ECE 8930: Blockchain Technology and Web 3.0

Homework 2

Saurabh Sharma
10/27/2023

Q1) In IOTA, the Weighted Random Walk method is proposed to address the “lazy tips” of Unweighted Random Walk. Another more “intuitive” solution is to force the incoming transactions to select only recent transactions to validate. What are the problems of this more intuitive solution?

As recommended as an "intuitive" solution, restricting incoming transactions to choose only recent transactions for validation has a number of basic difficulties.

- Opposed to Decentralization: Against Decentralization: A core component of distributed ledger technologies like cryptocurrencies and IOTA is decentralization. The network would exhibit predictable behavior if decision-making were centralized and all new transactions needed user authorization. Because predictable systems are easier to target or manipulate, this can lead to vulnerabilities and go against the decentralized ethos.
- Timing Concerns: IOTA's underlying data structure, the Tangle, does not have a reliable way to pinpoint the exact moment (in milliseconds or microseconds) at which a transaction was added. It is challenging to choose transactions that are truly "recent" in the absence of a precise timestamp. This could lead to disagreements or conflicts on which transactions are deemed current enough to validate.
- Punishing Lazy Tips:Transactions that are only accepted for older transactions are referred to as "lazy tips". As they don't improve the security or development of the system, they are perceived as being indolent. The Weighted Random Walk approach lessens the chance that lazy tips will be chosen for validation by biasing the system against them. In order to ensure justice, a lazy recommendation is therefore only marginally less likely to be accepted than it would otherwise be.
- Weighted Transactions: The cumulative weight of a transaction, which adds one to the overall number of approvers (direct and indirect), indicates how relevant the transaction is. Because each transaction has a weight, the system automatically favors more significant or "heavy" transactions, making sure that significant transactions receive the attention they deserve. This method considers the transaction's "weight" in addition to its date in order to help choose which transactions are chosen for validation.
- Probabilistic Approach: "No probability at all" relates most definitely to the problem of an unweighted strategy where each transaction has an equal chance of being chosen. Inefficiencies could arise if important or substantial transactions are overlooked in the absence of probability weighting.

In summary, despite its seeming simplicity, the "intuitive" method leads to inefficiencies, timing errors, and centralization issues. The Weighted Random Walk method, on the

other hand, addresses the problem of lazy tips, honors the system's decentralized nature, and ensures that significant transactions get the consideration they merit.

Q2) How does IOTA solve the scalability bottleneck in BTC and ETH?

IOTA presents a novel strategy intended to address scalability issues, which prominent blockchains such as Bitcoin (BTC) and Ethereum (ETH) particularly confront. The following summarizes IOTA's approach to resolving these problems:

- Using Tangle: IOTA uses a Directed Acyclic Graph (DAG) called the "Tangle" instead of a blockchain. This configuration circumvents the limitations imposed by blocks and the intervals that go along with it to enable concurrent transaction additions.
- No Miner Dependency: IOTA does away with the norm that miners are required to authenticate and add transactions, unlike BTC and ETH. A user who validates a new transaction in its model also authorizes two previous ones. This approach ensures charge-free transactions and prevents congestion caused by miners.
- Scalability is enhanced by Network Activity: One of IOTA's unique features is that it gets more scalable as activity levels rise. This is explained by its transactional nature, in which approving one transaction necessitates validating two more. However, during periods of heavy activity, blockchains such as Bitcoin and Ethereum frequently experience congestion.
- Absence of Fixed characteristics: Static characteristics, such as block size or creation intervals, sometimes impede the performance of blockchains. IOTA is unaffected by these possible obstacles because it does not have any barriers.
- Countering Mining Power Centralization: IOTA reduces centralization, a problem that affects ETH and BTC because of powerful mining pools. The architecture of the protocol encourages a more decentralized ecosystem by reducing the allure of specialized mining equipment.
- Fee-free Transactions: Free transactions are supported by the IOTA architecture. This contrasts sharply with BTC and ETH, where excessive transaction fees may arise from soaring network activity.
- Remain Safe From Quantum Dangers: IOTA is designed to resist the risks posed by quantum computing. Although this may not directly benefit scalability, it does highlight the platform's dedication to long-term security.
- Enabling Tangle Partitions: The architecture of IOTA enables its use in divided settings. This "sub-tangle" approach offers a distinct scalability advantage by enabling discrete processes that can later integrate back into the main Tangle.
- Criticism: In summary, even if IOTA's scaling solutions are novel, there is some criticism of them, particularly with regard to centralization and security issues. Since the crypto landscape is always changing, it is necessary to balance the advantages of using it against any potential drawbacks, as is typical with emerging technology.

