Secure Decentralized Auction System: Design and Implementation

Ruben Pereira
FCUP
up202006195@up.pt

Ricardo Costa
FCUP
up202006868@up.pt

Simão Rodrigues
FCUP
up202005700@up.pt

***Abstract*—This paper presents the design and implementation of a decentralized system that manages a public ledger for auctions. The implementation uses a blockchain that supports Proof-of-Work and Proof-of-Reputation consensus mechanisms, a secure peer-to-peer (P2P) network based on S/Kademlia, and a simple auction mechanism for managing bids and sales. All communications are implemented using TLS over Netty for secure and reliable networking. Key security features include cryptographic signatures for transactions and blocks, trust-based reputation scoring, and protections against Sybil and Eclipse attacks. Despite the challenges encountered, the system demonstrates core functionalities of a secure, decentralized auction platform, paving the way for future improvements such as full Proof-of-Reputation integration and cloud-based testing.**

I. Introduction

This report focuses on our design and implementation of a distributed public ledger for an auction system. Unlike traditional blockchains such as Bitcoin or Ethereum, our system was developed with the primary goal of securely managing auctions in a fully distributed and non-permissioned environment.

The entire project was developed in Java, relying on Netty for networking and BouncyCastle for cryptographic operations.

The work is structured into three major components: the secure distributed ledger, a peer-to-peer communication layer, and a simple auction mechanism. The blockchain supports Proof-of-Work (PoW) consensus algorithms, with a framework for Proof-of-Reputation (PoR) implementation. The P2P layer is built ”on top” of S/Kademlia, providing resilience against Sybil and Eclipse attacks. Finally, the auction system implements a single-attribute English auction model, with transactions being securely broadcast and permanently stored in the blockchain.

Throughout the development, particular attention was given to ensuring system security, scalability, and fault tolerance. The system includes attack simulation capabilities to test resistance against common network-level threats such as Sybil and Eclipse attacks.

II. General Architecture

The general architecture represents a distributed ledger system with peer-to-peer nodes. On the client side, network operations are handled by dedicated threads, ensuring parallel and efficient processing of multiple simultaneous requests.

Inside each peer, the server module is responsible for managing critical components such as the blockchain, transaction pool, miner, and Kademlia-based storage. Our NetworkEngine class serves as the coordination point between these components, providing methods for sending and receiving messages, maintaining peer connections, and broadcasting blocks and transactions.

Each node in our system is equipped with its own RSA key pair, which serves multiple purposes:

- Generating the node’s unique identifier (through SHA-1 hashing of the public key)
- Signing blocks and transactions
- Establishing TLS connections for secure communications

The system also maintains a reputation engine that tracks peer behavior and adjusts trust scores based on successful or failed interactions. These trust scores are integrated into the S/Kademlia routing mechanism and influence consensus selection.

III. Blockchain

In our system, the blockchain was implemented as an immutable and ordered list of blocks, each cryptographically linked to its predecessor. This chain of blocks ensures the integrity, transparency, and security of the recorded auction transactions. Each block encapsulates vital information that enables the validation of the chain and supports consensus mechanisms.

Each block contains the following fields:

- **Previous Block Hash**: A hash of the preceding block, ensuring the immutability and integrity of the chain.
- **Block Hash**: The unique hash of the current block (composed of the SHA256 of the concatenation of: previous block hash, timestamp, nonce and transactions).
- **Timestamp**: A record of the exact moment when the block was created, providing a chronological order of transactions.
- **Nonce**: A number that miners adjust to satisfy the difficulty condition in Proof-of-Work.
- **List of Transactions**: A set of auction-related operations (such as bids or auction creations) signed with public key cryptography.
- **Miner Signature**: A digital signature generated by the miner to authenticate the block creation.

- **Miner Public Key**: The miner's public key, which can be used to verify the miner's signature.

Blocks are linked through their hashes, forming a secure chain. Our implementation provides methods for calculating hashes, mining blocks, and verifying block integrity. All blocks and transactions undergo cryptographic verification before being added to the chain.

The blockchain implementation includes persistent storage using JSON, allowing the blockchain state to be saved and reloaded. This ensures that node restarts don't lose the transaction history.

A. Proof-of-Work

Our Proof-of-Work (PoW) implementation follows the established principles from Bitcoin, requiring miners to find a nonce that produces a block hash with a specified number of leading zeros. This process ensures computational effort is required to add new blocks, protecting against spam and Sybil attacks.

In our implementation, we set a difficulty of 4, requiring block hashes to begin with four zeros. The consensus engine verifies that mined blocks meet this requirement and that they contain at least one valid transaction. The validation process also checks that the block's previous hash correctly links to the latest block in the chain.

