Distributed consensus networks

Peter Rohde

BTQ & Macquarie University

Consensus (voting)

Decision outcomes determined by majority vote.

Consensus-time given by the median (robust against minority manipulation).

Theorem: Let T be the set all reported times and $M \subseteq T$ be any majority of T. If δ is the smallest number satisfying,

$$|M_i - \texttt{median}(T)| \leq \delta,$$

δ is independent of the minority.

```
# Consensus on binary decision
def consensus(S,id):
    return S.majority_vote(id)

# Consensus on timestamps
def consensus_time(S.id):
    times = S.reported_times(id)
    return median(times)

# Consensus on set of valid bidders
def consensus_bids(S):
    P = []
    for node in S:
        if consensus(valid_bid(node)):
            P.append(node)
    return P
```

Distributed consensus

Honestly signing transactions in a distributed environment.

Consensus sets: random subsets of participants from a pool of bidders notarise transactions:

$$\mathcal{C} \subseteq \mathcal{P}.$$

Proportion of dishonest parties (conspiratorial adversaries):

$$r < 1/2.$$

Must ensure consensus sets vote to reflect majority:

$$r_{\mathcal{C}} < 1/2.$$

Random subset problem

Choose a random subset $\mathcal{C} \subseteq \mathcal{P}$.

Probability of false-majority with $N = |\mathcal{C}|$ parties:

$$P_M(N, r) = \sum_{n=\lfloor N/2 \rfloor + 1}^{N} \binom{N}{n} r^n (1-r)^{N-n}.$$

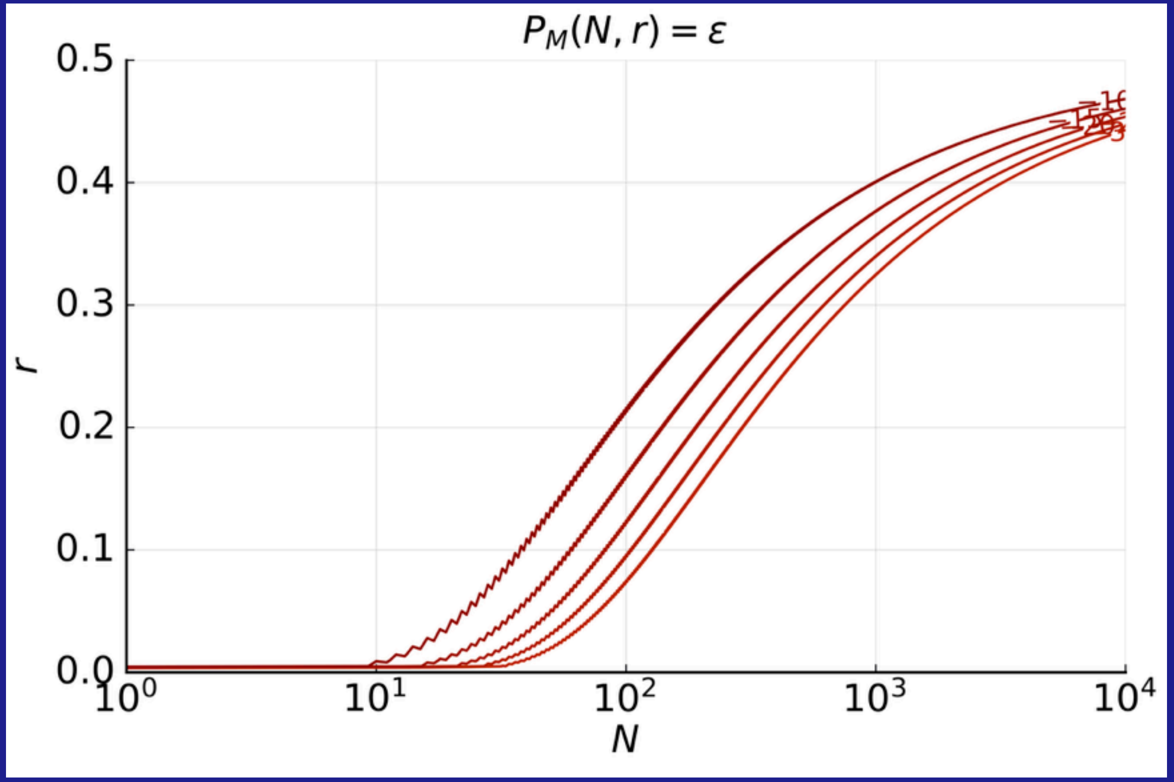
Randomisation ensures r is node-independent and P_M is strategy-independent.

