Public Ledger for Auctions - Assignment Report
Systems and Data Security

Carolina Trindade	João Neto	Pedro Jorge
up201706077	up201605883	up201706520

June 5, 2022

Contents

1 Introduction

The goal of this assignment is to develop, using Java programming language, a decentralized public ledger that is capable of storing auction transactions by implementing a public non-permissioned blockchain. The implementation should be divided in three parts: the distributed ledger, the secure P2P and the auction mechanisms.

The secure ledger should be modular and support proof-of-work (PoW) as the core consensus. The P2P layer, that broadcasts the necessary data to support the blockchain, should ideally: implement Kademlia as its routing mechanism, be resistant to Sybil and Eclipse attacks, and implement trust mechanisms. The auction system, capable of supporting sellers and buyers using a single attribute auction following the English auction standard, should: save transactions in the aforementioned distributed ledger and work on a publisher / subscriber basis, built on top of Kademlia to support auctions.

The present report does not provide a detailed description of the implementation of the system, but instead focuses on explaining the overall inner-working of the system, as well as the decisions that rule its implementation.

2 Architecture and Implementation

2.1 Distributed Ledger

The distributed ledger that was developed takes advantage of the implementation of a public non-permissioned blockchain in various ways.

Transactions are recorded only once as blocks of data and each block is connected to the previous and subsequent ones, creating a chain of transactions that are chronologically ordered and preventing any transaction to be tampered or the insertion of malicious ones between two others. Additional blocks help to verify the previous ones and, consequently, the entire blockchain. Therefore, the blockchain is immutable and tamper-resistant, so the ledger of transactions can be trusted. In order to validate a new block it must go through a mining process, in which a proof-of-work is accomplished by finding a nonce for which the hash of the block header abides to a certain challenge with predefined difficulty. This difficulty may be changed in order to guarantee a desired average mining time.

Wallets store a Public-Private Key Pair, used to sign and verify its transactions. Balance is not stored based on an account model, but on a UTXO model, where the balance of each wallet is the sum of all the unspent outputs to its public key. When creating a transaction, all the unspent outputs must be spent and originate new outputs, generally at least two as the wallet creates an output of the desired amount to the desired target public key, and a second output with the remaining balance to itself. This majorly solves the double spending problem, as spent outputs are not accepted in new transactions.

2.2 Secure P2P Network

The peer-to-peer network is based on the Kademlia protocol [1] for a distributed hash table. The main features of this protocol is that configuration information spreads automatically as a side-effect of key lookups, and the algorithm used to store other nodes' contacts in each node resists certain basic denial of service attacks.

Before getting into the Kademlia protocol implementation, our first added safety measure provides some mitigation against Sybil attacks by making nodes spend time and energy resources in creating the node Id. This is done through a similar mechanism to the proof-of-work in the blockchain, as nodes have to find a nonce that when digested with their address and port results in a hash that abides to a certain challenge. This idea of requiring the prospective (joining) node to solve cryptographic puzzles is also presented in [2].

The NodeId-based routing algorithm groups close nodes in the so-called "k-buckets". By keeping at least one node in each bucket, if any exists, the algorithm is able to reach any existing node by its id. This notion of closeness is also important for the four main Remote Procedure Calls (RPCs) that compose the protocol. In each "k-bucket" the nodes are sorted by time last seen, and each k-bucket has its maximum size defined by one of two important system-wide parameters. These two parameters are known as "k" for the replication parameter that defines, among other things, the maximum bucket size and "alpha" for the concurrency parameter, that defines how many operations are done concurrently. The parameter "alpha" leads us to a second added safety measure, this time against Eclipse attacks: increasing its value increases the chance that valid nodes are queried during an operation, mitigating the risk of a malicious entity fully controlling the network view of a certain node.

As for the differences to the standard Kademlia protocol, we implemented two optimizations that are presented in [1]: when inserting a new node to a full bucket, instead of pinging the first node and replacing it if it doesn't respond, in order to reduce the amount of messages in the network we keep the new node in a replacement cache for future replacement, when we actually need to contact the first node and it does not answer; when removing a node from the bucket, we don't actually remove it but instead increase its staleness count, until a predefined limit, which accounts for nodes that may leave the network and return at a later time.

In addition to the four essential RPC's presented in the Kademlia protocol, we implemented two new ones: the procedure to send a message to a certain NodeId and the procedure to broadcast a message to the network. The send message procedure is quite trivial, as it simply performs a node lookup for the target and then sends the message. The broadcast procedure is slightly more complex: in order to avoid sending duplicated messages, each nodes keeps a list of the hashes of all broadcast messages they receive, and if a new one is in that list it is simply dropped. Otherwise we follow an approach outlined in [3], where we choose a predefined number of ran-

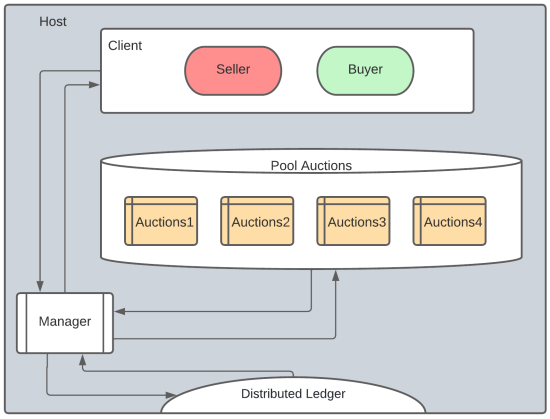

Figure 1: Auction Diagram

dom nodes per k-bucket and send them a broadcast message with the same information, and so on. Each node performs this only for the k-buckets at one lower depth than the iteration before.

2.3 Auction Mechanism

The figure 1 is a diagram that describes the architecture implemented for the auctions mechanism. The auction system is based on the English auction format. Any host is able to create an auction and place bids that are shared on the network. The client is responsible for creating auctions and placing bids, and represents buyers and sellers of available items. A local manager is an entity responsible for managing auctions on the host. The manager is responsible for all operations required for auctions. This unit uploads the auctions to the network and releases them when the customer trades locally. A structure, Pool Auctions, is used to store these auctions and bids on the network.

3 Conclusion

As a summary of what was done, we implemented a secure ledger that uses proof-of-work as its core consensus mechanism. This ledger is shared through a peer-to-peer network layer which implements Kademlia as its routing protocol, and provides some increased resistance to Sybil and Eclipse attacks. On top of this we provide a working auction system which saves

transactions in the aforementioned blockchain, supported by a publisher-subscriber model built on top of the Kademlia network.

Unfortunately we were not able to implement trust mechanisms in the peer-to-peer network, which would further increase resistance to Eclipse attacks, nor do we support proof-of-stake as the consensus mechanism for our blockchain.

References

[1] Maymounkov, P., Mazières, D. (2002). Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds) Peer-to-Peer Systems. IPTPS 2002. Lecture Notes in Computer Science, vol 2429. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45748-8_5

[2] Pecori, Riccardo. (2015). S-Kademlia: a Trust and Reputation Method to Mitigate a Sybil Attack in Kademlia. Computer Networks. 94. 10.1016/j.comnet.2015.11.010.

[3] Rohrer, Elias & Tschorsch, Florian. (2019). Kadcast: A Structured Approach to Broadcast in Blockchain Networks. 199-213. 10.1145/3318041.3355469.