Q3) What is a deterministic protocol and what is a nondeterministic protocol? Is PoW a deterministic protocol or nondeterministic protocol? Explain your answer.

- Deterministic Protocol:
  A deterministic protocol is one where a given input will always produce the same output, without any variation. In the context of blockchain, a deterministic system would be one where every participant can predict the outcome of a process or operation based on its initial state and any inputs it receives. For instance, the consensus process in many blockchains is deterministic; given the same set of transactions and the same network state, every node should come to the same conclusion about the validity of those transactions.
- Non-deterministic Protocol:
  A nondeterministic protocol, on the other hand, is one where the output may vary for the same input, often due to the introduction of randomness or other unpredictable factors. This unpredictability can arise from various reasons, such as the involvement of multiple actors, random number generation, or inherent unpredictability in the system's design.
- Proof of work and determinism:
  Proof of Work (PoW) is a consensus algorithm used by many blockchains, including the original implementation of Bitcoin. The core idea behind PoW is for network participants (miners) to solve a cryptographic puzzle, which requires significant computational effort.
- PoW is nondeterministic. Here's why:

  The "work" in PoW involves finding a value (called a nonce) that, when hashed with other transaction data, produces a hash value that meets certain criteria (e.g., starts with a specific number of zeros).

  This process is essentially a trial-and-error search. Even if two miners start at the exact same time with the same set of transactions, they're unlikely to find the correct nonce at the same moment. There's an element of randomness and unpredictability involved.

  Thus, while the rules of the PoW algorithm are well-defined and consistent, the actual outcome (i.e., who finds the correct nonce first) is not predictable with certainty ahead of time.

  In summary, while the protocols governing how PoW operates are clearly defined and consistent across the network, the actual process of finding a solution is nondeterministic due to its inherent unpredictability.

Q4) PoS is based on the PBFT algorithm. What are the major changes/improvements of PoS compared to PBFT? Or what are the main differences between the two?

While some Proof of Stake (PoS) implementations draw inspiration from the Practical Byzantine Fault Tolerance (PBFT) algorithm, it's not accurate to say that PoS is based on PBFT. However, both PoS and PBFT are consensus mechanisms aimed at achieving agreement across distributed systems.

Practical Byzantine Fault Tolerance (PBFT):
PBFT is a consensus algorithm designed to handle malicious nodes in a distributed system. It operates in rounds, requiring a series of communication steps between nodes to reach consensus.

- Leader-based: A leader node is elected for each round. This leader proposes a value, and other nodes then vote on this value. If a supermajority (typically 2/3) of nodes agrees, consensus is achieved. If the leader is found to be malicious, it is replaced in the next round.
- Communication Overhead: PBFT requires a significant amount of communication between nodes. In systems with a large number of nodes, this can lead to scalability issues.
- Fixed Set of Validators: PBFT generally works with a known set of validators, meaning the total number of nodes in the system is predetermined.

Proof of Stake (PoS):

- Algorithm Overview: PoS is a type of consensus mechanism where validators are chosen to create new blocks based on the number of tokens they hold and are willing to "stake" or lock up as collateral. The idea is that validators have something to lose and are, therefore, incentivized to act honestly.
- Staking: Validators are chosen based on the amount of cryptocurrency they hold and are willing to "stake" or lock up. This stake acts as both an incentive for honest behavior and a deterrent against malicious activity.
- Dynamic Validators: Unlike PBFT, which typically has a fixed set of validators, PoS systems can have a dynamic set of validators, as participants can choose to join or leave the staking process.
- Economic Incentives: Validators in PoS are rewarded for adding new blocks (often through block rewards and/or transaction fees). Malicious actors stand to lose their staked tokens if they are discovered.

Differences between PBFT and PoS:

- Basis for Validator Selection: While PBFT relies on a rotating leader-based system, PoS depends on the amount of stake to select validators.

- Communication: PBFT requires a lot of back-and-forth communication, making it less scalable for large networks. PoS, depending on its implementation, can be more scalable.
- Penalties: PoS introduces economic penalties (loss of stake) for malicious actions. PBFT relies more on the rotation of potentially malicious leaders.
- Validator Set: PBFT has a more static set of validators, while PoS allows for a dynamic set based on participants willing to stake their tokens.
- Incentives: PoS often provides economic rewards for block validation, while PBFT does not inherently reward nodes in the same manner.

While there are pure PoS systems, many modern blockchains use a combination of PoS and other consensus mechanisms (including elements inspired by PBFT) to achieve network consensus. The design often depends on the specific goals and requirements of the blockchain in question.