B. Proof-of-Reputation

Our Proof-of-Reputation (PoR) consensus mechanism leverages the historical behavior and trustworthiness of participants to secure the network. The implementation evaluates a node's past actions through a trust score that must meet or exceed a threshold of 0.7 to propose blocks.

While PoW focuses on computational effort, PoR places emphasis on demonstrated reliability and honest behavior. This makes the system more efficient by giving greater influence to well-behaved peers and reducing the environmental impact associated with computational puzzles.

The reputation scores decay over time, requiring continuous positive participation in the network. Our implementation also includes automatic consensus switching - when a node's trust score rises above 0.7, the system can automatically switch from PoW to PoR for that node, and vice versa if the score drops below the threshold.

IV. Kademlia & S/Kademlia

Our implementation starts with the basic Kademlia protocol as described by Maymounkov and Mazieres, then extends it with S/Kademlia security improvements. The core of our implementation includes deterministic node ID generation based on public keys, XOR-metric based routing with k-buckets enhanced with trust-based node selection, and secure messaging to ensure message integrity and authenticity through cryptographic signatures.

A key security innovation in our implementation is the trust-weighted distance metric. When finding closest peers, our system considers both the XOR distance and the peer's trust score using a balance factor. This approach ensures that malicious nodes cannot easily position themselves strategically in the ID space to isolate targets, as both distance and trustworthiness are considered.

A. Node structure

A Kademlia/S-Kademlia node in our implementation is composed of several key components:

- **Node ID**: A unique identifier derived from the node's public key using SHA-1 hashing.
- **IP Address**: The network address where the node can be reached.
- **Port**: The communication port used by the node.
- **Public Key**: Used for secure communication and identity verification.
- **Routing table**: Contains a list of buckets organized by XOR distance.
- **Storage**: A persistent system for blocks and DHT values.

Our implementation includes key security checks to prevent IP address spoofing and detect node ID conflicts.

B. Operations

Our implementation supports the standard Kademlia operations plus security enhancements:

- **JOIN_NETWORK**: Allows a node to join the network
- **PING**: Checks if a node is active and reachable.
- **FIND_NODE**: Locates the K closest nodes to a target ID.
- **FIND_VALUE**: Retrieves a value by key or returns closest nodes.
- **STORE**: Stores a block in the appropriate node(s).
- **GET_ROUTING_TABLE**: Retrieves routing table information.
- **GET_BLOCKCHAIN**: Retrieves the blockchain from a peer.
- **GET_STORAGE**: Retrieves stored blocks from a node.

All operations use secure messaging with TLS encryption to ensure confidentiality and integrity. Additionally, node authenticity is verified by checking that the node ID is correctly derived from the public key.

C. Resistance against Sybil and Eclipse attacks

Our implementation includes several security mechanisms to combat Sybil and Eclipse attacks:

- **Cryptographic ID Generation**: Node IDs are derived from their public keys, which are computationally expensive to generate, limiting the ability to create numerous identities.
- **Trust-based Admission**: Our peer manager implements trust thresholds (0.2) that nodes must exceed to be added to routing tables.
- **Trust-weighted Routing**: Our routing algorithm balances XOR distance with peer trust scores, making it harder for attackers to isolate nodes.

- **ID-IP Binding Validation**: The system detects and rejects attempts to use the same IP/port with different node IDs.

These mechanisms collectively make it difficult for attackers to insert numerous malicious nodes or isolate specific targets.

V. Peer-to-peer (P2P)

Our P2P communication layer is built on Netty, a high-performance asynchronous event-driven network framework. The key components include a P2P server for handling incoming connections with TLS encryption, a P2P client for managing outgoing connections to other peers, and a message handler that processes various message types and coordinates responses.

For secure communications, we implement TLS encryption using utilities that create server and client SSL contexts. Each node generates a self-signed X.509 certificate derived from its RSA key pair. This ensures that all network traffic is encrypted and that nodes can authenticate their peers.

The network engine coordinates higher-level operations like bootstrapping, message broadcasting, and blockchain synchronization. It also manages reputation tracking and consensus algorithm selection based on peer behavior.

VI. Auction

Our auction system implements a traditional English auction model with three primary transaction types:

- `START_AUCTION`: Creates a new auction with item details and minimum bid.
- `PLACE_BID`: Submits a bid on an active auction.
- `CLOSE_AUCTION`: Finalizes an auction and determines the winner.