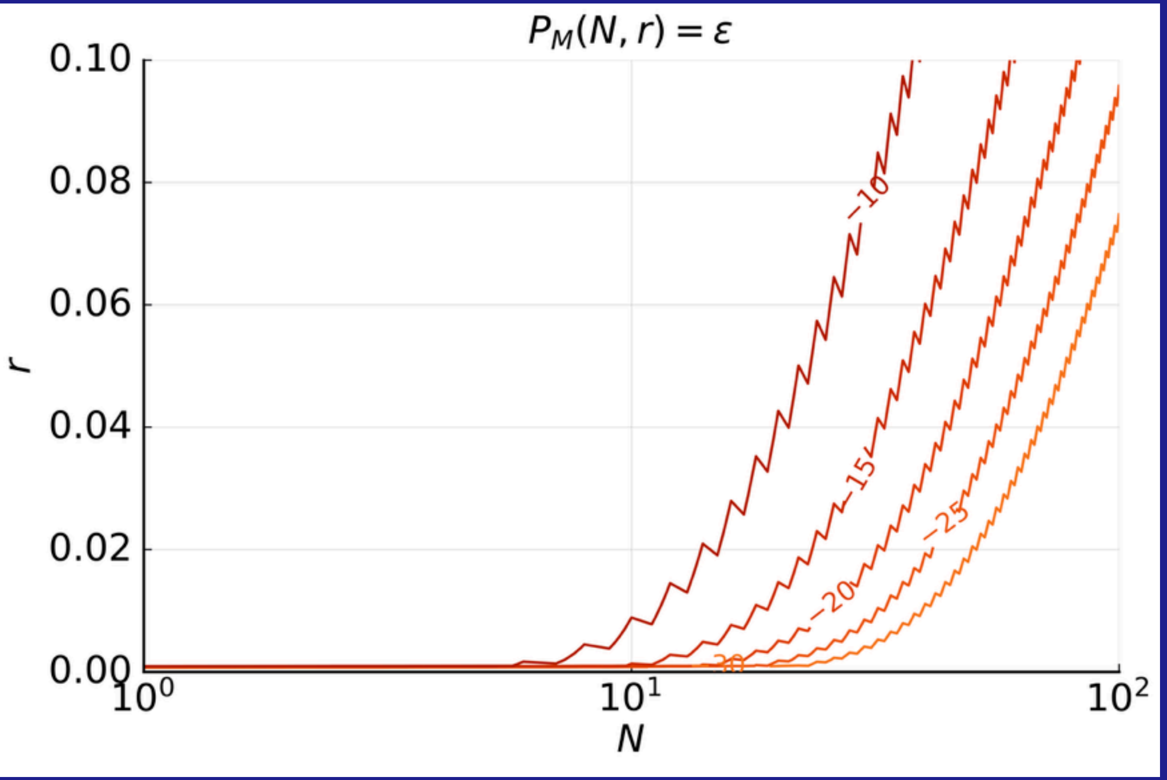
We require:

$$P_M(N, r) \leq \varepsilon \ (\text{where } \varepsilon \ll 1).$$

Requires consensus set size:

$$N_C = \arg\min_N (P_M(N, r) \leq \varepsilon).$$





Random subset algorithm

Simple centralised algorithm for closed sets.