Q5) The Ethereum consensus consists of Casper FFG and the LMD-GHOST fork choice rule. We covered the former in class. Please explain the main idea of the fork choice rule.

The Casper FFG consensus mechanism is intended to be placed on top of the current Ethereum blockchain in order to add finality to the chain and facilitate Ethereum's shift from Proof of Work (PoW) to Proof of Stake (PoS). Casper FFG functions in conjunction with LMD-GHOST and other fork choice rules. It specifies how the protocol should choose between conflicting chains (forks) by calculating the canonical chain based on the most recent validator votes.

Multiple blocks may be suggested at about the same time in a blockchain system, particularly one with a quick block time like Ethereum. This may cause the chain to momentarily split. A fork choice rule aids nodes in the network in selecting the chain to continue when given several viable possibilities.

LMD-GHOST addresses this by focusing on the "latest messages" (votes) from validators:

- Weighted Votes: Generally based on their investment in the system, each validator's most recent vote (i.e., the block they have viewed and attested to most recently) is given weight. The canonical chain is the one that has received the most total weight in these votes.

- Greedy Selection: The protocol begins at the genesis block and, depending on the most recent messages from validators, chooses the child block at each height with the highest aggregate weight.
- Safety and Liveness: LMD-GHOST is designed to ensure both liveness (that is, that the system keeps evolving and creates new blocks) and safety (that nodes reach an agreement and do not complete conflicting blocks).

- Recency: LMD-GHOST responds dynamically to the judgments and observations made by network members in real time by considering the most recent signals from validators.

The primary goal of LMD-GHOST is to guarantee that, even in the face of numerous blocks and possible transient forks, the blockchain can swiftly settle on a single canonical version of the truth. This is in contrast to other systems that might use different criteria or just choose the longest chain. LMD-GHOST attempts to be a strong fork decision rule that is well-suited to the particular requirements and features of the Ethereum network by taking into account the most recent validator attestations and their corresponding weight.

Q6) Ethereum block datastructure.

Blocks consist of a Block header, transactions and an ommers list. Transactions and ommers list are stored in the form of Merkle Tree for faster access and verification.

Block header -

The block header is a pivotal component within the Ethereum blockchain architecture. It encapsulates crucial information about a specific block, serving as a fingerprint that uniquely identifies and validates the block's content. Comprising various fields, the block header includes the parent hash, which links the block to its predecessor, ensuring the chronological integrity of the blockchain. Additionally, it features uncle hashes, which acknowledge and reward miners who contributed valid but not canonical blocks. The state root, transaction root, and receipts root are essential components that represent the current state of accounts, the transactions within the block, and the receipts detailing transaction outcomes, respectively. The difficulty and nonce fields embody the cryptographic puzzle-solving process, known as mining, contributing to the security and immutability of the Ethereum network. As a whole, the block header encapsulates critical metadata that ensures the transparency, integrity, and immutability of the Ethereum blockchain. The "gas limit" defines the maximum computational work a block can perform, while "gas used" indicates the actual amount of computational work expended within a block, reflecting the resources consumed by transactions and smart contract executions on the Ethereum blockchain.

Now, we will explore each field in greater detail:

parentHash: The "parent hash" field in the Ethereum block header is a cryptographic reference to the preceding block in the blockchain. It plays a pivotal role in establishing the chronological order and immutability of transactions. This field contains the unique hash value generated

through the application of a cryptographic hash function to the header of the previous block. By referencing the parent hash, each block is securely linked to its predecessor, forming an unbroken chain of blocks. Any attempt to alter the data in a specific block would necessitate the alteration of all subsequent blocks, an impractical and computationally infeasible task. Thus, the parent hash field serves as a foundational element in ensuring the integrity and security of the Ethereum blockchain, guaranteeing that transactions are permanently recorded in the correct order.

ommerHash: In the Ethereum blockchain, the "ommerHash" holds a distinctive significance. It serves as a cryptographic fingerprint, encapsulating a set of valid but alternative blocks colloquially known as "ommers" or uncles. Ommers are blocks that were successfully mined around the same time as the main block but ultimately did not make it into the primary blockchain. This field is integrated within the block header, functioning as a crucial reference point for these uncle blocks. It plays a pivotal role in expanding and strengthening the network's mining participation, thereby enhancing its overall security posture.

