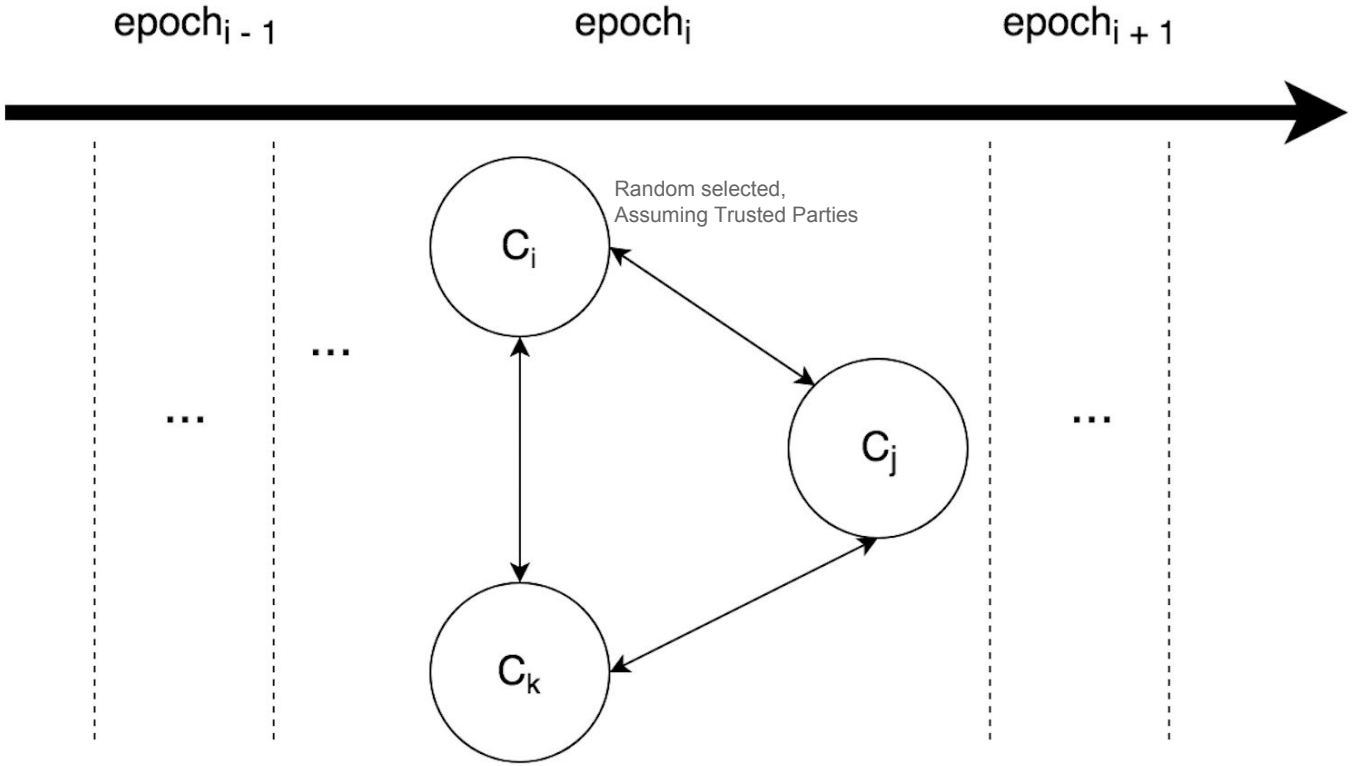
RapidChain Whitepaper Sharing

Shu Dong
09/23/2018

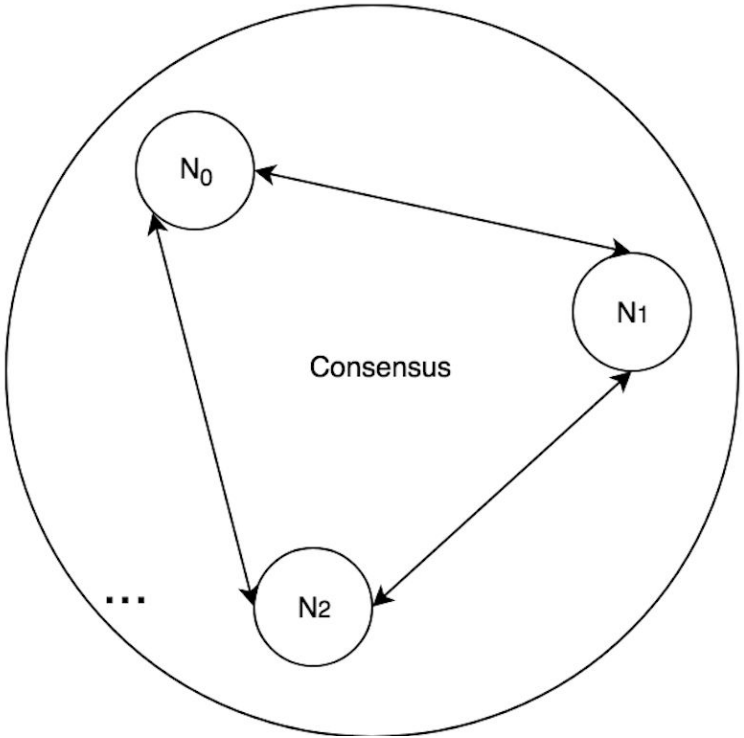
Review: Committee-Based Sharding

- Elastico
- Omniledger
- RapidChain

Review: Committee-Based Sharding



Review: Committee-Based Sharding



Review: Committee-Based Sharding

- Bootstrap
- Consensus
  - Intra-Committee
  - Inter-Committee: cross-shard
- Reconfiguration

Review: Committee-Based Sharding

Q: How to initiate the sharding assignment in a reliable way?

Q: How to maintain a consistent view of the ledger?

Q: How to keep the network safe?

Review: Elasticos

In every consensus epoch, each participant solves a PoW puzzle based on an epoch randomness obtained from the last state of the blockchain. The PoW's least-significant bits are used to determine the committees which coordinate with each other to process transaction

Drawbacks:

1. Elastico requires all parties to re-establish their identities (i.e., solve PoWs) and re-build all committees in "every" epoch.
