FinalTerm Peer2Peer Systems and blockchains

Smart Auctions

Lorenzo Bellomo, 531423

Attached Files Structure

Attached with the project comes the auction files. The chosen auction is the Dutch one.
The final term code consists of three files:

- *dutchAuction.sol*: This file contains the Dutch auction logic. The file is composed by

Notes Common to Both Auctions

Handling of Time Time was handled in the same way between the two auctions. All the durations (except for the one of grace time) are expressed in *number of blocks* and are passed as parameter to the constructor of the auction. Grace time, instead, is constant and "hard coded" in the contract. To respect the specifications of the assignment this value should be around 20/23 in order to respect the request of a time close to 5 minutes for the grace time. This is computed assuming an average mining time of 13/15 seconds per block on the Ethereum blockchain (according to the values in `https://etherscan.io/chart/blocktime`). However, in the actual implementation, the constant value for graceTime is set to 2 in order to ease the burden of testing.
The time passage is implemented in a *lazy* way. What this means is that every time a major function, like *bid()*, is called, the contract checks the current block number, and by confronting it to the first one (the block number when the auction was generated), and by checking the various durations specified, it computes the current phase. So no central entity is required to synchronize the auction (no auctioneer is required to dictate the passing of time).
The main pros of this approach (lazy way), with respect to the one with the auctioneer that dictates the passage of time are:

- *Completely decentralized*: No central entity (auctioneer) is required to make the time pass.
- *Guaranteed time correctness (on average)*: The auctioneer model requires the this entity to make the time flow. This means that he might be late while calling the phase switch (but not early supposing that the contract checks the correctness of the auctioneer calls). This is not the case of the lazy implementation, where the checks are made for each call by the contract.
  The time is guaranteed to be respected on average, with possible fluctuations given by the variability in the mining process. This could have been avoided by using the timestamp of the block of the function call transaction, but this hypothesis was discarded in order to stick to the project requirements and implement time with respect to number of blocks.

Instead the main cons of this choice are:

- *Events variability*: Both of the auction implementations emit events related to the passage of time. Those events are fired whenever a phase switch is recognized. Since the phase switch process is lazy, the events may fire late with respect to the actual time in which the phase switched. This issue is inevitable in this kind of implementation, and it can be eased by providing a special method, only accessible to the owner of the auction, that checks the current phase of the auction and updates the state of the contract if enough time has passed. This method is *checkIfAuctionEnded()* for the Dutch Auction and *updateCurrentPhase()* for the Vikrey one.

- *Events cost*: The cost of those events emissions (which is very high with respect to normal EVM instructions) are in the general case at the expenses of the users (the bidders), which fire those events by executing the main methods of the contracts. The same ”solution” of the first point is adopted, but this problem is evident, particularly when dealing with ”almost empty auctions”.

Dutch Auction

Testing

Vikrey Auction

Testing