A Peer-To-Peer Auction System

Project 17: Devrim Baran Demir, Ammar Alsaeed, and Serge Kotchourko

Department of Service Computing

1 Introduction

Auctions have been a part of human culture as early as 500 B.C. and have been used to sell various goods and services in various ways [2]. A common way of auctioning goods in the Netherlands is through the Dutch auction, in which the auctioneer starts with a high asking price and lowers it until a bidder accepts the price. The most common auction type is the English auction, where bidders submit increasing bids until no one is willing to bid higher. The highest bidder wins the auction and pays the amount they bid. This form of auction can still be seen today in the form of eBay, or reverse auctions such as to some extent on Amazon (where sellers compete for the lowest price on the platform).

1.1 Problem Statement

Most auctions are held in a centralized manner, either in the physical or digital world. Especially in the digital world, this has several disadvantages. The auctioneer must have resources (e.g. servers, network, volume) and failure mechanisms in place to conduct auctions. Considering the growth of a platform, a decentralized approach removes the need to provide such infrastructure. Such approaches are already in use, especially in the energy market [5, 3, 6] and are continuously improved [4, 1].

1.2 Proposition

We propose a distributed peer-to-peer forward auction system, mimicking the real-world process closely, where the user can take part in auctions and auction off items, while in return contributing with its computational resources. These resources are delegated to replicate the auction system, aiding in holding auctions from other users.

In the following sections, we will introduce and describe our implementation[1] of such a system. Section 2 gives an intuition of what functionalities the system exhibits. Section 3 highlights the discrete implementation decisions of the system, as well as which technical requirements are met (Section 3.2). Finally, in Section 4 we discuss assumptions and shortcomings of the systems and possible changes to mitigate these.

[1] The implementation can be found here: https://github.com/confusedSerge/distributed-systems-project

2 System Overview

This section serves as an overview of the system. Due to the peer-to-peer architecture decision, the system can be naturally divided into two distinct parts, namely the **server** (Section 2.2) and **client** (Section 2.1) with their own respective set of tasks.

2.1 Client Functionality

The client provides an interactive session for a user of the system. As the system mimics the real auction as closely as possible, a user naturally fills the role of being an *auctioneer* or *bidder* in an auction. In this case, however, these roles are not exclusive, and a user can switch freely between them without losing the status of the other. The user acts in the role of a *bidder* analogous to the real bidder, checking current auctions, joining and leaving auctions, and placing bids. As the goal is to mimic the real-world process, bidders are able to miss certain updates on the state of the auction. Furthermore, if the user sets a winning bid and leaves the auction, this user is still considered the winner and is expected to pay the winning bid, just as in the real world. On the other hand, the user can also act as an *auctioneer*, creating new auctions and monitoring his current auctions. An auction in progress can not be stopped by the auctioneer, again mimicking the real world.

2.2 Server Functionality

A server is the backbone of the system, providing the necessary infrastructure to hold auctions and coordinate between replicas in an auction. The server is divided into two parts, the *server* and the *replicas*. A *replicas* is a process subprocess of the server, which is responsible for managing an auction. This is done by coordinating with other replicas in the auction, such that the state of the auction is consistent between replicas. Depending on the replicas' role, it can also be responsible for managing certain aspects of the auction, such as announcing the current state of the auction or finding new replicas. The *server* manages the local replicas, creating new ones for new auctions and removing them when the auction is finished.

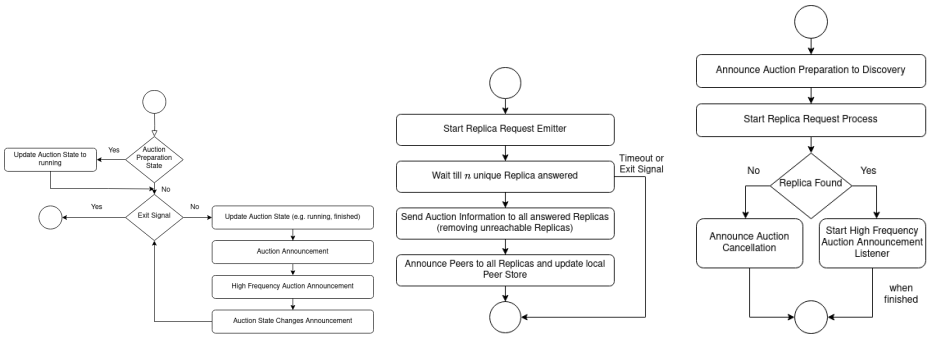
3 Implementation

This section will describe the discrete implementation of the system, going over certain implementation decisions for the system (Section 3.1) and how technical requirements are met (Section 3.2).