The auction manager coordinates these operations and ensures that all actions are validated before being executed. Key validation checks include:

- Bids must exceed the current highest bid.
- Only the auction creator can close an auction.
- Bids on closed auctions are rejected.

When an auction is closed, the winning bid is recorded on the blockchain as a transaction, providing a permanent and tamper-proof record of the outcome. Our implementation includes automatic consensus selection based on bidder reputation. This allows the system to adapt to the trustworthiness of participants, using more efficient consensus mechanisms when dealing with highly trusted peers.

A. "Publishers/Subscribers"

Our auction system implements a basic publish-subscribe pattern to facilitate real-time updates:

- **SUBSCRIBE**: Allows peers to register interest in auction events.
- **PUBLISH**: Broadcasts auction events to subscribed peers.
- **NOTIFY**: Delivers event notifications to subscribers.
- **UNSUBSCRIBE**: Removes peers from topic subscription lists.

This enables real-time auction updates, such as new bids or auction closures, to be delivered efficiently to interested parties without requiring constant polling.

VII. User Interface and System Functionality

Our system provides a comprehensive menu interface offering 20 distinct operations that showcase both the auction functionality and the underlying networking and consensus mechanisms:

A. Auction Operations

- **Create auction**: Allows users to initialize a new auction with item details and minimum bid.
- **Place bid**: Enables users to submit bids on active auctions.
- **Close auction**: Finalizes an auction and determines the winner.
- **View open auctions**: Lists all currently active auctions in the system.
- **View bids and winner**: Shows the bid history for an auction and the winner if closed.

B. Blockchain Operations

- **View local blockchain**: Displays the current state of the blockchain stored locally.
- **Test PoW**: Demonstrates Proof-of-Work consensus by mining a new block.
- **Test PoW with failure**: Shows validation rejection of an improperly mined block.
- **Test PoR**: Demonstrates Proof-of-Reputation consensus when trust thresholds are met.
- **Test PoR with failure**: Shows consensus rejection when reputation is insufficient.

C. Network Operations

- **View known peers**: Lists all peers in the node's routing table.
- **Send PING**: Tests connectivity to another node.
- **Send STORE**: Stores data in the distributed hash table.
- **Send FIND_NODE**: Searches for the k-closest nodes to a target ID.
- **Send FIND_VALUE**: Retrieves a value from the distributed hash table.

D. Security Testing Operations

- **Simulate Sybil attack**: Tests the system's resistance to identity spoofing attacks.
- **Simulate Eclipse attack**: Tests the system's ability to prevent node isolation attacks.

E. Pub/Sub Operations

- **Subscribe to a topic**: Registers interest in receiving updates on a specific topic.
- **Publish to a topic**: Sends a message to all subscribers of a topic.
- **Unsubscribe from a topic**: Removes a subscription from a topic.

This comprehensive interface allows for thorough testing and demonstration of the system's capabilities, from basic auction functionality to advanced security features.

VIII. Security Implementation Details

Our system implements several security mechanisms that deserve detailed examination:

A. TLS Implementation

All peer-to-peer communication is encrypted using TLS. Each node generates a self-signed certificate using BouncyCastle. This prevents eavesdropping and ensures the confidentiality and integrity of all network traffic.

B. Trust Scoring Mechanism

Our reputation system maintains dynamic trust scores for each peer based on their behavior. Trust scores include a time decay factor, ensuring that inactive peers cannot maintain high reputation indefinitely. The scores are used to influence routing decisions, peer admission, and consensus mechanism selection.

C. Cryptographic Key Management

Our key store utility manages RSA key pairs for each node, storing them securely and providing methods for key retrieval. This allows nodes to maintain persistent identities across restarts while protecting private keys from exposure.

IX. Attack Simulation

To evaluate the robustness of our system against common threats in decentralized networks, we implemented a comprehensive attack simulation framework. This framework, contained in the AttackSimulator class, allows for controlled testing of the system's resilience against two primary attack vectors:

A. Sybil Attack Simulation

The Sybil attack simulation creates multiple malicious nodes with artificially crafted identities that attempt to join the network:

- The simulator generates a configurable number of attack nodes (default: 10)
- Each node receives a randomly generated ID
- The attack nodes attempt to join the network simultaneously
- Our defense mechanisms verify node identity through public key validation
- Trust threshold enforcement blocks nodes without proper cryptographic identity proofs

Through extensive testing, we observed that our cryptographic identity verification successfully blocks Sybil nodes that lack valid public key to node ID mappings, while the trust threshold mechanism prevents infiltration by newly created identities.