Partitions set $\mathcal{S}$ into a set of independent random consensus sets $\{\mathcal{C}_i\}$ of size N:

$$\mathcal{C}_i \subseteq \mathcal{S}, \ |\mathcal{C}_i| = N,$$

$$\mathcal{C}_i \cap \mathcal{C}_j = 0 \ \ (i \neq j), \ \cup_i \mathcal{C}_i = \mathcal{S}.$$

Equivalent to random permutation & partitioning of ordered set:

$$\pi \in S_n,$$

$$\mathcal{S}' = \pi \cdot \mathcal{S}.$$

```
# Random subsets
def random_subsets(S,N):
    for node in S:
        node.key = random()
    C = S.sort_by(key).partition(N)
    return C

# Distributed random subsets
def dist_random_subsets(S,N):
    for node in S:
        node.key = dist_random(S,node)
        broadcast(node.key)
    C = S.sort_by(key).partition(N)
    return C
```

Secure shared randomness

Collectively established by distributed algorithm.

Robust against manipulation.

Maps to random node permutation.

Proof-of-work

Open networks of unknown nodes.

Asynchronous algorithm.

For transaction `id`, find inputs $x \in \{0,1\}^n$ satisfying:

$$\texttt{hash}(x|\texttt{id}) \le c.$$

Optimal to randomly choose x due to hash function pre-image resistance & uniqueness requirement.

Success (mining) probability for hash length n:

$$p_{\text{mine}} = c/2^n.$$

```
# Random subsets by proof-of-work
def proof_of_work(S,N,id):
    C = [] # Consensus set
    while len(C)<N:
        for node in S: # Parallel
            # Random winners
            if hash(id,random()) < c:
                C.append(node)
    return C
```

Inefficiency

Mining rate for network hash-rate R_{hash}:

$$R_{\text{mine}} = p_{\text{mine}} \cdot R_{\text{hash}}.$$

Maintaining constant mining rate (to prevent double-mining the same transaction) implies efficiency scales inversely with network size:

$$p_{\text{waste}} = 1 - O\left(\frac{1}{n}\right).$$

Almost all computational resources are wasted (asymptotically perfectly inefficient).

Reflected in energy consumption & transaction cost.