2. In practice, Elastico requires a small committee size (about 100 parties) to limit the overhead of running PBFT in each committee. Unfortunately, this increases the failure probability of the protocol significantly(can be 0.97 after only six epochs)
3. The randomness used in each epoch of Elastico can be biased by an adversary
4. Elastico requires a trusted setup for generating an initial common randomness that is revealed to all parties at the same time.
5. It still has to broadcast all blocks to all parties and requires every party to store the entire ledger.
6. Finally, Elastico can only tolerate up to a 1/4 fraction faulty parties even with a high failure probability. Elastico requires this low resiliency bound to allow practical committee sizes.

Review: OmniLedger

1. The protocol runs a global re-configuration protocol at every epoch (about once a day) to allow new participants to join the protocol.
2. The consensus protocol assumes partially-synchronous channels to achieve fast consensus using a variant of ByzCoin, where the epoch randomness is further used to divide a committee into smaller groups.

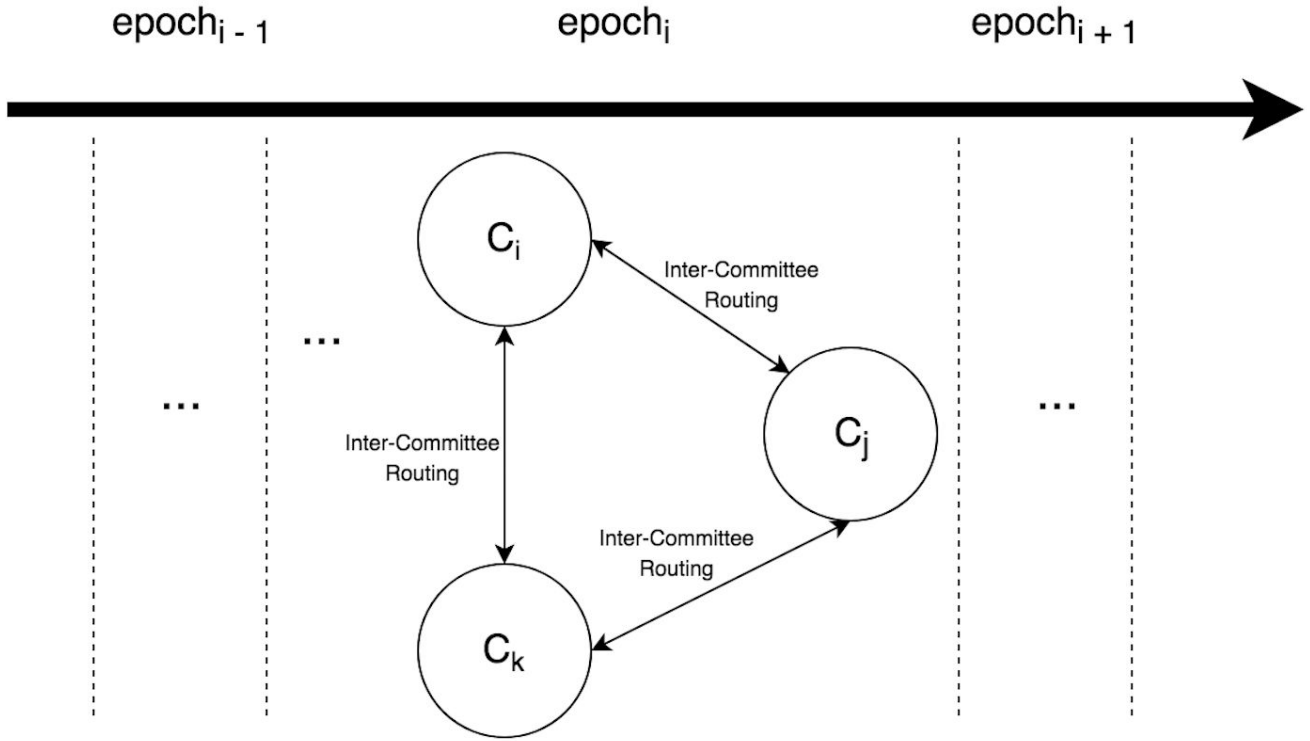
Drawbacks:

1. OmniLedger can only tolerate t < n/4 corruptions. In fact, the protocol can only achieve low latency (less than 10 seconds) when t < n/8.
2. OmniLedger's consensus protocol requires O (n) per-node communication
3. OmniLedger requires a trusted setup to generate an initial unpredictable configuration to "seed" the VRF in the first epoch. (RandHound)
4. OmniLedger requires the user to participate actively in cross-shard transactions which is often a strong assumption for typically light-weight users.
5. OmniLedger seems vulnerable to denial-of-service (DoS) attacks by a malicious user who can lock arbitrary transactions leveraging the atomic cross-shard protocol
6. When t < n/4, OmniLedger can achieve a high throughput (i.e., more than 500 tx/sec) only when an optimistic trust-but-verify approach is used to trade-off between throughput and transaction confirmation latency.

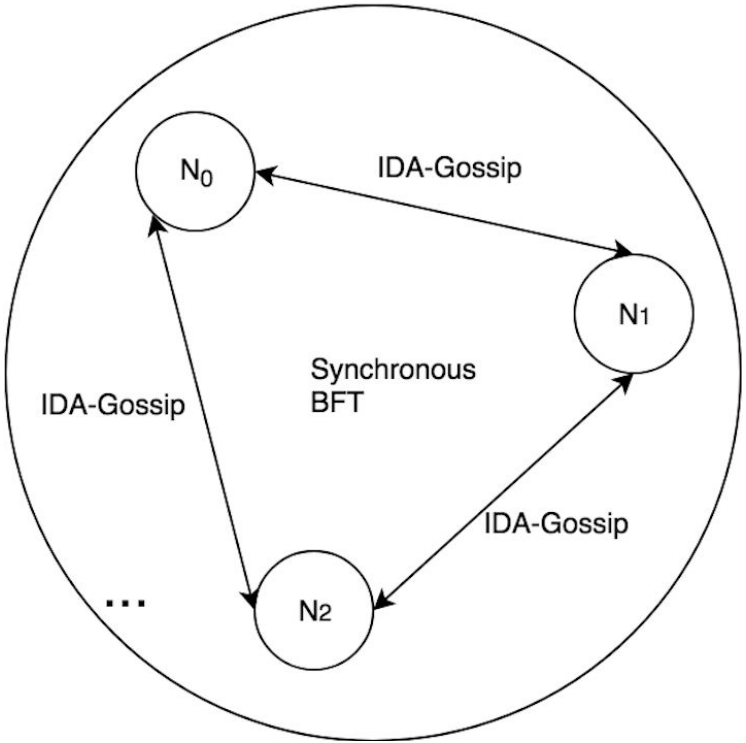
RapidChain

- Bootstrap
  - Sampler Graph
- Consensus
  - Intra-Committee: IDA-Gossip + Synchronous Consensus
  - Inter-Committee: Routing via Kademlia DHT
- Reconfiguration
  - Bounded Cuckoo Rule

Workflow - Consensus



Workflow - Consensus



Consensus - IDA-Gossip

What is IDA?

Let M denote the message to be gossiped to d neighbors, f denote the fraction of corrupt neighbors, and κ denote the number of chunks of the large message. First, the sender divides M into (1 − f)κ-equal sized chunks M1, M2, . . . , M(1−f)κ and applies an erasure code scheme (e.g., Reed-Solomon erasure codes [58]) to create an additional f κ parity chunk to obtain M1, M2, . . . , Mκ . Now, if the sender is honest, the original message can be reconstructed from any set of (1 − f)κ chunks.

Workflow

Next, the source node computes a Merkle tree with leaves M1, . . . Mκ . The source gossips Mi and its Merkle proof, for all 1 ≤ i ≤ κ, by sending a unique set of κ/d chunks (assuming κ is divisible by d) to each of its neighbors. Then, they gossip the chunks to their neighbors and so on. Each node verifies the message it receives using the Merkle tree information and root. Once a node receives (1 − f)κ valid chunks, it reconstructs the message M, e.g., using the decoding algorithm of Berlekamp and Welch [10].

