Improving Blockchain Scalability: Sharding

Sajan Maharjan

20182095

CSED 601 Dependable Computing

19th Dec. 2018

Recap

- Discussion of dependability issues in blockchain
- 1st Presentation-
  - Solving integrity issues in blockchain using LedgerGuard
- 2nd Presentation-
  - Scalability issues, transaction processing capability, scalability trilemma
  - Proposed solutions- sharding, offchain solutions, consensus protocols, *etc.*
- Final Presentation-
  - Details on Sharding

Introduction

- Sharding- partitioning of network into smaller committees, each of which processes a disjoint set of transactions (or “shards”)
- Makes use of Proof of Work and Byzantine consensus algorithms
- ELASTICO – first secure candidate for a sharding protocol to open blockchain that tolerate byzantine adversaries
- Achieves linear transaction throughput with increase in the computational power of the network

Assumptions

- Given n miners in the network, can tolerate upto $f <= n/4$ byzantine adversaries.
- Honest nodes are reliable during protocol runs and failed or disconnected nodes are counted as byzantine nodes.
- Entire network of size n is divided into smaller shards or committee each of size at least c nodes
- Each shard outputs a separate set of transactions
- Partially synchronous network- any broadcasted message will reach a node within a bounded delay of δt seconds

Design

- Parallelization is achieved by dividing total computational power into smaller shards and Directory Service committee, each processing a disjoint set of transactions
- Each DS committee or shard of size c, runs byzantine consensus protocol to agree on a set of transactions
- Each DS committee or shard has its own leader
- DS committee is responsible for combining the transaction blocks selected from each shard
- Algorithms proceed in epochs or rounds

Design (contd…)

- Each epoch involves the following steps
  - Identity Establishment and Committee Formation
  - Shard Assignments
  - Sharding Logic Publishing
  - Transaction Sharding and Building Micro-Block Consensus
  - Final Block Proposal and Consensus
  - Block Confirmation and Epoch Randomness Generation

Design (contd…)

- Step 1: Identity Establishment and Committee Formation
  - Pseudonymous identity using public key and IP address
  - Two group of nodes/miners- DS nodes and regular nodes
  - Miners find a PoW solutions corresponding to their identity to join mining
  - $H(\text{epochRandomness} \,||\, IP \,||\, PK \,||\, \text{nonce}) <= 2^{\gamma - D}$
    - γ is the length of hash output
    - D is the no. of leading zeros
  - PoW prevents sybil nodes
  - Epoch randomness prevents byzantine adversary from precomputing the hash value output before a specified time or epoch

Design (contd…)

- Step 1: Identity Establishment and Committee Formation (contd…)
  - Election of directory service nodes takes place first
  - DS Committee has a fixed number of nodes and the first N_O nodes to solve the PoW are selected into DS Committee
  - A leader is elected from one of the members of DS Committee and new leader/committee members are added in each epoch
  - Previous epoch randomness value is used for valid PoW solutions

Design (contd…)

- Step 1: Identity Establishment and Committee Formation (contd…)

Algorithm 1: PoW_1 for DS committee election.

Input: i: Current `DS-epoch`, DS_{i-1}: Prev. DS committee composition.
Output: `header`: DS-Block header.

On each competing node:
// get epoch randomness from the DS blockchain
// DB_{i-1}: Most recent DS-Block before start of i-th epoch
$r_1 \leftarrow \mathsf{GetEpochRand}(\mathrm{DB}_{i-1})$
// get epoch randomness from the transaction blockchain
// TB_j: Most recent TX-Block before start of i-th epoch
$r_2 \leftarrow \mathsf{GetEpochRand}(\mathrm{TB}_j)$
// pk: node's public key, `IP` = node's IP address
`nonce`, `mixHash` $\leftarrow \mathsf{Ethash\text{-}PoW}(pk, \mathtt{IP}, r_1, r_2)$
`header` $\leftarrow \mathsf{BuildHeader}(\mathtt{nonce}, \mathtt{mixHash}, pk)$
// `header` includes pk and `nonce` among other fields
// `IP`, `header` is multicast to members in the DS committee
$\mathsf{MulticastToDS}_{i-1}(\mathtt{IP}, \mathtt{header})$
return `header`

Design (contd…)

- Step 1: Identity Establishment and Committee Formation (contd…)
  - Completion of DS Committee elections -> Regular Nodes/Miner nodes elected

Algorithm 2: PoW_2 for shard membership.

