Implementation of auctions in the context of ByzCoin

Makléwa Téta Diana Harena

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Master Semester Project

June 2019

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Allen Jeffrey Richard
EPFL / DEDIS

1 Abstract

In this project, I designed and implemented an open ascending auction system using the Byzcoin-blockchain and the Cothority framework. The designed client application works well with the Byzcoin service and users through it can set up auctions and place bids. The conducted evaluation of the system shows that the system can create more than 200 simultaneous auctions per 20s and allows a maximum of 180 simultaneous bids per 20s.

Keywords: Decentrilized auction, ByzCoin, Open ascending auctions

2 Terminology

General terms and abbreviations

Auction
A method to sell products in which a seller proposes goods (or services) for sale and bidders propose the price they are willing to pay for it.

Auctioneer
a person or server who conducts auctions by accepting bids and declaring goods sold

Cothority
Collective authority. A set of servers running protocols in Cothority

Conode
Individual servers in a cothority

DARC
Distributed Access Right Controls. It's a specific entity of Byzcoin. It is used for access control. Each Darc has a set of rules that define a pair of action/expression that need to be fulfilled to execute any instruction on the instances governed by that Darc.

Reserve price
A reserve price indicates the lowest price acceptable by a seller to sell an item in an auction.

Smart contract
A self-executing program, run on a blockchain, that perform predefined tasks defined in their code

Contents

3 Introduction

As demonstrated by online auctions systems like eBay or Amazon, online auctions or e-auctions have become a popular way of selling goods (168 millions of active buyers in 2018 on eBay [1]). They present several advantages as for instance not requiring the physical presence of buyers and sellers at a central location. However, they also pose a significant problem which is how to guarantee to bidders that the auctioneer server is not corrupted and does not manipulates the auctions proceedings. For example by inserting fake bids, stealing payments or awarding the item to someone other than the legitimate winner.

One solution would be to have multiple auctioneer servers jointly managing the auction and computing the outcome. In other words, decentralising the auctioning process. This can be achieved by running the auction on blockchains with smart contracts, which would at the same time provide some transparency to the auction.

The Cothority project [2] provide a framework (Cothority) for experimenting with distributed, blockchain-related, decentralized applications. This project aimed at using it to experiment a blockchain-based decentralised e-auction system. The idea was to build an e-auction system running on the Byzcoin service of Cothority. The purpose of this report is to provide a description of the design, implementation and evaluation of the system I built.

4 Background

This section introduces Byzcoin.