Beneficiary: It designates the address to which the block rewards, also known as the coinbase reward, are allocated. This address is typically associated with the miner or mining pool responsible for successfully mining the block. It plays a pivotal role in ensuring that the mining incentives, in the form of newly minted Ether (ETH) and transaction fees, are directed to the rightful recipient. The beneficiary address is specified by the miner during the mining process, and it is a crucial element in maintaining transparency and accountability within the Ethereum network.

stateRoot : The "stateRoot" field in the Ethereum blockchain block header represents a critical component of the platform's data structure. It serves as a cryptographic hash that encapsulates the root of the state trie, a Merkle tree structure. This trie holds the current state of all accounts on the Ethereum network, including their respective balances and contract storage. By including the stateRoot in the block header, Ethereum ensures that every block contains a reference to the precise state of the network after executing all transactions within that block. This is crucial for validation and verification processes, allowing nodes to quickly confirm the correctness of the state and the results of transactions.

TransactionRoot: The "transactionRoot" field in the Ethereum block header is the cryptographic root hash of the transaction trie. This trie is a Merkle tree structure that efficiently organizes all the transactions contained within the block. Each transaction's data is hashed using the Keccak-256 algorithm and these hashes are hierarchically arranged to form the transaction trie. The resulting root hash, stored in the transactionRoot field, acts as a unique identifier for the set of transactions in the block. Any modification to even a single transaction within the block would lead to a completely different transactionRoot, providing a robust mechanism to detect tampering or unauthorized changes in the block's content. This ensures the integrity and immutability of the transactions within the block.

ReceiptsRoot: The "receiptsRoot" field in the Ethereum block header stores the root hash of the receipts trie, another Merkle tree structure. Receipts contain detailed information about the outcomes of transactions, such as events and logs. Similar to the transaction trie, each receipt's data is hashed and organized in a tree structure. The root hash, stored in the receiptsRoot field, represents the collective information from all the receipts in the block. This provides a strong mechanism to detect any tampering or unauthorized changes in the receipt information.

LogsBloom: The "logsBloom" field in the Ethereum block header is a compact representation of the event logs generated by smart contracts during the execution of transactions within the block. It is a Bloom filter, which is a space-efficient probabilistic data structure that allows for efficient membership tests. The logsBloom field provides a way to efficiently query and filter logs based on specific criteria, making it a crucial component for applications and services that rely on event-driven interactions on the Ethereum blockchain.

Difficulty: The "difficulty" field in the Ethereum block header is a numerical value that indicates how challenging it was for miners to find a valid nonce for the block. It's a crucial aspect of the Proof-of-Work (PoW) consensus algorithm. The difficulty is dynamically adjusted by the network protocol to maintain a consistent block creation time, aiming for roughly 15 seconds between blocks. This mechanism ensures that mining remains competitive and that new blocks are created at a relatively constant rate.

Number: The "number" field in the Ethereum block header represents the block number, which is essentially the position of the block within the blockchain's chronological sequence. It serves as a unique identifier for each block, allowing nodes to precisely locate and reference specific blocks in the chain. The block number increases sequentially with each new block added to the blockchain, reflecting the continuous progression of the Ethereum network.

GasLimit: The "gasLimit" field in the Ethereum block header designates the maximum amount of computational work, measured in gas units, that can be performed within the block. It is dynamically adjusted through consensus mechanisms and aims to strike a balance between allowing efficient transaction processing and preventing network abuse. Setting an appropriate gas limit is crucial; too low a limit may hinder legitimate transactions, while too high a limit could lead to network congestion and inefficiency.

GasUsed: The "gasUsed" field in the Ethereum block header signifies the actual amount of computational work, measured in gas units, that was expended within the block. It reflects the resources consumed by the execution of transactions and smart contracts. This field provides a concrete metric of the computational effort expended during the block's processing, which is critical for understanding the network's efficiency and resource allocation.

Timestamp: The "timestamp" field in the Ethereum block header records the exact time when the block was mined. It is represented as a Unix timestamp, which is the number of seconds since January 1, 1970 (UTC). The timestamp is an essential component for tracking the order of blocks and providing a temporal context for the transactions and activities recorded within the block.

ExtraData: The "extraData" field in the Ethereum block header allows for arbitrary data to be included by miners. It is typically used to convey additional information or messages, though its specific contents can vary widely. It provides a degree of flexibility for miners to include personalized or informative data within the block.

MixHash and Nonce: The "mixHash" and "nonce" fields are associated with the Proof-of-Work (PoW) mining process. Miners must find a valid nonce value through a computational process that involves repeatedly hashing the block header. The resulting "mixHash" must meet certain criteria defined by the network's difficulty level. These fields are instrumental in ensuring the security and integrity of the blockchain, as they require significant computational effort to find a valid nonce, which in turn helps to prevent unauthorized alterations to the block.