Input: i: Current `DS−epoch`, DS_i: Current DS committee composition.

Output: `nonce`, `mixHash`: outputs of Ethash-PoW

On each competing node:

// get epoch randomness from the DS blockchain

// DB_{i-1}: Most recent DS-Block before start of i-th epoch

$r \leftarrow \mathsf{GetEpochRand}(\mathrm{DB}_i)$

// pk: node's public key, `IP` = node's IP address

$\texttt{nonce}, \texttt{mixHash} \leftarrow \mathsf{Ethash\text{-}PoW}(pk, \texttt{IP}, r)$

// `IP`, `header` is multicast to members in the DS committee

$\mathsf{MulticastToDS}_i(\texttt{nonce}, \texttt{mixHash}, pk, \texttt{IP})$

return `nonce`, `mixHash`

Design (contd…)

- Step 2: Shard Assignment
  - Nodes that solve PoW solutions are assigned to different shards numbers-
    - Compare nonce values of PoW solutions and assign them to shard number on increasing order
    - If there are 2^s different shards, use last s-bits of H to assign to different shard
  - H is random -> last s-bits of H is random -> assignment of a node to a particular shard is also random -> each shard will have no more than 1/3 of malicious nodes
  - One of the nodes in each shard is assigned leader (Compare nonce values?)

Design (contd…)

- Step 3: Shard Logic Publishing
  - DS Nodes publish information on public channel
    - Identities and connection information of DS Nodes
    - List of selected nodes in each shard
    - Sharding Logic for Transaction
      - Which shard will store a specific transaction?
      - For a given transaction from A to B, and assuming there are 2^s different shards, use the rightmost s-bits of the sender address to determine which shard stores the transaction
      - Double spending can be prevented by checking nonce value of transaction against nonce value in the account states

Design (contd…)

- Step 4: Transaction Sharding and Building Micro-Block Consensus
  - As explained earlier, which shard stores which transaction is determined
  - Transactions are collected by nodes in a shard and sent to shard leader
  - Transactions are grouped into a block called MicroBlock by leader
  - Consensus for validity of each transaction and MicroBlock -> requires 2/3 of signatures of nodes in the block
  - Upon success, MicroBlock headers (containing txn hash) and Bitmap (signifies multi-signatures) sent to few DS Nodes by leader

Design (contd…)

- Step 5: Final Block Proposal and Consensus
  - DS Nodes collect MicroBlock headers and sends to DS Committee leader
  - Leader validates MicroBlock and combines them to generate Final Block header and proceeds to consensus
  - Consensus requires 2/3 of signatures approval from DS nodes
  - Leader builds Bitmap and Final Block header and sends it to nodes in each shard

Design (contd…)

- Step 6: Block Confirmation and Epoch Generation
  - Nodes in each shard confirm the signature in Bitmap against DS Nodes in the public channel
  - Final Block header containing transaction hash compared against Micro-block header transaction hash
    - If match -> transaction data is appended to the final block in the local shard
    - Account States & Global States are updated (State Sharding)
  - At the end of each epoch some random value is generated to prevent malicious nodes from pre-computing hash values

Leader Change

- Byzantine leader can intentionally drop or delay messages from honest nodes
- Periodically change leader of each shard and DS committee
- Leader of each shard is changed after every micro block
- Leader of the DS Committee is changed after every final block
- If size of DS consensus group is *n*, then epoch lasts for generation of *n* final blocks with leader change after every final block
- Each final block consists of 1 micro-block from each shard