Speed up large block broadcasting

Consensus - Synchronous Consensus

Round 1: The leader gossips a messages containing Hi and a tag in the header of the message that the leader sets it to **propose**.

Round 2: All other nodes in the network echo the headers they received from the leader, i.e., they gossip Hi again with the tag **echo**. This step ensures that all the honest nodes will see all versions of the header that other honest nodes received in the first round. Thus, if the leader equivocates and gossips more than one version of the message, it will be noticed by the honest nodes.

Round 3: If an honest node receives more than one version of the header for iteration i, it knows that the leader is corrupt and will gossip Hi′ with the tag **pending**, where Hi′ contains a null Merkle root and iteration number i.

Round 4: If an honest node receives f + 1 echoes of the same and the only header Hi for iteration i, he accepts Hi , and gossips Hi with the tag **accept** along with all the f + 1 echoes of Hi . The f + 1 echoes serve as the proof of why the node accepts Hi .

Consensus - Synchronous Consensus

Pipelining

RapidChain allows a new leader to propose a new block while re-proposing the headers of the pending blocks. The votes can be permanent or temporary relative to the current iteration.

Safe value

A new proposal is safe if it does not conflict with any accepted value with a correct proof, if there is any.

DoS Attack

If a node sends an echo for Hj at any iteration i ≥ j, its temporary vote is Hj in iteration i. To accept a header, a node requires at least f + 1 votes (permanent or temporary for the current iteration). If a node accepts a header, it will not gossip more headers since all nodes already know its vote. This will protect honest nodes against denial-of- service attacks by corrupt leaders attempting to force them echo a large number of non-pending blocks.

Consensus - Cross-shard Transaction

OmniLedgers' Solution

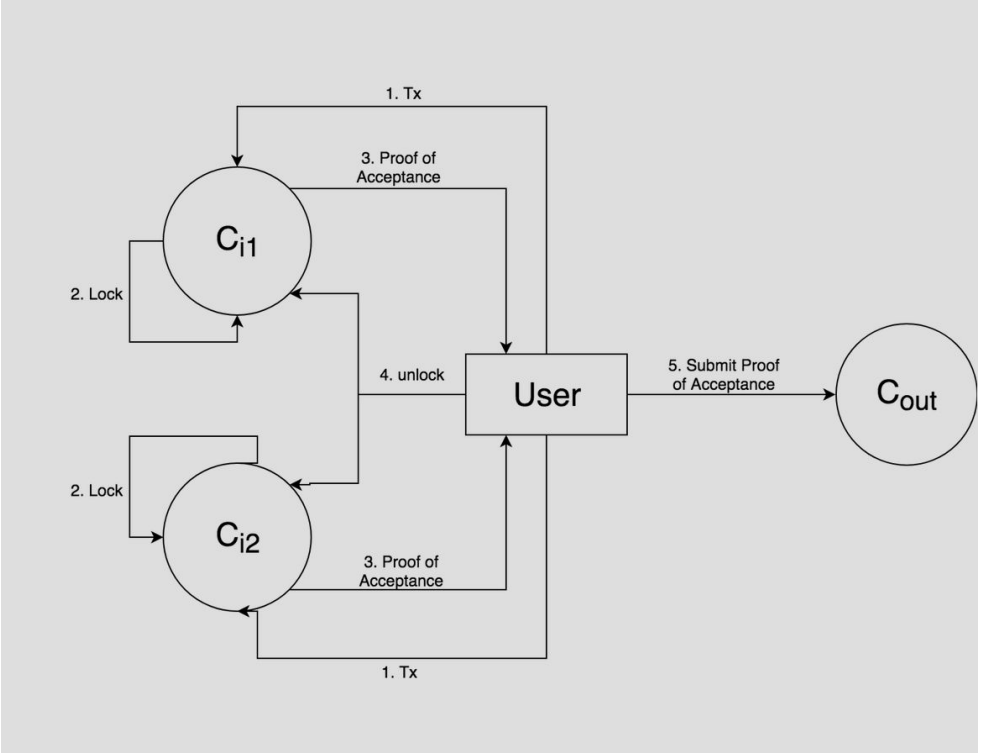
To avoid the tx committed partially, User obtain a proof-of-acceptance from every input committee and submit the proof to the output committee for validation

1. User sends tx to all input committees
2. Each input committee first locks the corresponding input UTXO(s) and issues a proof-of-acceptance, if the UTXO is valid.
3. The user collects responses from all input committees and issues an "unlock to commit".