3.1 System Implementation

In this section, we will describe implementation design decisions for the system. The Section is divided into three parts, namely **Processes**, **Server**, and **Client**. The **Processes** will describe the background processes used by both the server and the client, which are necessary for the system to work. Sections for **Server** and **Client** go over how they achieve their respective functionalities, as described in Section 2. Finally, the Section for **Communication** gives a brief overview of what communication channels are used by the system.

Processes In this section, we will describe background processes used by both the server and the client, such that the system can work as intended. These processes can be divided into two major groups, managers and listeners. Managers are processes that proactively manage certain aspects of the system, such as the auction manager used by the leader replica in an auction to periodically announce the current state of the auction and check if the auction can progress further. Listeners are processes that listen to certain events and react or store information, such as the auction bid listener, which listens to incoming bids and updates the bidding reliably between replicas. For this, we use shared memory objects between the processes, such that the main and sub-processes have access to the updates and states.



(a) Auction Manager (b) Replication Manager (c) Sub-Auctioneer

Fig. 1: Managers used by the system. These are background processes used by the system to manage certain aspects of the system

Managers The system employs three managers, namely the *Auction Manager*, the *Replication Manager*, and the *Sub-Auctioneer*. The *Auction Manager* (Figure 1a) is used by the leader of the replicas in an auction to manage the auction progress. Initially, the auction manager checks, if the auction is in its preparation process, and if so, changes the state to the running state, as at this point all

preparations are completed and announces the auction to the discovery multicast group. This is done by sending an auction announcement message (cf. Table 1) to the discovery multicast group, containing the current state of the auction and the address to join the auction multicast group. After that, the auction managers continue to its main management loop. This includes checking if the auction can be moved to its next state (e.g. from running to finished). If so, the auction manager announces the new state to the auction multicast group, such that the replicas are aware of the new state. As this needs to be done in a reliable and ordered manner, the auction manager uses the reliable ISIS algorithm to multicast these messages (cf. Paragraph **Communication**). Furthermore, the auction manager periodically announces the current overall state (e.g. casted bids, progress, ...) of the auction to the discovery multicast group as well as with high frequency to the auction multicast group, such that clients outside the group can see the current state of the auction. This is done by using an unreliable multicast, as the recipients do not need to have a consistent view of the state of the auction.

The *Replication Manager* (Figure 1b) is used by the leader of the replicas in an auction to find new replicas if a replica crashes. This is done by emitting a find-replica request message (cf. Table 1) periodically to the discovery multicast group, containing the address of the auction and the port of the reliable UDP unicast to send the response to. When enough replicas are found, the replication manager releases the auction information replication message over a reliable unicast to the new replicas, ensuring that the state of the auction is consistent between replicas. If a message is not possible to deliver, the replica is removed from the list of peers. After this, the peers are announced to the auction multicast group using a reliable ISIS multicast, such that the replicas are aware of their peers. If the replication manager does not receive enough responses, it cancels the search.

The *Sub-Auctioneer* (Figure 1c) is used by the auctioneer (client) to outsource the auction to the replicas. This is done by announcing the auction preparation to the discovery multicast group, following a delegation of the search for new replicas to the replication manager. If not enough replicas are found, the sub-auctioneer announces the cancellation of the auction to the discovery multicast group. Otherwise, the auction should be successfully outsourced to the replicas and the sub-auctioneer should start a high-frequency auction announcement listener (see next paragraph) to monitor the auction.

Listeners Figure 4 gives an overview of the basic steps a background listener process takes to listen to incoming messages and react to them. The system employs five listeners, namely the *Auction Announcement Listener*, the *Auction Bid Listener*, the *Auction Peer Listener*, the *Auction State Listener* and the *Auction Reelection Listener*.