Cross-Shard Transactions

- No need to communicate with another shard if both sender and receiver in same shard
- Alice (address in Shard #1) wants to send tokens to Bob (address in Shard #2)
- Requires updating state in both Shard #1 and Shard #2
- Two Approaches
  - Synchronous
    - State Transition information related to both addresses produced at the same time
  - Asynchronous
    - “Credit” shard executed after “Debit” shard has completed execution
    - Validity of transactions questionable in the case of forks

Cross-Shard Transactions (contd…)

- Requires the assumption of less than 1/3 malicious nodes in each shard to prevent forking
- Say (by a negligible probability), malicious nodes collude on a shard

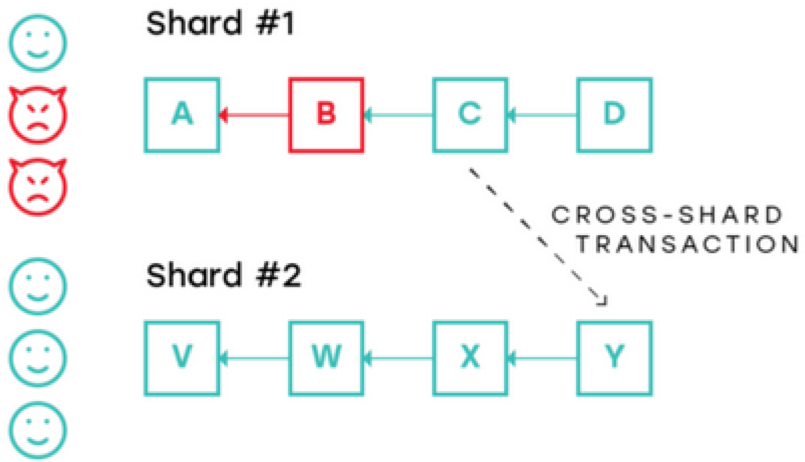

Image Courtesy: [Medium]

Cross-Shard Transactions (contd…)

- Undirected Graph can be used that shares the cross-shard transaction history

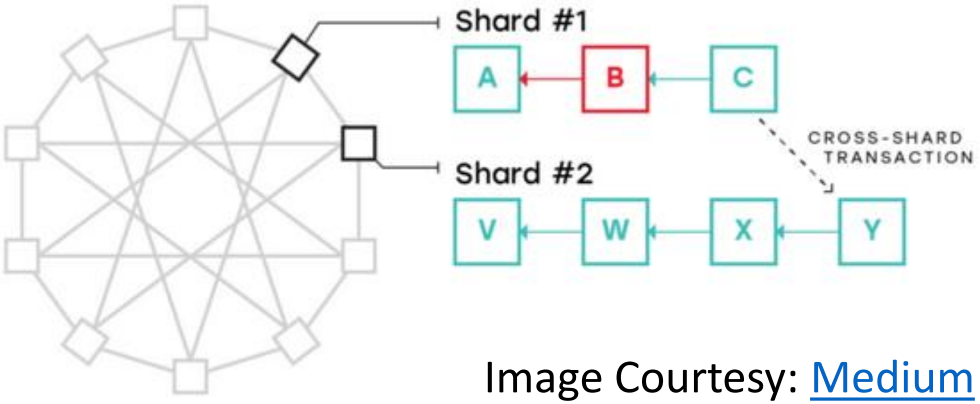

Image Courtesy: [Medium]

- Shard #2 invalidates block B from Shard #1

Cross-Shard Transactions (contd…)