Drawbacks

1. Users heavily involved
2. One proof needs to be generated for every transaction
3. The transaction has to be gossiped to the entire network

Consensus - Cross-shard Transaction



Consensus - Cross-shard Transaction

Rapidchain's Solution

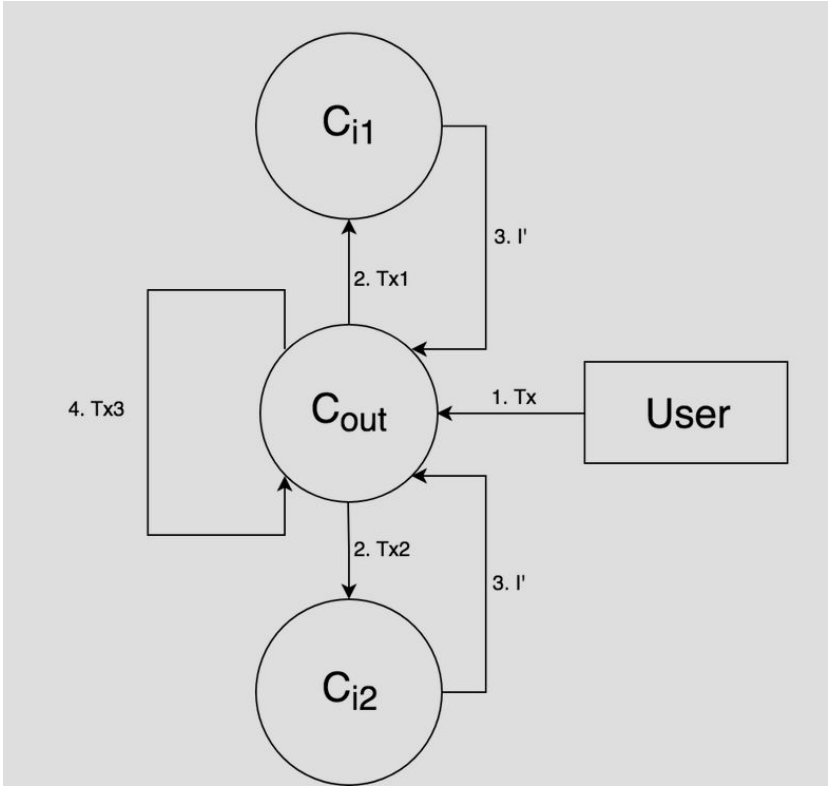
We assume tx has two inputs I1, I2 and one output O.(all belong to different committee). The leader of Cout, creates three new transactions: For i ∈ {1, 2}, txi with input Ii and output Ii′, where |Ii′| = |Ii | (i.e. the same amounts) and I ′ belongs to Cout. tx3 with inputs I ′ and I ′ and output O. The leader sends txi to Ci via the i i 1 2 i ′ in inter-committee routing protocol, and Cin adds txi to its ledger. If txi is successful, Cin sends Ii to Cout. Finally, Cout adds tx3 to its ledger.

Shift the responsibility from user to Cout

Inter-committee Routing

Strawman Scheme vs **Reference Committee** vs **Kademlia DHT Routing**

Consensus - Cross-shard Transaction



Consensus - Cross-shard Transaction

Kademlia DHT Routing

Rapidchain employ the Kademlia routing mechanism in RapidChain at the level of committee-to-committee communication.

- Each RapidChain committee maintains a routing table of log n records which point to log n different committees which are distance 2^i for $0 \leq i \leq \log n - 1$ away.
- Each node stores information about all members of its committee as well as about loglog(n) nodes in each of the logn closest committees to its own committee.

Workflow Reconfiguration

Offline PoW

Epoch Randomness Generation

Committee Reconfiguration

Workflow Reconfiguration

Offline PoW

- To against Sybil Attack
- In each epoch, a fresh puzzle is generated based on the **epoch randomness**
- Reference Committee C_R : Committee to check PoW
- Reference Block: Block including the list of active nodes

Workflow Reconfiguration

Epoch Randomness Generation

Steps:

1. Run DAG(Distributed Random Generation) to agree on a unbiased randomness
2. C_R includes the randomness in the reference block so other committees can randomize their epochs

Randomness Generation: VSS(Verifiable Secret Sharing)

Workflow Reconfiguration

Bounded Cuckoo Rule

Goal: At any moment, the committees are balanced and honesty.

1. The reference committee defines the set of the largest m/2 committees (who have more active members) as the active committee set, which we denote by A. We refer to the remaining m/2 committees with smaller sizes as inactive committee set denoted by I.
2. Active committees accept new nodes that have joined the network in the previous epoch, as new members of the committee. However, inactive committees only accept the members, who were part of the network before, to join them.
3. For each new node, the reference committee, C_R , chooses a random committee C_a from the set A and adds the new node to C_a . Next, C_R evicts (cuckoos) a constant number of members from every committee (including C_a) and assigns them to other committees chosen uniformly at random from I.