The *Auction Announcement Listener* listens to incoming auction announcements over the discovery multicast group and stores them so that the user can see current auctions. This is also employed by the auctioneer to filter viable auction multicast groups for its auction. Note, that overlapping auction multicast

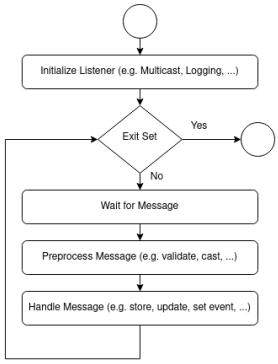


Fig. 2: Overview of the steps a background listener process takes to listen to incoming messages and react to them.

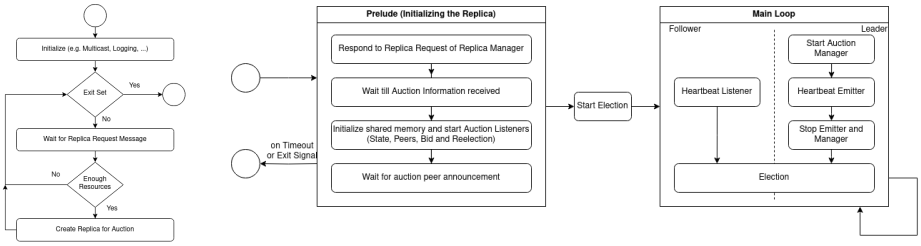
groups are still possible, however, the system ensures over the ID of messages, that messages not intended for the auction can be safely ignored. Furthermore, the auction announcement listener can be parameterized to only listen to a certain auction, to get quicker updates on the state of the auction. As this is used to communicate the whole state of the auction to clients outside the group, an unreliable multicast is used, as a consistent view of the state of the auction is not necessary.

The *Auction Bid Listener* listens to incoming bids over the auction multicast group and updates the state of the auction. This is employed by the replicas and uses the reliable ISIS algorithm to ensure that the state of the auction is consistent between replicas. Paragraph **Communication** gives a brief overview of how ISIS is used to ensure this.

The replicas employ three more listeners over the auction multicast group, namely the *Auction Peer Listener*, the *Auction State Listener*, and the *Auction Reelection Listener*. The peer listener and state listener are used to update the list of peers and the progress (e.g. running, finished, canceled) of the auction. Hence, these listeners also use the reliable ISIS algorithm to ensure that the state of the auction is consistent between replicas. The reelection listener is used to listen to incoming election requests and coordinator messages and either answers the election request or marks the coordinator as the leader, respectively. Due to the underlying leader election algorithm (bully algorithm), these messages are sent over a reliable unicast, as the algorithm requires a reliable communication channel.

Server Figure 3 gives an overview of the server and replica processes used by the system.

Server The server is mainly responsible for managing replicas. It listens for replica request messages (cf. Table 1) over the discovery multicast group sent by



(a) Server (b) Replica

Fig. 3: Server and Replica processes used by the system. The server is mainly responsible for managing replicas, while a replica is responsible for holding auctions and coordinating with each other.

the replication manager process. Depending on if the server has enough resources, it creates a corresponding replica process for the auction and starts it. For no request can, multiple replicas be started, as the server keeps track of requests seen and ignores requests from the same auction.

Replica A replica is responsible for holding auctions and coordinating with each other.

When a replica is initially started, it executes its prelude phase. This prelude phase is used to negotiate with the replication manager if the replica is allowed to join the peers of the auction. This is done by sending a response to the replica request message (cf. Table 1) over a reliable unicast. If the replica is allowed to join the auction, it will start waiting till the replication manager shares the overall auction state with the new replicas, again over a reliable unicast, to ensure that the state of the auction is consistent between replicas. The wait is done to allow the replication manager to find enough replicas for the auction, however is upper bound by a certain time, such that failed managers do not block the servers' resources. After this, it initializes its listeners as described in the previous section, waits for the announcement of its peers, and starts the election between the replicas. This is done to ensure that the leader of the replicas is the replica with the highest ID and is known to all replicas, which might be the newly introduced replica.

After the prelude phase, the replica starts its main loop, which is used to manage the auction and its peers. The tasks of the main loop depend on whether the replica is the leader or not. If the replica is a follower, it listens for incoming heartbeats from the leader and if no heartbeat is received, starts a new election, as the leader is considered crashed. Note, that the background processes are not stopped, which continues to monitor and update the overall state of the auction. If the replica is the leader, it starts the auction manager process in the background and starts emitting heartbeat messages to its peers. If a heartbeat is not received, the leader considers the peer crashed and removes it from the list of peers. If the number of peers falls below a certain threshold, the leader intro-

duces new replicas to the auction by starting the replication manager process in the background. Finally, if an election is started, it stops its manager process, optionally also the replication manager, so that no new replicas are introduced, and participate in the election process.