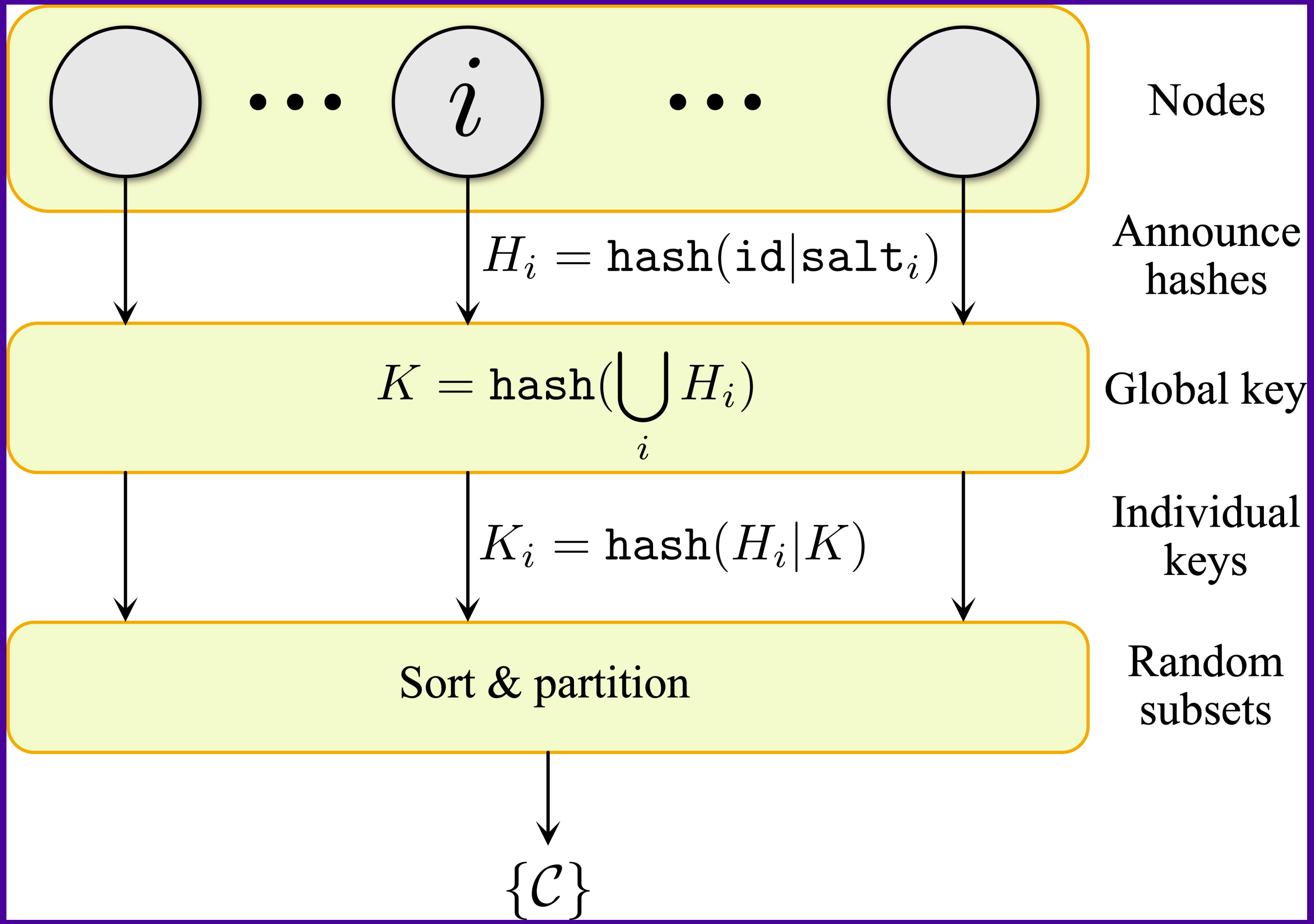
Closed networks

Network nodes agreed upon, known & identifiable.

Utilise secure, shared random numbers.

Acts as global permutation.

Remains random if a single player is honest.




```
# Distributed hash-based random number
def dist_random(S,node,id):
    # Nodes announce salted hashes of id
    for node in S: # Parallel
        salt = random() # Must be unique
        H[node] = hash(id,salt)
        node.announce(H[node])

    # Global key hashes them together
    K = hash(H.join())

    # Individual keys
    node.key = hash(K,H[node])
```

```
# Distributed quantum random number
def dist_quant_random(S,node,id):
    # Nodes announce quantum random numbers
    for node in S: # Parallel
        H[node] = quantum_random(id)
        node.announce(H[node])

    # Global key XORs them together
    K = XOR(H)

    # Map global key to node permutation
    perm = to_permutation(K)

    # Individual keys
    node.key = H[perm[node]]
```

Distributed consensus networks

Floating market of known nodes bid to participate in consensus.

All messages signed & broadcast.

Self-incentivised towards honesty.

Non-compliance & dishonesty only result in self-elimination.

```
# Communication primitives

def announce(statement):
    message = sign(statement)
    broadcast(message)

def commit(statement):
    salt = random()
    H = hash(statement,salt)
    # 1. Initally commit hash
    announce(H)
    # 2. Later reveal pre-image
    announce(statement,salt)

def timestamp(statement):
    time = local_time()
    commit(statement,time)
```

Synchronous protocol

All nodes announce salted hashes & place a stake.

Nodes form consensus on who placed valid bids (participants).

Establishes secure global key.

```
def proof_of_consensus(S,N,id):
    # Nodes bid to participate
    for node in S: # Parallel
        if node.bidding:
            salt = random()
            H[node] = hash(id,salt)
            node.announce(H[node])

    # Consensus on participants
    P = consensus_bids(S)

    # Assign consensus sets
    C = dist_random_subsets(P,N)

    # Consensus on transactions
    for c in C: # Parallel
        c.consensus(id[c])

    # Compliant nodes
    Q = compliance[S]
```

Compliance

All steps in the protocol must be followed.