Workflow Reconfiguration

Protocol 1 Epoch Reconfiguration

1. **Random generation during epoch** $i-1$
   (a) The reference committee (C_R) runs the DRG protocol to generate a random string r_i for the next epoch.
   (b) Members of C_R reveal r_i at the end of epoch $i-1$.
2. **Join during epoch** i
   (a) Invariant: All committees at the start of round i receive the random string r_i from C_R.
   (b) New nodes locally choose a public key PK and contact a random committee C to request a PoW puzzle.
   (c) C sends the r_i for the current epoch along with a timestamp and 16 random nodes in C_R to P.
   (d) All the nodes who wish to participate in the next epoch find x such that $O = \mathsf{H}(\mathtt{timestamp}||\mathsf{PK}||r_i||x) \leq 2^{\gamma-d}$ and sends x to C_r.
   (e) C_r confirms the solution if it received it before the end of the epoch i.
3. **Cuckoo exchange at round** $i+1$
   (a) Invariant: All members of C_R participate in the DRG protocol during epoch i and have the value r_{i+1}.
   (b) Invariant: During the epoch i, all members of C_R receive all the confirmed transactions for the active nodes of round $i+1$.
   (c) Members of C_r will create the list of all active nodes for round $i+1$ and also create A, the set of active committees, and I, the set of inactive committees.
   (d) C_R uses r_{i+1} to assign a committees in A for each new node.
   (e) For each committee C, C_R evicts a constant number of nodes in C uniformly at random using0 r_{i+1} as the seed.
   (f) For all the evicted nodes, C_R chooses a committees I uniformly at random using r_{i+1} as the seed and assigns the node to the committee.
   (g) C_R adds r_i and the new list of all the members and their committees and add it as the first block of the epoch to the C_R's chain.
   (h) C_R gossips the first block to all the committees in the system using the inter-committee routing protocol.

Workflow Bootstrap

Sampler Graph

A deterministic random graph called the sampler graph which allows sampling a number of groups such that **the distribution of corrupt nodes in the majority of the groups is within a δ fraction of the number of corrupt nodes in the initial set**.

Subgroup Election

Members of each group run the DRG protocol to generate a random string s and use it to elect the parties associated with the next level groups. (H(s||ID) <= 2^{256-e}, e=2)

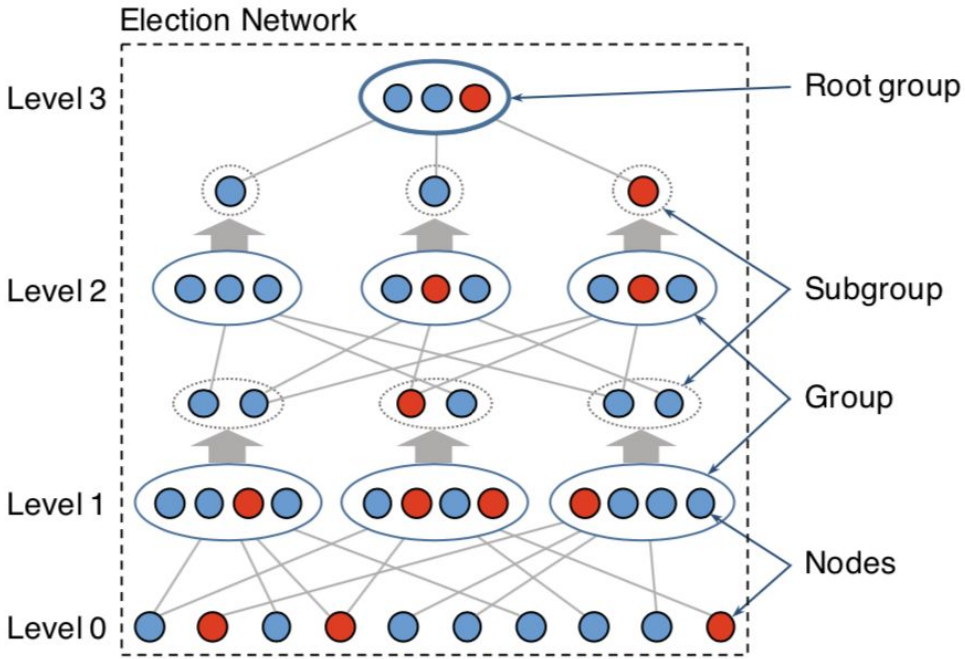
Subgroup Peer Discovery

The elected nodes will gossip this information and a proof from different members of the group, to all the nodes.

Committee Formation

The result of executing the above election protocol is a group with honest majority whom we call root group. Root group selects the members of the first shard, *reference shard/committee*. The reference committee partitions the set of all nodes at random into sharding committees

Workflow Bootstrap



Evaluation

Experiments

- Language: Go
- 4,000 nodes by oversubscribing a set of 32 machines each running up to 125 RapidChain instances
- Each machine has a 64-core Intel Xeon Phi 7210 @ 1.3GHz processor and a 10-Gbps communication link.