Block body -

TransactionsList -

The "transactions" field in the Ethereum block body is a crucial component that holds a list of all the transactions that are included in the block. Each transaction represents an interaction on the Ethereum network, and they can be either simple value transfers between accounts or more complex operations executed by smart contracts.
Transactions are encoded in a binary format and organized in a specific order within the block. Each transaction contains various fields, including sender and recipient addresses, the amount of ether transferred, data to be executed (for contract interactions), gas limits, and more.

Transaction Execution: When a block is mined, the Ethereum Virtual Machine (EVM) processes each transaction in the order they appear in the "transactions" field. It verifies the sender's account balance, checks the validity of the transaction, executes any associated code (in the case of contract interactions), and updates the state of accounts accordingly. This process consumes a certain amount of gas, which is the computational effort required to execute the transaction.

Transaction Receipts: For each transaction, a corresponding "receipt" is generated and stored in the block's "receipts" list. A receipt contains information about the outcome of the transaction, including logs and events that were emitted during its execution. These receipts are then used to compute the "receiptsRoot," a crucial component of the block header.
Interactions with Smart Contracts:
Transactions play a vital role in interacting with smart contracts on the Ethereum network. When a transaction involves a smart contract, the contract's code is executed, potentially altering the contract's internal state and emitting events. This interaction enables a wide range of decentralized applications (DApps) to function on the Ethereum blockchain.
In summary, the "transactions" field in the Ethereum block body holds a list of transactions, each representing an interaction on the network. These transactions are processed by the Ethereum Virtual Machine, potentially leading to changes in account balances or smart contract states. The outcomes of these transactions are recorded in receipts, which contribute to the overall state of the blockchain.

Ommers List -

The "Ommers List" (also known as "Uncles List") in the Ethereum blockchain's block body is a crucial component that plays a unique role in the network's consensus mechanism and incentive structure. Ommers, also referred to as uncles, are valid but non-main chain blocks that were mined nearly simultaneously with the main block. Despite not becoming part of the main blockchain, ommers contribute significantly to the security and decentralization of the Ethereum network.
It’s role -
Ommers serve as a way to acknowledge and reward miners who put in the effort to mine a block but did not have their block included as the canonical block. This is crucial for maintaining network security and encouraging more miners to participate. Without this incentive, miners

might be discouraged from participating if they believe their blocks won't be included in the main chain.

Structure of the Ommers List:
The Ommers List is a part of the block body and contains references to uncle blocks. Each entry in the list includes the header of an uncle block. This header typically contains fields like parent hash, beneficiary, state root, transaction root, etc., similar to the main block header.

The inclusion of ommers enhances the security of the network. By recognizing valid, but non-canonical blocks, Ethereum incentivizes a wider pool of miners to participate, increasing the network's overall computational power and making it more resistant to adversarial attacks. This, in turn, contributes to a more decentralized and secure blockchain.
In summary, the Ommers List in the Ethereum block body contains references to valid but non-main chain blocks, known as ommers or uncles. These ommers serve as an essential component of Ethereum's incentive structure, encouraging more miners to participate and contributing to the overall security and decentralization of the network.

Now that we have explained all the fields of the Ethereum blockchain in great detail the only thing remaining is explaining what is Merkle Tree that we have mentioned so many times and how and why it is used in the Ethereum blockchain. We'll do that below.

Merkle Tree -
A Merkle Tree is a hierarchical data structure used in computer science and cryptography. It is named after Ralph Merkle, who first described the concept in 1979. The tree is constructed by recursively hashing pairs of data (called leaves) until a single hash, known as the root hash or Merkle root, is obtained. It has mainly two properties: construction and verification.

Construction of a Merkle Tree:
A Merkle Tree is constructed through a recursive process of hashing data elements until a single root hash remains. Initially, each individual piece of data (often called a "leaf node") is hashed. These hashes are then paired, and each pair is concatenated and hashed again to form a parent node. This process continues until there is only one hash left, known as the Merkle root. The key property of a Merkle Tree is that any change in the original data, no matter how small, will result in a completely different Merkle root. This is due to the avalanche effect of cryptographic hash functions, where even a tiny change in input leads to a drastically different output.

Verification in a Merkle Tree:
To verify the integrity of a specific data element in a Merkle Tree, one doesn't need the entire dataset; only a subset of hashes is required. Starting from the leaf node representing the desired data element, one retrieves its hash. Then, using a series of paired hashes up the tree, combining with sibling hashes at each level, the process leads to the Merkle root. This resulting hash is then compared to the known Merkle root of the dataset. If they match, it confirms the

integrity of the desired data element. This verification process is highly efficient, requiring only a logarithmic number of hash operations compared to a linear scan of all data elements, making Merkle Trees a powerful tool for secure and efficient data verification in various applications, including blockchain technology.