4.1 Overview of Byzcoin implementation

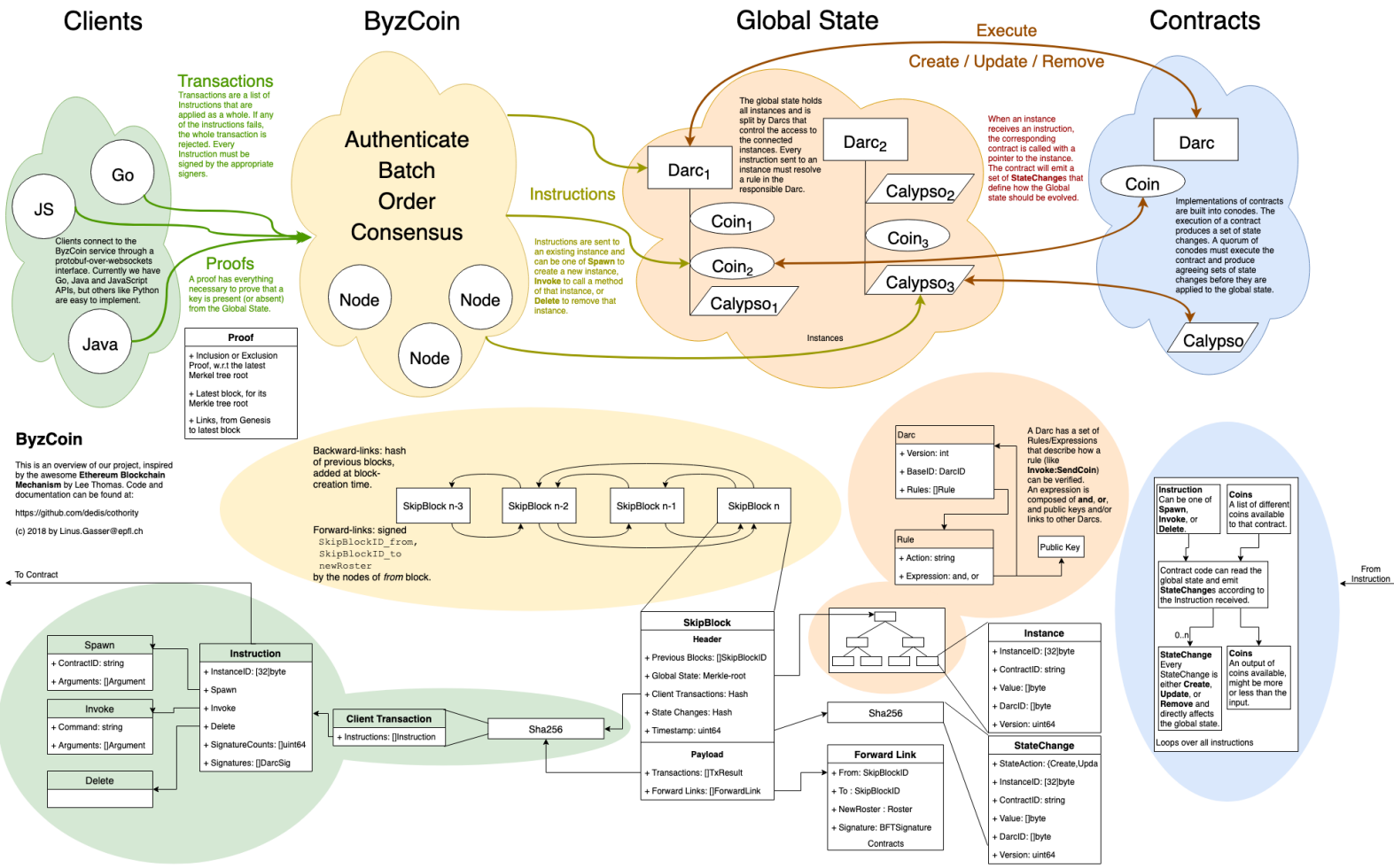

Figure 1: overview of Byzcoin implementation [3]

4.2 Byzcoin smart contracts

Byzcoin works with smart contracts [4]. They define how instructions received from a client should be processed. In Byzcoin, a client instruction can be of three types: spawn which instantiates a contract, invoke which updates the state change or delete which delete a targeted instance.

The Byzcoin global state consists of the instances created by the instanciation of the contracts (see figure 1). Each instance maintains information about its instance ID, version and data. An easy interpretation is to think of a contract as a class and the instance as an object instantiated from that class.

5 System design

As stated in the introduction, in this project, I had to built an e-auction system which operate thanks to the Byzcoin service. Therefore, the first step was to choose a type of auction and design the Byzcoin smart contract which would run it.

I chose open ascending auctions. In open ascending auctions, bidders place bids of progressively higher amounts, aiming to outbid each other. The winner is the one who places the highest bid by the end of the auction. Once the type of auction chosen and its principle understood, I had to think about the types of instructions that the contract should handle and how he would handle them.

I have reached the conclusion that the contract should be able to handle three types of instructions: the one that asks for the creation of a new auction (spawn), the one that asks to place a bid (invoke:bid) and the one that asks to close an auction (invoke:close).

Upon receiving the first type, the contract must create a new auction instance in the global state that will hold the auction data (good description, seller account, highest bid, highest bidder account, auction state).

Upon receiving the second type, the contract must evaluate whether the bid is valid or not before updating the auction instance data (highest bid and highest bidder). An invalid bid (lower or equal to zero or lower than the highest registered bid), must be rejected and the data in the instance left unchanged.

Finally, upon receipt of the third type, the contract must label the auction as closed, transfer the coins to the seller account and refuse any later bid. The diagram below illustrates the expected behavior.

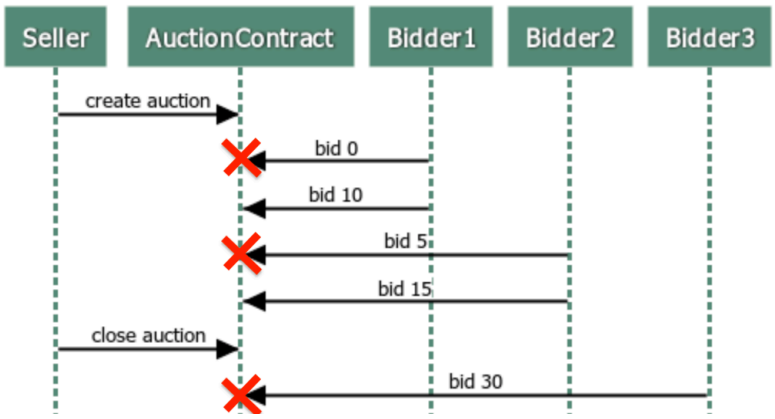

Figure 2: auction contract behaviour

6 Implementation

6.1 Auction contract

Following my design, I created three methods, Spawn(), Invoke()-bid and Invoke()-close in the contract file. I also created a structure to be hold the auction instance data (good description, seller coin account instance ID, highest bid, highest bidder coin account instance ID, auction state).