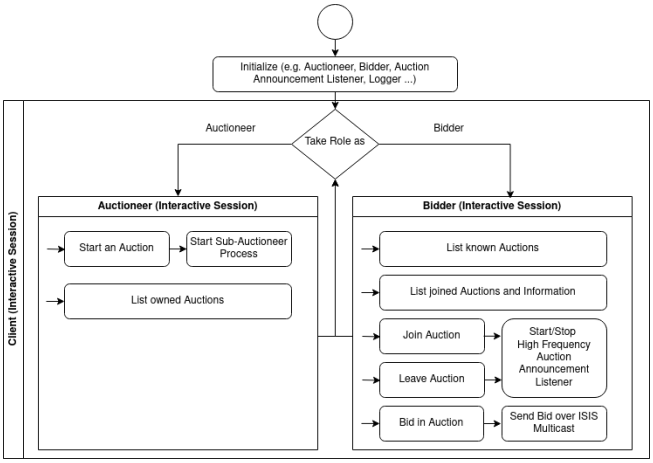


Fig. 4: Overview of how the client operates, including its interactive sessions and corresponding processes it starts.

Client The client, which is executed in the main process, provides the interactive session for a user of the system. As described in Section 2, the user can take part in auctions and auction off items, while in return contributing with its computational resources. This is done by the client, starting the auctioneer and bidder processes responsible for their respective tasks. Furthermore, the client starts an auction announcement listener, listening to incoming auction announcements over the discovery multicast group and stores them, providing them to the bidder and auctioneer processes.

As an auctioneer, the user can create new auctions and monitor the current auctions. When a new auction is created, the client starts the sub-auctioneer process (cf. Section 3.1) responsible for outsourcing the auction to the replicas and monitoring the auction.

As a bidder, the user can list its currently known auctions, join auctions, leave auctions, and place bids in auctions. When a user joins an auction, the client starts another auction announcement listener, however, this time only for the joined auction, to get updates on the state of the auction with higher frequency. When a user leaves an auction, the client stops the corresponding auction announcement listener. Bidding is done by sending a bid message (cf. Table 1) to the auction multicast group. Due to the requirements, that replicas need to have a consistent view of the state of the auction, the bid is sent over the reliable

ISIS multicast, ensuring that the state of the auction is consistent between replicas. However, a client is not considered inside this group and, therefore, does not receive bids from other clients, allowing for a more efficient communication described in Section 3.1.

Table 1: Message types and their routing. Messages are grouped based on their intended use and use a common prefix to indicate their type. The description gives an intuition of the message, and the routing indicates the intended communication channel to use for the message.

Message Postfix	Description	Routing
Reliable Wrapper messages: Prefix MessageReliable		
Request	Wrap messages	Reliable UDP Unicast
Response	Response indicating delivery	Reliable UDP Unicast
Multicast	R-Multicast message	R-Multicast
ISIS Wrapper messages: Prefix MessageIsis		
Message	Initial message for ISIS process	Reliable ISIS Multicast
ProposedSequence	Response with proposed sequence	Reliable UDP Unicast
AgreedSequence	Final sequence announcement	Reliable ISIS Multicast
Replica messages: Prefix MessageFindReplica		
Request	Request for new replicas aiding in auction	Multicast (Discovery Group)
Response	Response of replica to join auction	Reliable UDP Unicast
Heartbeat messages: Prefix MessageHeartbeat		
Request	Heartbeat request	UDP Unicast
Response	Heartbeat response	UDP Unicast
Election messages: Prefix MessageElection		
Request	Election message to call for election	Reliable UDP Unicast
Answer	Election answer, i.e. vote message	Reliable UDP Unicast
Coordinator	Announce own winning	Reliable UDP Unicast
Auction messages: Prefix MessageAuction		
Announcement	Contains whole state of auction	Multicast (Discovery Group)
	and address to join	Multicast (Auction Group)
InformationReplication	Replicate whole auction state	Reliable UDP Unicast
PeersAnnouncement	Inform replica of its peer in an auction	Reliable (ISIS) Multicast (Auction Group)
StateAnnouncement	Announcing current auction progress state	Reliable ISIS Multicast (Auction Group)
Bid	Auction bid send by bidder to auction group	Reliable ISIS Multicast (Auction Group)

Communication The system uses a variety of messages to communicate between the processes, shown in Table 1. These messages are grouped based on their intended use and use a common prefix to indicate their type. The name of messages also corresponds to the process, which uses the message, such that the message can be easily identified.