B. Eclipse Attack Simulation

The Eclipse attack simulation tests the network's resistance to targeted isolation of specific nodes:

- The simulator identifies a target node in the network
- It creates multiple malicious nodes that attempt to surround the target in the routing table
- These nodes connect from the same IP address but with different port numbers
- Our trust-weighted routing algorithm prevents the isolation by considering both XOR distance and trust scores
- The IP address restriction mechanism (maximum 3 peers per IP) further limits the attack capability

Results showed that our trust-weighted routing mechanism significantly improves resistance to Eclipse attacks compared to standard Kademlia routing, with the success rate decreasing further as the attacker nodes' trust scores decay due to malicious behavior detection.

X. Publish-Subscribe System

To support real-time event notification for auction activities, we implemented a distributed publish-subscribe (pub/sub) system built on top of our Kademlia DHT. This system allows peers to subscribe to topics of interest and receive updates without continuous polling.

A. Architecture

Our pub/sub system is primarily implemented in the KadStore class with the following components:

- **Topic Registry**: A concurrent hash map that stores topic subscriptions, mapping topic names to sets of subscribed node IDs
- **Message Cache**: A temporary store for recently published messages to handle delivery to nodes that reconnect
- **Event Types**: Support for different event categories including auction creation, bid placement, and auction closure

B. Key Operations

The pub/sub system supports four primary operations accessible through the user interface:

- **SUBSCRIBE**: Registers a node's interest in a specific topic, such as a particular auction ID or general auction categories.
- **PUBLISH**: Broadcasts a message to all subscribers of a given topic. The message includes the topic identifier, payload data, and sender information.
- **NOTIFY**: Delivers event notifications to subscribers when specific auction events occur, such as new bids or auction closures.
- **UNSUBSCRIBE**: Removes a node from the subscription list for a particular topic when they no longer wish to receive updates.

C. Implementation Challenges

Several challenges were addressed in our pub/sub implementation:

- **Decentralized Topic Management**: Unlike centralized pub/sub systems, our implementation distributes topic registries across nodes using the DHT.
- **Message Reliability**: To handle node disconnections and reconnections, our system maintains a temporary cache of recent messages for each topic.
- **Duplicate Message Prevention**: Each message receives a unique identifier combining the sender ID, topic, and timestamp to prevent duplicate delivery.

D. Integration with Auction System

The pub/sub system is tightly integrated with the auction functionality:

- When a new auction is created, an event is automatically published to the "auctions" topic
- Bid placements trigger publication to the specific auction's topic
- Auction closure events notify all subscribers about the final result
- Users viewing an auction are automatically subscribed to its updates

This functionality significantly enhances the user experience by providing real-time updates on auction activities without requiring manual refreshing or constant API polling.

XI. Conclusion

In this paper, we developed a distributed system capable of managing a public ledger for auctions, integrating key components such as a blockchain, a secure P2P communication layer based on S/Kademlia, and a simple auction mechanism. Throughout the development process, we encountered several challenges, most of which we were able to successfully overcome.

The key innovations in our implementation include:

- Trust-weighted routing for improved security against Eclipse attacks
- Automatic consensus switching based on reputation scores
- TLS-secured communications for all peer interactions
- Deterministic node ID generation from public keys
- Time-decaying reputation scores with influence on network operations
- Comprehensive attack simulation capabilities for security testing
- Integrated publish-subscribe system for real-time event notifications

References

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[2] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper, 151:1–32, 2014.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems(TOCS), 20(4):398–461, 2002.

[4] João Sousa, Eduardo Alchieri, and Alysson Bessani. State machine replication for the masses with bft-smart. 2013.

[5] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In International Workshop on Peer-to-Peer Systems, pages 53–65. Springer, 2002.

[6] Ingmar Baumgart and Sebastian Mies. S/kademlia: A practicable approach towards secure key- based routing. In Parallel and Distributed Systems, 2007 International Conference on, pages 1–8. IEEE, 2007.

[7] Francis N. Nwebonyi, Rolando Martins, and Manuel E. Correia. Reputation based approach for improved fairness and robustness in p2p protocols. Peer-to-Peer Networking and Applications, Dec 2018.

[8] Bittorrent publish/subscribe protocol. http://bittorrent.org/beps/bep_0050.html. Accessed: 2021-02-15.

[9] Yu, Jiangshan, David Kozhaya, Jeremie Decouchant, and Paulo Esteves-Verissimo. "Repucoin: Your reputation is your power." IEEE Transactions on Computers 68, no. 8 (2019): 1225-1237.

[10] Neto, Nuno, Rolando Martins, and Luís Veiga. "Atlas, a modular and efficient open-source BFT framework." Journal of Systems and Software 222 (2025): 112317.