- Say (very very negligible probability) two or more shards are colluded

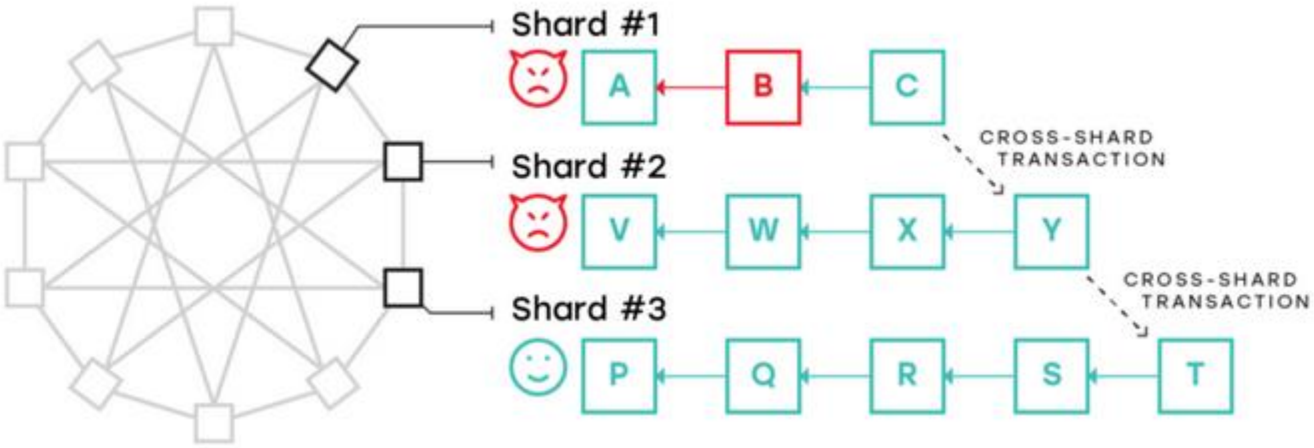

Image Courtesy: [Medium]

- Malicious block B is obfuscated

Cross-Shard Transactions (contd…)

- Fisherman Approach
  - Honest validators issues a challenge that a certain block is invalid, for a given period of time
  - Shard is secure even with just one honest node in the shard

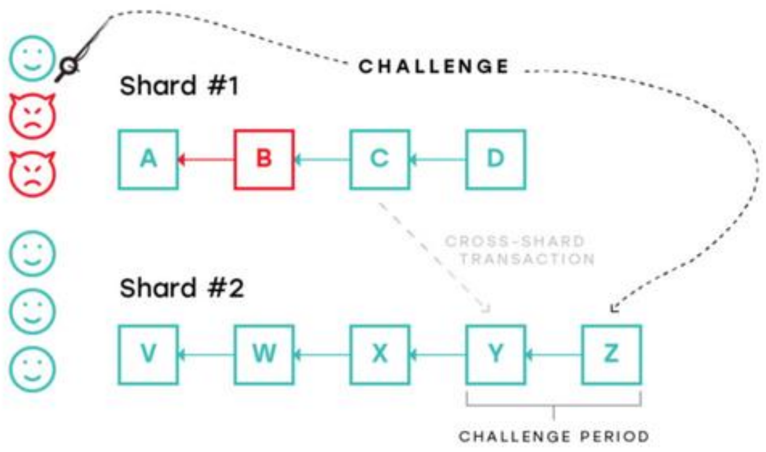

Image Courtesy: Medium

- Cryptographic Proof of Computation

Scalability Achievement

- Transaction data is not transmitted, only micro block header and final block header containing transaction hash is transmitted
- Use of EC-Schnorr signature algorithm achieves speed and requires less size for multi-signatures
- Elastico shows that the throughput scales up linearly in the computation capacity of the network
- Ziliqa believes achieving 1000X scalability of Ethereum given a network size of Ethereum

References Used

- The Ziliqa Technical Whitepaper, https://docs.zilliqa.com/whitepaper.pdf
- CCS 2016 - A Secure Sharding Protocol For Open Blockchains, https://www.youtube.com/watch?v=AlPMYyGVkRk
- A Secure Sharding Protocol For Open Blockchains, http://delivery.acm.org/10.1145/2980000/2978389/p17-luu.pdf?ip=141.223.124.8&id=2978389&acc=ACTIVE%20SERVICE&key=0EC22F8658578FE1%2E25E83D88E716D18F%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&__acm__=1544631713_e0a9cf3b64b146e76b8dd6a126a46c01
- Unsolved Problems in Blockchain Sharding, https://medium.com/nearprotocol/unsolved-problems-in-blockchain-sharding-2327d6517f43
- The Authoritative Guide to Blockchain Sharding, Part 1, https://medium.com/nearprotocol/the-authoritative-guide-to-blockchain-sharding-part-1-1b53ed31e060