For reliable uni-directional communication, the system uses a reliable UDP unicast, where messages are wrapped in a reliable wrapper message, containing the payload, a checksum, and a message identifier. On reception, the message is checked for its integrity and if the message identifier has not yet been seen, the message is processed and delivered. On delivery, a response is sent back to the sender, notifying the sender of the delivery.

Due to the system's requirements for a reliable totally ordered multicast, the system uses the ISIS algorithm building on top of R-Multicast over IP multicast and reliable UDP unicast. The ISIS algorithm is used to ensure that the state updates including new bids, peers, and state changes are consistent between replicas. Furthermore, the ISIS algorithm is adjusted, so that it can be

used with open groups, as bidders do not have the same reliability requirements as the replicas. This is done by disabling clients to receive ISIS-messages and hence participate in the ordering, only allowing them to participate if they are a sender. This reduces the group to only containing replicas, as they are known and monitored through the heartbeat by the leader. Hence, the algorithm can be further simplified, as a sender can upper bound the response time to the heartbeat inside the replica group. Also, replicas can clean up the hold-back queue, if a sender crashes, as the final agreed sequence is also upper bounded. Therefore, if the ISIS algorithm with R-Multicast is used only inside the auction multicast group, these simplifications can be made.

Figure 5 illustrates an example of how the system uses its channels to communicate. Note how auction-related updates between replicas are only sent over reliable communication channels so that the state of the auction between replicas can be guaranteed to be consistent. Furthermore, note how messages outside the auction multicast group are sent over unreliable communication channels, as there is no need for a consistent view of the state of the auction. This allows to reduce the load on the system and to increase the efficiency of the communication.

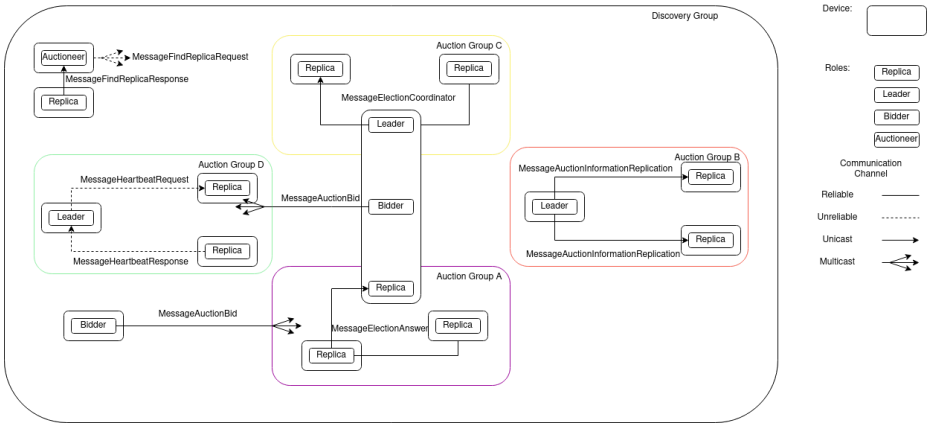


Fig. 5: Multicast group diagram.

Miscellaneous To configure the system, we use a configuration file, which is read at the start and defines the parameters of the system (e.g. port numbers, multicast group addresses, ...). To see all the configuration options, see the configuration file in the repository. Furthermore, the system logs all messages and actions taken by the system on a process level, such that the system can be debugged and monitored for errors. This is done to ensure that the system can be debugged and that the system can be monitored for errors.

3.2 Technical Requirements

In this section, we will briefly describe which technical requirements are met by the system and how they are met.

Architectural Model The system implements a peer-to-peer architecture, where each peer acts both as a client and a server. The *client* part is responsible for the user interaction, allowing the user to take part in auctions and auction off items. The *server* on each device, manages incoming auction replica requests by creating new replicas, if possible, and removing them when the auction is finished. These replicas are responsible for holding auctions and coordinating with each other, in a sense acting as the distributed server for a given auction.

