted class of functions known as trapdoor claw-free functions.

1. Trapdoor claw-free functions

Trapdoor claw-free functions (TCF) are a class of cryptographic, 2-to-1, one-way functions,

$$f_{\mathcal{I}}(x) \to w, \tag{7.1}$$

which are classically efficient to evaluate in the forward direction, but for which it is hard to find simultaneously satisfying inputs $\{x_0, x_1\}$ (the 'claw') mapping to the same output,

$$w = f(x_0) = f(x_1), \tag{7.2}$$

where $x \in \{0,1\}^n$ and $w \in \{0,1\}^{n-1}$ are bit-strings.

Here, $\mathcal{I}$ denotes a problem instance derived from a secret (the trapdoor). If the secret is known, finding claws $\{x_0, x_1\}$ is classically efficient for any w. Since $f(\cdot)$ is easy to evaluate in the forward direction, verifying solutions is classically efficient, and the problem of claw-finding by definition resides in the complexity class **NP**.

2. The LWE problem

A candidate TCF is the lattice-based learning with errors (LWE) problem (). This problem is believed to be post-quantum, where the associated claw-finding problem lies outside of **BQP**, the class of problems efficiently solvable by quantum computers.

For matrix $A \in \mathbb{Z}_q^{m\times n}$ and vectors $x, y, s, e \in \{0,1\}^n$ related by,

$$y = A \cdot s + e, \tag{7.3}$$

under modulo q arithmetic, a TCF may be constructed as,

$$f_{\mathcal{I}}(b, x_b) = \lfloor A \cdot x + b \cdot y \rceil, \tag{7.4}$$

where $b = \{0,1\}$ is a single bit and claws are related by,

$$x_0 = x_1 + s. \tag{7.5}$$

Here, $\mathcal{I} = \{A, y\}$ specifies the problem instance derived from the secret trapdoor $\mathcal{T} = \{s, e\}$ secretly held by the verifier, enabling efficient classical claw-finding and verification if known.

Since $f(x) \to w$ is classically efficient to evaluate in the forward direction, it is easy to find a w for which a single satisfying input x is known. The challenge lies in finding simultaneously satisfying pairs of inputs, believed to be hard for both classical and quantum computers.

3. Protocol

Taking a cryptographic TCF function, $f_{\mathcal{I}}(x) \to w$, an interactive proof of quantumness may be implemented as follows:

1. The verifier specifies a problem instance $\mathcal{I}$, without revealing the associated secret $\mathcal{T}$ from which it was derived.

2. The prover prepares a uniform superposition of all length-n bit-strings x using Hadamard gates,
$$|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle . \tag{7.6}$$

3. Evaluating $f_{\mathcal{I}}(\cdot)$ on the input x register into an output register yields,
$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle\, |f(x)\rangle . \tag{7.7}$$

4. The prover measures the output register, collapsing the state onto,
$$|\psi\rangle = \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle)\, |w\rangle , \tag{7.8}$$
for random w, which is communicated to the verifier. Since the input space x is a uniform superposition and $f(\cdot)$ is a 2-to-1 function, w is also uniform.

5. The verifier specifies a random measurement basis in which the prover should measure qubits in the x register, where $b = \{0,1\}$ correspond to the Pauli-Z and X bases respectively.

6. When measuring in the Z basis, the prover randomly measures either $m = x_0$ or $m = x_1$, easily verified by directly evaluating $f(\cdot)$ and comparing with the prover's previously reported w. When measuring in the X basis verification suceeds if,
$$m \cdot x_0 = m \cdot x_1. \tag{7.9}$$

7. The above is repeated for some constant number of rounds, randomising the measurement basis b at every round.

The key observation is that since X and Z measurements do not commute, it is not possible for the prover to know both measurement outcomes simultaneously and therefore must measure in accordance with the verifier's stated measurement basis to pass verification of a single round. While a single round can be classically spoofed if the measurement basis b is known in advance of announcing w, if unknown, b can only be guessed with a probability of $1/2$. Upon repetition, the probability of correctly guessing all measurement bases scales as $1/2^{N_{\text{rounds}}}$ for N_{rounds} rounds, ensuring asymptotic confidence in the honesty of the prover.

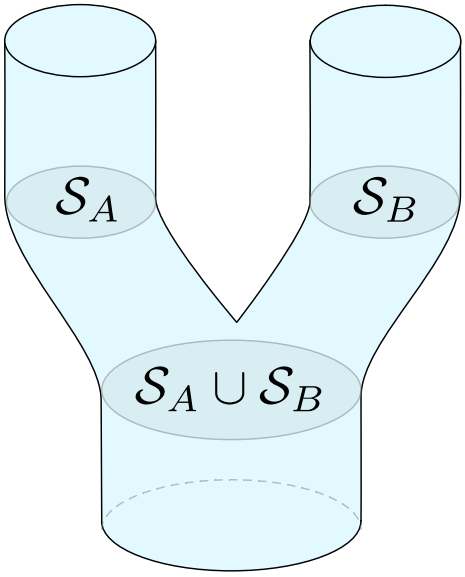

Figure 3: When the network $\mathcal{S}_A \cup \mathcal{S}_B$ supporting a blockchain segregates into two non-interacting networks, $\mathcal{S}_A$ and $\mathcal{S}_B$, a fork is created, forming two unique, legitimate blockchains, one associated with each network.

VIII. ECONOMICS

IX. BLOCKCHAINS

Blockchains are protocol-level applications for consensus, following their own rules on what consensus in formed on.

A. Hard forks

X. APPLICATIONS

XI. CONSENSUS HIERARCHIES

Consider a consensus set $\mathcal{C}_i$ of size $|\mathcal{C}_i| = N_c(r_i)$. This consensus set assumes a ratio of dishonest nodes r_i. Nodes in $\mathcal{C}_i$ may delegate their votes to consensus sets subnetworks, $\mathcal{C}_{i,j}$ where the ratio of dishonest parties is $r_{i,j}$. In this instance, we refer to $\mathcal{C}_i$ as a *quasi-node*, an unphysical node abstractly representing a delegated consensus outcome. To afford the required r_i, subnetworks comprising less trusted nodes, $r_{i,j} < r_i$ may employ larger $N_c(r_j)$.

XII. RESOURCE CONSUMPTION

Distributed consensus networks are highly resource-efficient, as all nodes participate in consensus, and the allocation of consensus sets is.

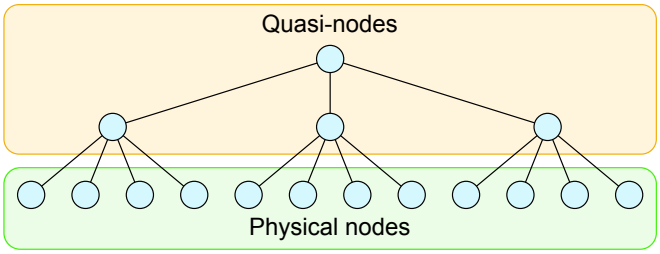

Figure 4

XIII. STRATEGIC CONSIDERATIONS

XIV. RESULTS

XV. CONCLUSION

REFERENCES