All votes must agree with the majority.

All timestamps must be within δ of consensus-time (enforces self-synchronisation):

$$|t_i - \texttt{median}(\{t\})| \leq \delta.$$

Compliance revealed retrospectively from $\mathcal{B}$.

```
def compliance(node):
    compliant = true

    # Must agree with majority votes
    for v in votes:
        if node[v] != consensus[v]:
            compliant = false

    # Must agree with consensus time
    for t in timestamps:
        if abs(t[node]-consensus_time(t))>delta:
            compliant = false

    return compliant
```

Proof-of-consensus

Parameterised, timestamped, cryptographic proof that signatories form a random subset of a network & satisfy consensus set criteria:

$$\mathcal{C} \leftarrow \mathrm{RandomSubset}(\mathcal{P}, N), \\ P_M(|\mathcal{C}|, r) \leq \varepsilon.$$

Any complete & compliant subset of announcements acts as a proof system for $\mathcal{C}$:

$$\mathscr{P}(\mathcal{C}) \subseteq \mathcal{B}.$$

Proofs not unique since majorities aren't unique.

But are equivalent (prove the same thing):

$$\mathscr{P}(\mathcal{C}) \cong \mathscr{P}'(\mathcal{C}).$$

Economics

Proof-of-consensus is the commodity.

Nodes place deposit to participate, returned if compliant.

Compliant nodes receive a transaction fee, initially staked the transaction.

Proof-of-consensus has market value. Ideally should be cheap.

Contrast proof-of-work: Cost dominated by inefficiency not utility.

A universal resource

Proof-of-consensus is a cryptographic primitive.

File systems: Universal format for timestamping documents.

Global consensus time: High-speed, low-latency network with small ε.

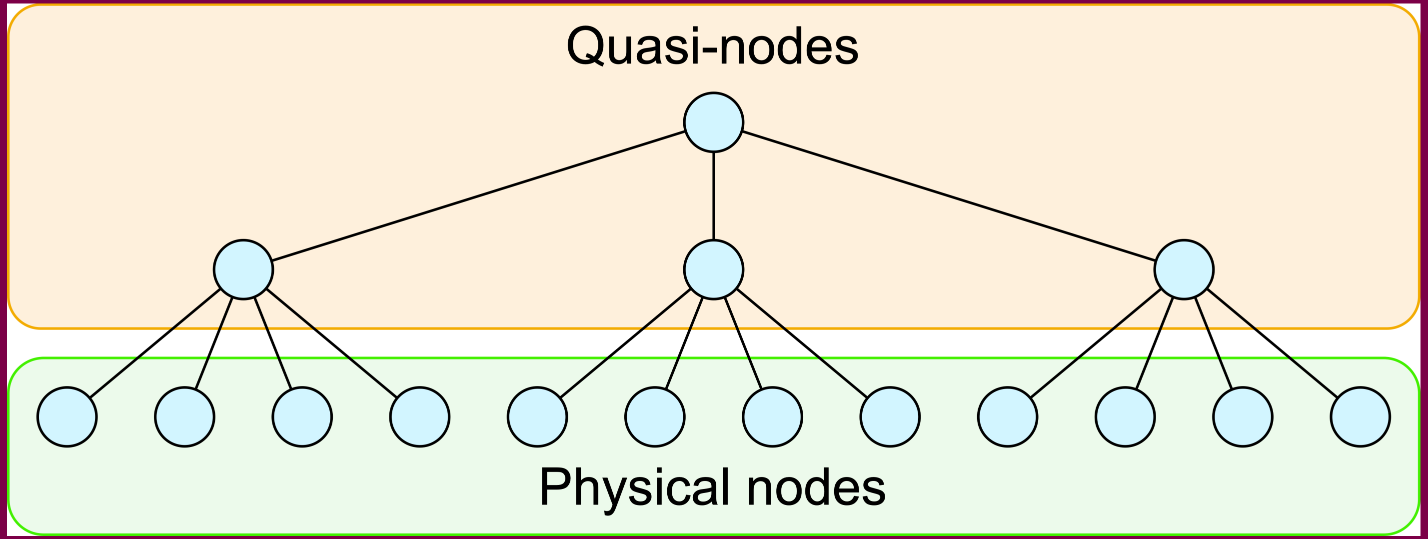
Blockchains

Protocol-layer application for proof-of-consensus.