To begin, I defined these methods for dealing with the simple case of an auction without a reserve price. It is the case described and illustrated in the design (see figure 2) but with a slight difference. In my implementation, when a bidder places a bid, the amount is deducted from his account. First, there is an invoke:fecth instruction that fetches the bid amount of coins from the bidder account and pass it as an array of coins to the invoke:bid instruction. The bid amount is taken from that array. This allows to guarantee some accountability of the bidder to the seller and prevent the bidder to withdraw. The refunding process in the Invoke()-bid method is described in the figure below.

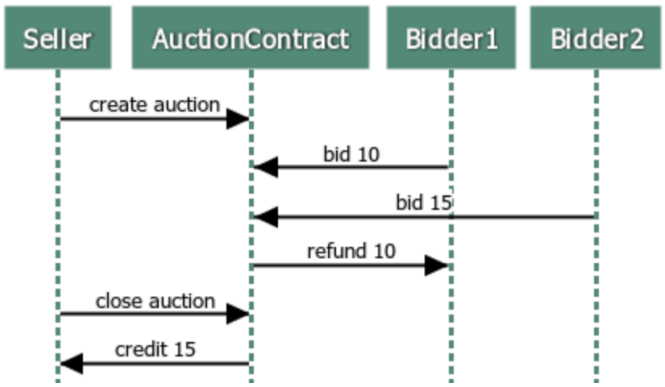

Figure 3: Invoke bid

Next I modified them so that they also take into account the case of an auction with a reserve price (new field in auction data structure).

Since the reserve price must remain secret until the end of the auction and the data in Byzcoin public (anyone can see the data), I had to look for a way that would allow to keep the reserve price secret until the end of an auction.

I chose to use a commitment scheme. The applied principle is that the reserve price sent at the auction creation won't be the real value but the salted hash of it using sha256. At the end of the auction, the salt and the reserve price are sent to the auction contract to reveal the reserve price if the hash of these data match the one sent. The commit reveal is performed by the Invoke()-close method. The whole treatment of the method is described in the following figure.

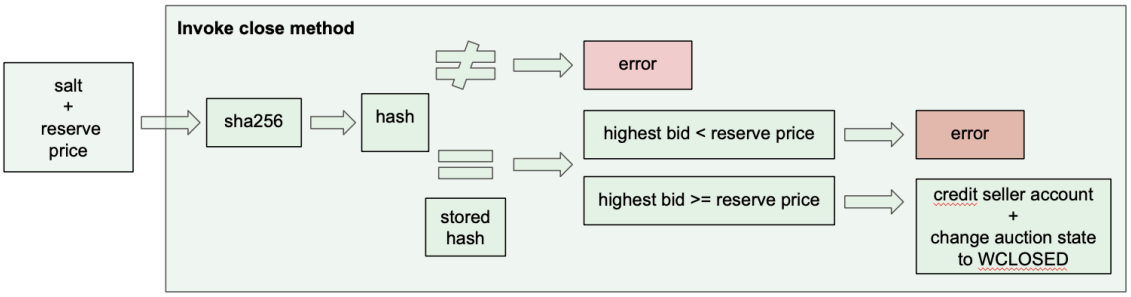

Figure 4: overview of the auction contract closing method with reserve price

Since the method Invoke close needs to receive the verification data (reserve price and the salt) in order to close the auction, I also define the corresponding structure to be given as an argument to the invoke:close instruction.

Next, I created a new method to handle the case in which a seller wants to drop the auction, Invoke()-drop. In that case, the highest bidder is refunded and the seller account does not get credit. Except that difference, the process is similar to Invoke()-close.

Finally I added a new field in the auction data structure to hold the winning proof of the winner (its public key). This is to ensure the identity of the winner.

At each step of the contract implementation, I performed unit tests to make sure that the contract worked as expected.