Assumption

- A latency of 100 ms for every message and a bandwidth of 20 Mbps for each node
- Bootstrap Phase: each node in the global P2P network can accept up to 8 outgoing connections and up to 125 incoming connections.(Same with Bitcoin Core)
- Intra-Committee: 16 outgoing connections and up to 125 incoming connections
- Each block of transaction consist of 4,096 transactions, where each transaction consists of 512 bytes resulting in a block size of 2 MB.

Evaluation

Metrics

- Block Size
- Throughput Scalability
- Transaction Latency
  - confirmation latency
  - user-perceived latency
- Reconfiguration Latency
- Number of Nodes
- Committee Size

Evaluation

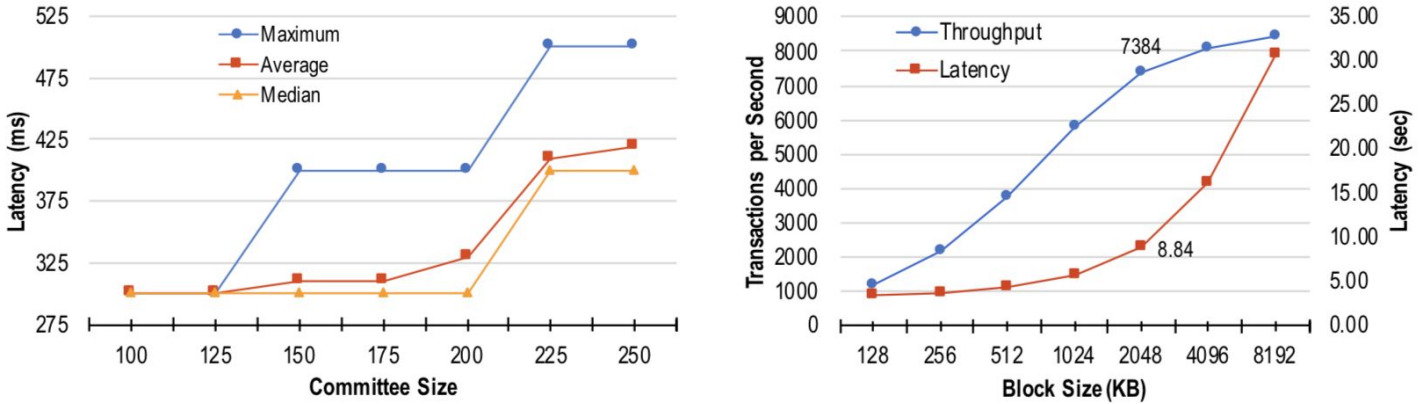


Figure 4: Latency of gossiping an 80-byte message for different committee sizes (left); Impact of block size on throughput and latency (right)

Evaluation

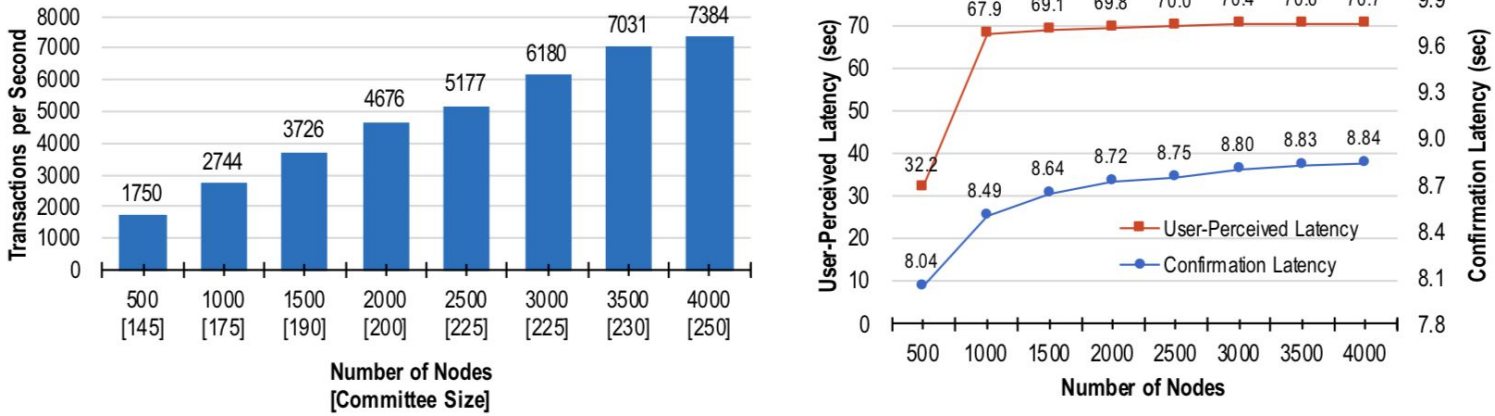


Figure 5: Throughput scalability of RapidChain (left); Transaction latency (right)