Rules define implementation & use.

Different blockchains can rely on different consensus networks.

Network architecture



Hierarchical representation of physical nodes & quasi-nodes.

Quasi-nodes represent the outcome of consensus delegated to another network or consensus set.

The r of a quasi-node inherits the delegate's P_M.

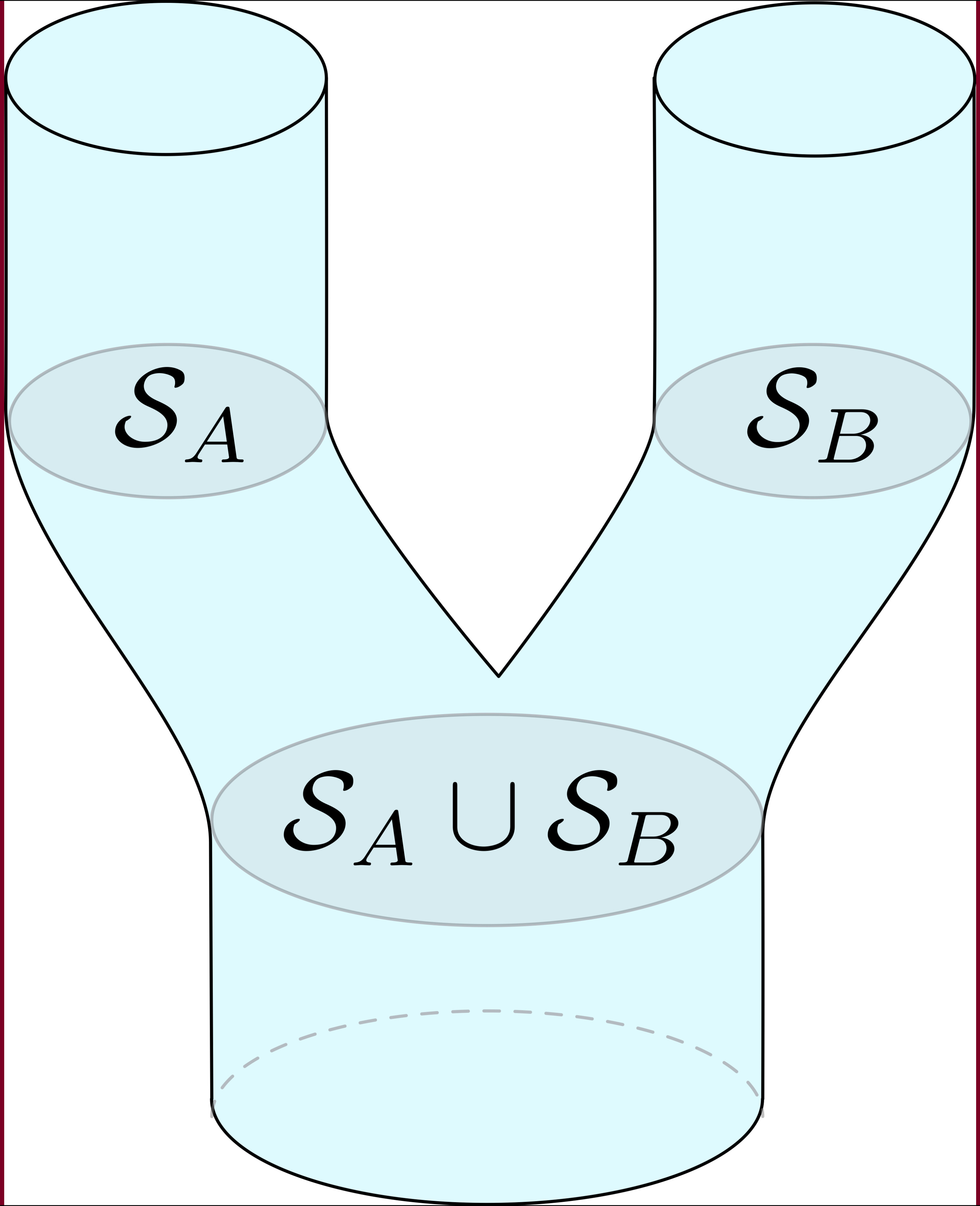
Only nodes at the base of the hierarchy are physical.