6.2 Client application

Having the contract working, I moved on to the development of the client application for communication between the end users and the Byzcoin service. I was asked to built the user interface in an existing application that can already interacts (through an API) with the cothority backend called Dynasent [5]. Dynasent is built with Angular CLI and typescript.

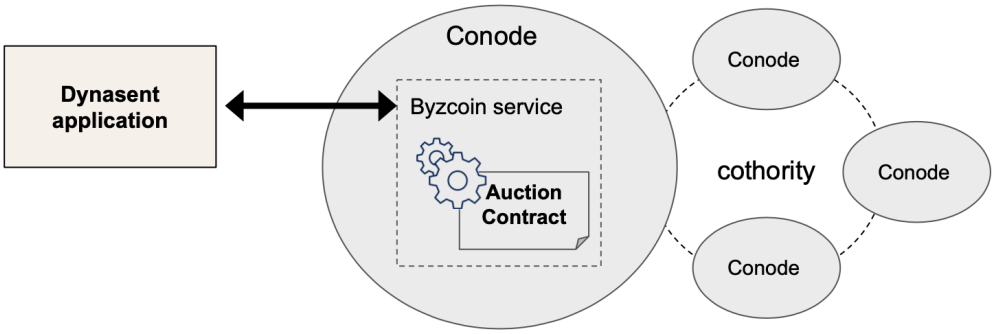

Figure 5: interaction with Byzcoin service

I designed the client application to enable the following functionalities:

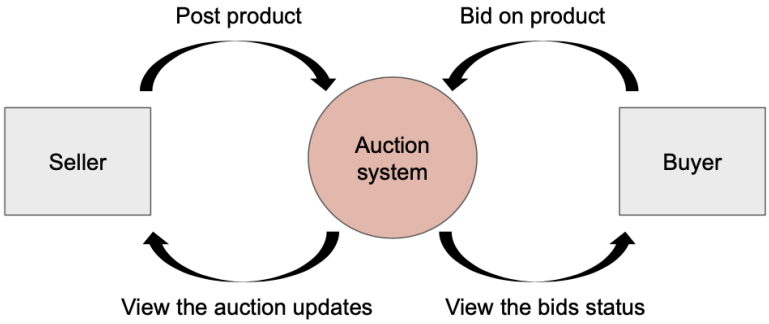

Figure 6: interaction of the users with the system

When a user asks to creates an auction, places a bid, close/drop an auction, the instructions as sent to Byzcoin service which execute the auction contract. Since auction data need to be displayed in the interface, I created auction objects that are tied to the auction instances in Byzcoin and allow to access the auctions data on Dynasent side.