Evaluation

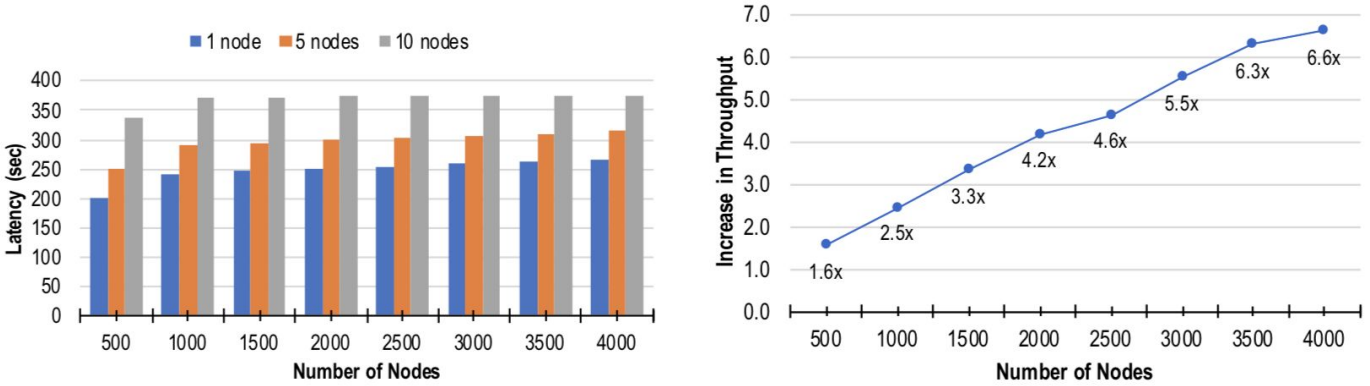


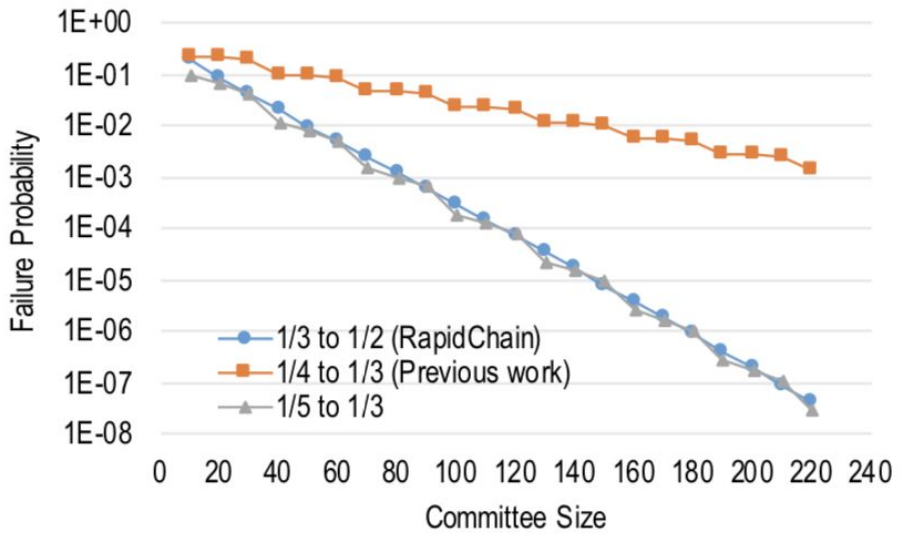
Figure 6: Reconfiguration latency when 1, 5, or 10 nodes join (left); Impact of batching cross-shard verifications (right)

Evaluation

Protocol	Network Size	Storage
Elastico [46]	1,600 nodes	2,400 MB (estimated)
OmniLedger [41]	1,800 nodes	750 MB (estimated)
RapidChain	1,800 nodes	267 MB
RapidChain	4,000 nodes	154 MB

Table 2: Storage required per node after processing 5 M transactions without ledger pruning

Security


1E+00
1E-01
1E-02
1E-03
1E-04
1E-05
1E-06
1E-07
1E-08
Failure Probability
0
20
40
60
80
100
120
140
160
180
200
220
240
Committee Size
1/3 to 1/2 (RapidChain)
1/4 to 1/3 (Previous work)
1/5 to 1/3

Complexity

Protocol	ID Generation	Bootstrap	Consensus	Storage per Node
Elastico [46]	$O(n^2)$	$\Omega(n^2)$	$O(m^2/b + n)$	$O(\|B\|)$
OmniLedger [41]	$O(n^2)$	$\Omega(n^2)$	$\Omega(m^2/b + n)$	$O(m \cdot \|B\|/n)$
RapidChain	$O(n^2)$	$O(n\sqrt{n})$	$O(m^2/b + m \log n)$	$O(m \cdot \|B\|/n)$

Table 3: Complexities of previous sharding-based blockchain protocols

Thank you!

Shu Dong
09/23/2018