Strategic considerations

Compliance & voting of all nodes is public, revealing honest & dishonest subsets.

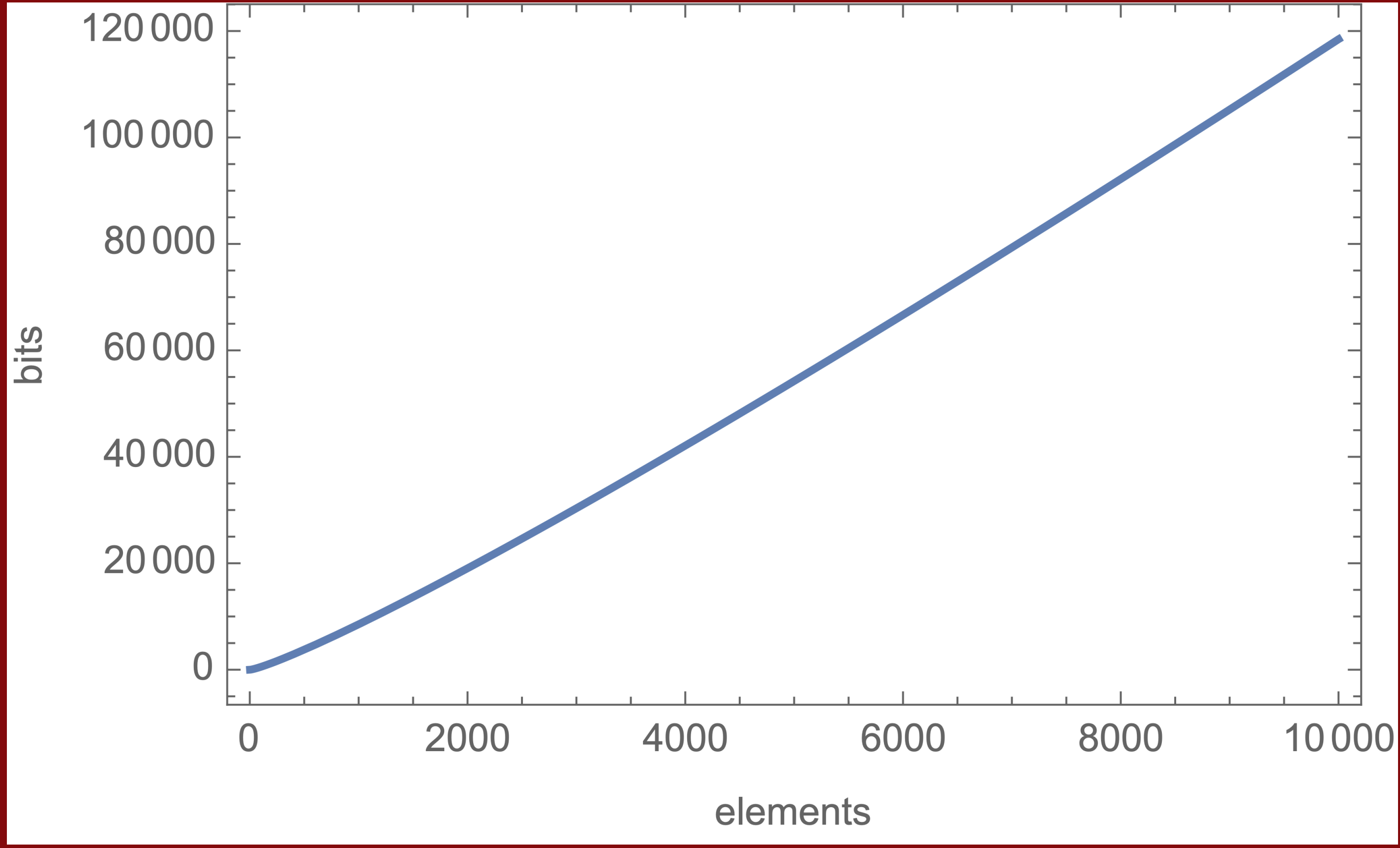
Strategically forking provides a defence against hostile takeover.