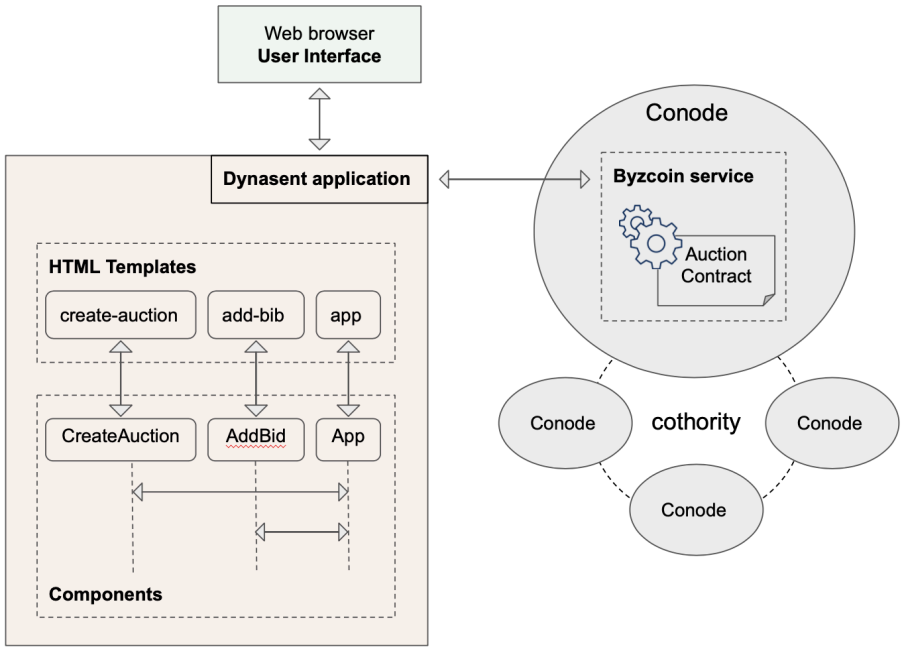

Figure 7: Interaction beetween the differents entities

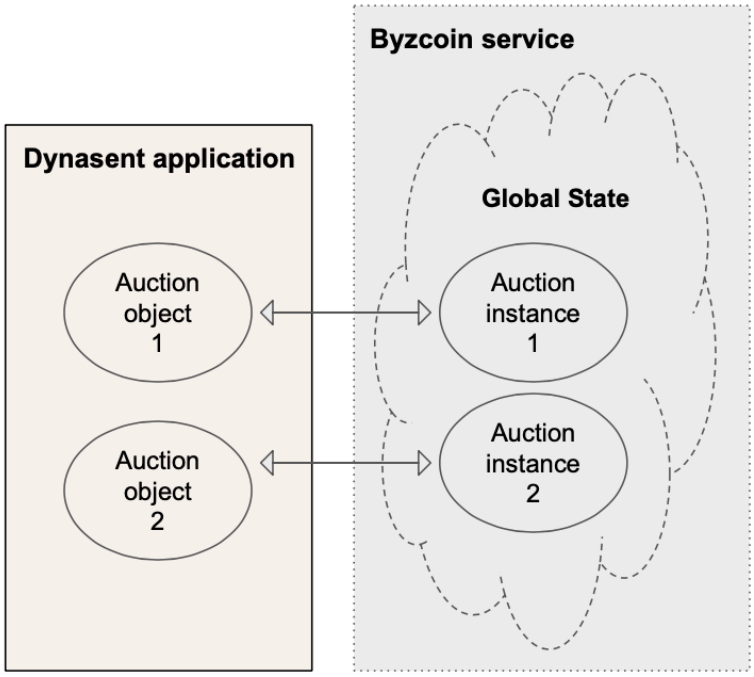

Figure 8: auction object in Dynasent are tied to auction instance in Byzcoin

Following are some screenshots of the user-interface:

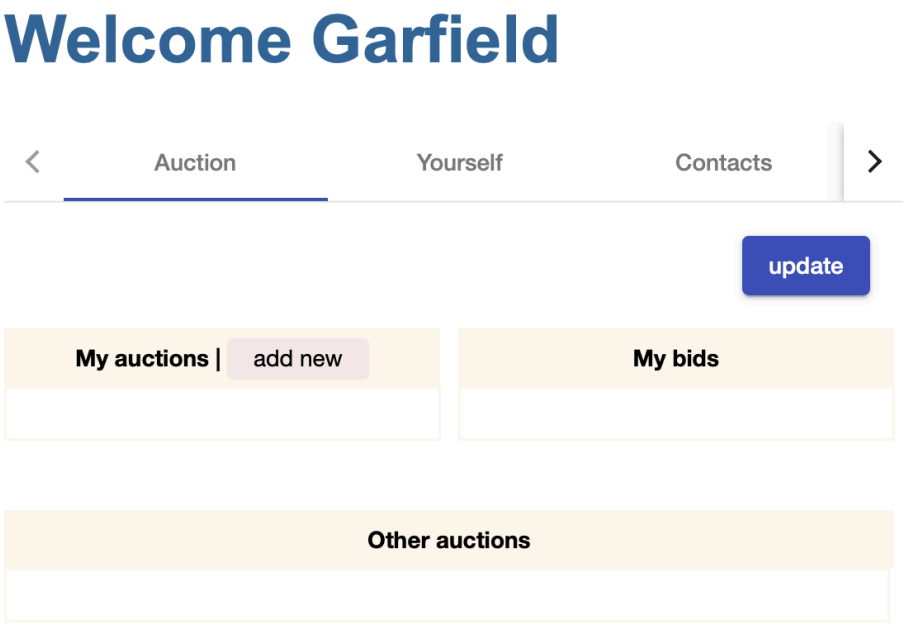

Figure 9: initial screen

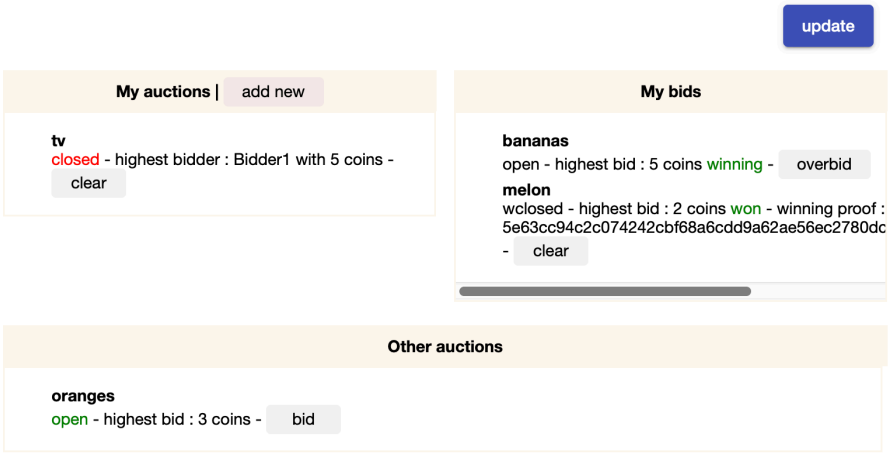

Figure 10: active user screen

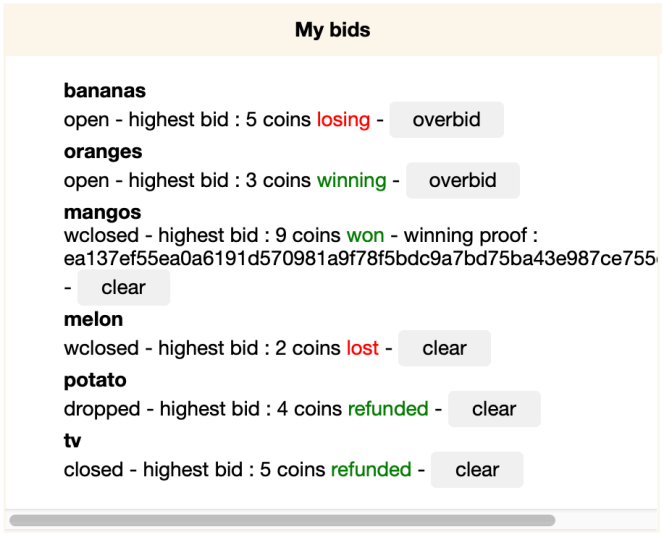

Figure 11: View of bids state

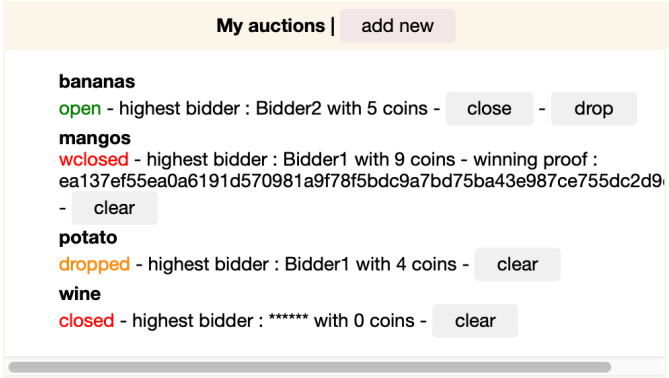

Figure 12: View of auctions state

7 Evaluation

Having successfully developed the client Application, I moved on to the step of evaluating the system. To do so, I ran different simulations using the Onet library [6] and the Deterlab platform [7]. The goal was to measure: the max number of auctions than can be created in parallel during a certain laps of time, the max number of bid than can be placed in parallel during a certain laps of time, the time and bandwidth it takes to concurrently create a certain number of auctions or place a certain number of bids.

For the simulations, I used a cothority of 7 nodes with each conode being on a different server (bpc2133). This way the use of each node would be optimal. I configured a network delay of 100ms to be realistic. I ran each simulation for 20s with a block interval of 1s (which make 20 blocks). For each simulation, the measurements were stored in a csv file and included the wall time (time with the network communication included) and also the system cost calculated in seconds (time during which the system nodes use their CPUs). The results of the simulations are described in the graphs below.

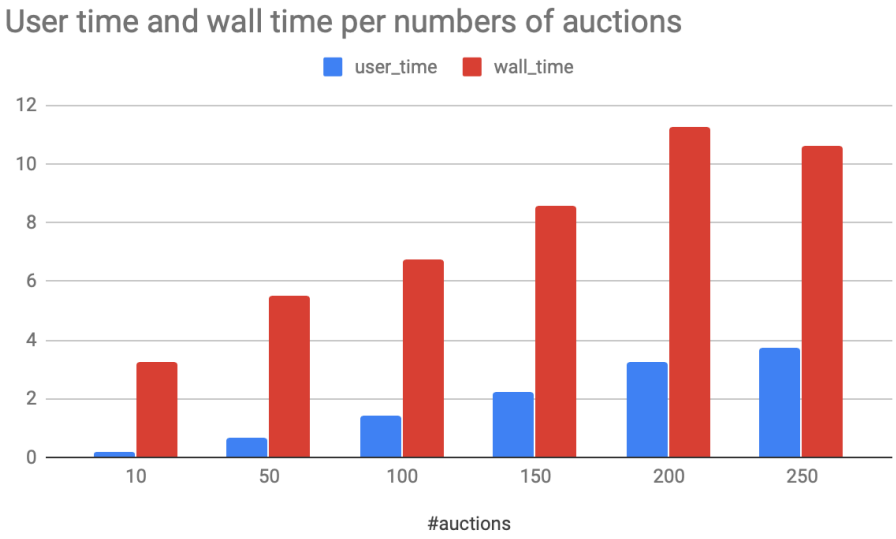

Figure 13: Wall and user time per number of parallels auctions

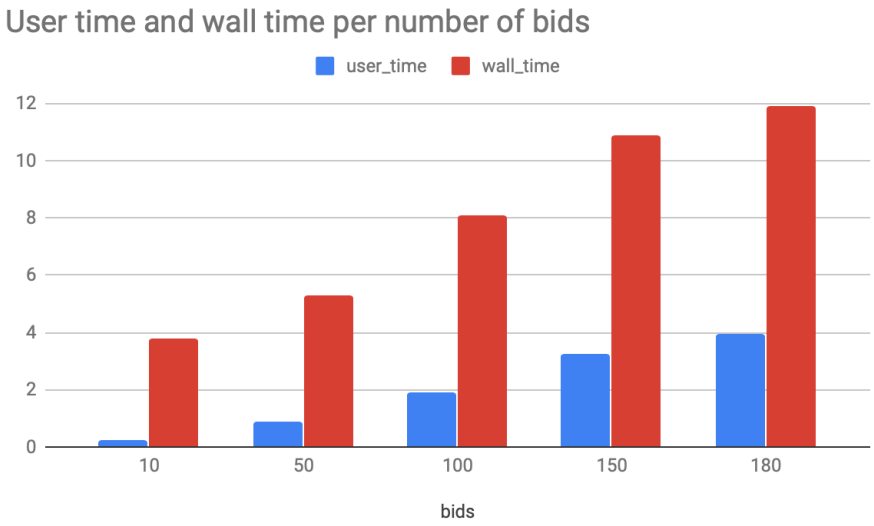

Figure 14: Wall and user time per number of parallels bids

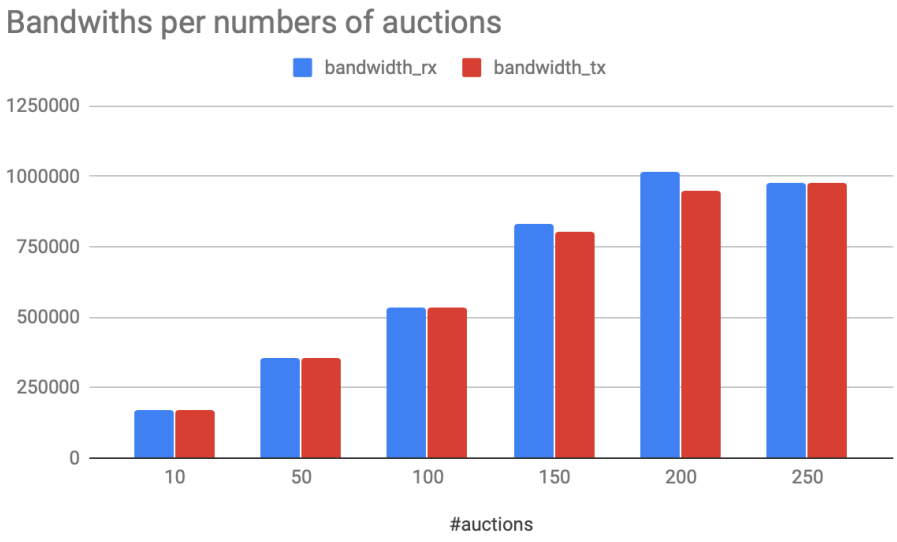

Figure 15: Bandwiths at transmission and reception per number of parallels auctions