Subnets may form trusted groups with the ability to self-exclude and fork as a defence against DoS or hostile takeover.



Distributed quantum consensus networks

Quantum-random subsets.



Using quantum-random numbers instead of hashes H_i establishes a quantum-random global key:

$$K = \bigoplus_i H_i.$$

Quantum-random numbers are maximum entropy.

XOR operation cannot reduce entropy.

K is quantum-random if at least one H_i was.

Map global key to a permutation:

$$K \cong \pi \in S_n.$$

Individual keys are permuted by K:

$$K_i = H_{\pi_i}.$$

Trapdoor claw-free (TCF) functions

2-to-1 function for problem instance $\mathcal{I}$:

$$f_{\mathcal{I}}(x) \to w.$$

For every w there is exactly one pair of simultaneously satisfying inputs $\{x_0, x_1\}$:

$$f_{\mathcal{I}}(x_0) = f_{\mathcal{I}}(x_1) = w.$$

A *claw* is a valid $\{w, x_0, x_1\}$.

Claws (i.e collisions) are hard to find from $\mathcal{I}$.

Easy to find if the secret trapdoor $\mathcal{T}$ is known.

Incomplete claws $\{w, x_0\}$ are easy to find (just evaluate the function).

Learning with errors (LWE) problem

A lattice-based TCF believed to be post-quantum.

Problem instance:

$$\mathcal{I} = \{A, y\},\ A \in \mathbb{Z}_q^{m\times n},\ y \in \{0,1\}^n.$$

Secret trapdoor:

$$\mathcal{T} = \{s, e\},\ s, e \in \{0,1\}^n.$$

Trapdoor claw-free function:

$$y = A \cdot s + e,$$

$$f_{\mathcal{I}}(b, x_b) = \lfloor A \cdot x + b \cdot y \rceil,\ b \in \{0,1\}.$$

Claws related by:

$$x_0 = x_1 + s.$$

Interactive proofs of quantumness

A quantum prover proves to a classical verifier that they have implemented a quantum computation.

Requires only classical communication.

Prover prepares uniform superposition of bit-strings:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2}} \sum_{x \in \{0,1\}^n} |x\rangle.$$

Prover evaluates TCF function into a register:

$$H^{\otimes n}|0\rangle^{\otimes n} = \frac{1}{\sqrt{2}} \sum_{x \in \{0,1\}^n} |x\rangle|f(x)\rangle.$$

Prover measures register to obtain w (uniformly distributed QRN):

$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle)|w\rangle.$$

Verifier specifies a random measurement basis $b \in \{0,1\}$ (Pauli Z or X bases).

Prover measures x register in b basis to obtain measurement m.

Verification:

$$\begin{aligned} b = 0 :&\ \ m \in \{x_0, x_1\}, \\ b = 1 :&\ \ m \cdot x_0 = m \cdot x_1. \end{aligned}$$

Repeat constant number of times for random b.

Quantum key distribution

Secure shared randomness using quantum communication.

Open question?

How do I uniquely associate a quantum random number derived from a QKD link with an id?

Summary

Distributed consensus networks.

Proof-of-consensus is a cryptographic primitive & economic commodity with market value.

High speed, efficient, low cost.

Proof-of-work artificially prices consensus via its algorithmic inefficiency.

Randomness of subsets robust against manipulation.

Strategically robust. Affords defence against hostile takeover (malicious majority).

Quantum-random subsets via interactive proofs of quantumness.