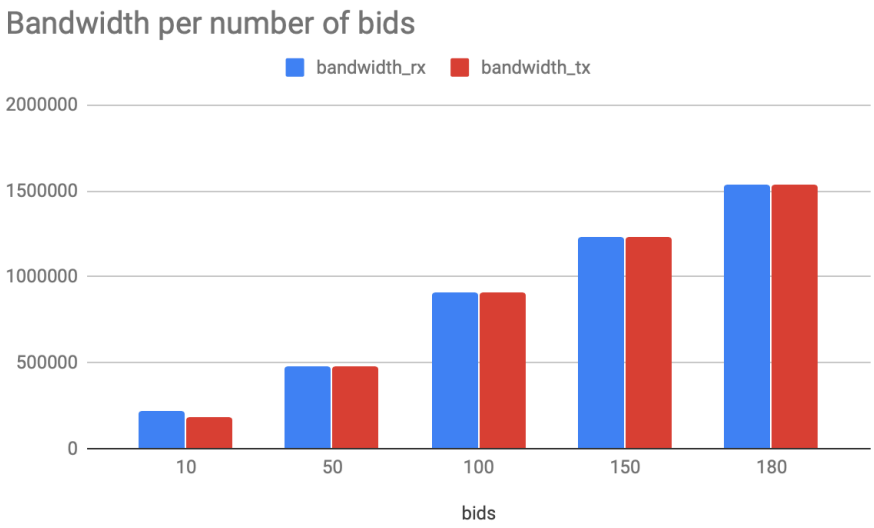

Figure 16: Bandwiths at transmission and reception per number of parallels bids

The results show that:
- as expected, the more transactions there are (auctions or bid),the higher the processing time of the system
- the bandwidth at transmission and reception stay almost constant, if not constant, for a same considered number of auctions or bids
- the max number of auctions that could be created in parallel during 20s is 220 and the maximum number of simultaneous bids per 20s is 180. Upper, the system gives errors and stop. The maximum number of bids is lower because each transaction contains two intructions (fetch and bid).

8 Conclusion and future work

In this project, I implemented open ascending auctions in the context of Byzcoin. The first step has been to create the auction contract. After performing some unit tests to ensure that it worked as expected, I built a client application to communicate with the Byzcoin service. Finally, in order to evaluate the system, I sat up some experiments. The evaluation has demonstrated that the system works well with up to a maximum of 220 auctions per 20s and 180 bids per 20s.

As possible future works, it would be interesting to study how to introduce a limited time for auctions and/or implement other types of auctions. For example sealed-bid auctions. That would be an interesting challenge since it would be necessary to ensure that the bids are kept secret despite the public nature of Byzcoin.

9 Acknowledgement

It has been a great pleasure for me to carry out this project as it helped me develop Go and Typescript programming skills and gain knowledge about blockchains, smart contracts and auctions.

I would like to thank the DEDIS laboratory for welcoming me.

I also address my sincere thanks to my supervisor, Mr. Allen Jeffrey Richard, software engineer at DEDIS Lab, for accepting me on this project, for his guidance throughout the whole project and the trust he has given me.

Thank you also to Mr Gasser Linus, software engineer at DEDIS Lab, for his help with the Dynasent application.

10 References

[1] SMALLBIZTRENDS, There Are 168 Million Active Buyers on eBay Right Now (INFOGRAPHIC), accessed 03.06.2019, https://smallbiztrends.com/2018/03/ebay-statistics-march-2018.html

[2] DEDIS, Cothority, accessed 19.02.2019, https://github.com/dedis/cothority

[3] DEDIS, ByzCoin Figure, accessed 08.03.2019, https://github.com/dedis/cothority/tree/master/byzcoin

[4] DEDIS, Contracts, accessed 04.03.2019, https://github.com/dedis/cothority/blob/master/byzcoin/Contracts.md

[5] DEDIS, Cothority, accessed 11.02.2019, https://github.com/dedis/cothority

[6] DEDIS, Onet, accessed 20.05.2019, https://github.com/dedis/cothority

[7] DETERLAB, accessed 24.05.2019, https://www.isi.deterlab.net/showuser.php?user=18826

11 Repositories

Auction smart contract in Byzcoin: https://github.com/dedis/student_19_auctions

Client application in Dynasent: https://github.com/c4dt/dynasent/tree/student_19_auctions

The README files of the repositories provides a Step-by-step guide for Installation.