Dynamic Discovery of Hosts A static discovery multicast group is used to share information about the auction and request resources, which is known to all clients and servers. Clients can use this discovery multicast group to find new auctions and to keep track of current auctions. Auctioneers and leaders of replicas are able to use this discovery multicast group to announce auctions and corresponding updates to these and to find new replicas or replace replicas if replica peers of the auction crash. For each auction, a multicast group is dynamically created and used to hold the auction. The so-created multicast group is communicated through the auction announcement and during replica requests so that bidders and replicas respectively can join the auction. This group is used for bidding, replicas to discover and keep track of its peers and the state of the auction, as well as high-frequency updates on the auction for clients. Even if the multicast group is used multiple times, the system ensures that the delivered message over this group is only used by the intended auction recipients by using the message identifier to filter out messages for other auctions. An overview of how the systems' multicast groups are used can be seen in Figure 5. To simplify communication and improve latency in an auction, certain processes use pre-defined ports.

Leader Election Section 3.1 describes, how the leader starts his respective auction manager process, handling aspects of the auction, like answering auction information requests, and announcing the auction and its state. Furthermore, a leader's responsibility is to find new replicas, if the heartbeat detects a crashed replica. Hence, a leader election under replicas is necessary to find the leader under the set of replicas. For this, we choose the bully algorithm, as certain assumptions made by the algorithm are naturally fulfilled by the system. The algorithm assumes that a reliable and bounded message transmission over UDP is possible, which is fulfilled by the reliable UDP unicast. Furthermore, for replicas, the participants of the election are known to each other by the peer announcement of the leader, and the total ordering of replicas is possible by using their addresses. Also, the algorithm assumes that crashed peers can be detected and are not replaced during the election. This is also fulfilled by the system,

as the heartbeat is used to monitor the leader and peers and to detect crashes. The leader, responsible for introducing new replicas, ensures that during the election, new replicas are only introduced after the election by the new leader, by stopping the replication manager process. To ensure that the correct leader is elected after new replicas are introduced, a new election is held. Therefore, by construction and from the algorithm guarantees, we can ensure that the leader is always the replica with the highest ID from the non-crashed replicas and that the leader is always known to the peers, even under crashes.

Ordered Reliable Multicast Replicas in an auction need to have a consistent view of the state of the auction. A state change in an auction includes new bids received from bidders, changes in an auction progress state (e.g. start, finish), but also changes in the peers of the auction and replication of the auction state to new replicas. As these changes need to be consistent between replicas and are transmitted over the auction multicast group, a reliable totally ordered multicast is necessary (in combination with reliable unicast for state replication). Additionally, bidders are considered outside the multicast group, as they do not participate in keeping the auction running, and hence, the algorithm needs to be able to handle open groups.

Therefore, we choose the ISIS algorithm, as it guarantees a totally ordered multicast, even in the presence of open groups, building on top of R-Multicast over IP multicast and reliable UDP unicast. The underlying R-Multicast over IP multicast and reliable UDP unicast are used to ensure that the messages are delivered during the ISIS algorithm and are implemented as described in the lecture notes. However, as the R-Multicast assumes that the group is closed, the ISIS algorithm is adjusted, so that it works with our implementation. Clients using ISIS are only allowed to send bids, hence do not receive ISIS-messages from other clients and can be considered outside the multicast group. Therefore, clients only participate in the ordering of messages for their messages sent. However, the ISIS algorithm still ensures the correct ordering of messages, as the proposed sequence is given by the replicas participating in each ordering of received messages of the auction multicast group. With this, the ISIS algorithm can be used to ensure that the state of the auction is consistent between replicas, even if the group is open and clients can send messages to the group. Furthermore, due to the heartbeat used to monitor the leader and peers, the ISIS algorithm can be simplified, as the messages from each replica must be upper bounded by the heartbeat time. This allows the sender to stop waiting for a response after the heartbeat time, as replicas are considered crashed. Also, replicas can clean up the hold-back queue during their reordering, if a sender crashes, as the final agreed sequence is also upper bounded. These system guarantees allow the ISIS algorithm to be simplified significantly, decreasing the load on the system and reducing the latency of the system. Finally, when new replicas are introduced into the system, the state of the auction is transmitted to the new replicas using a reliable UDP unicast. With this, we can ensure that the state of the auction is consistent between replicas.

Fault Tolerance Due to the architecture of the system, the system fault tolerance can be divided into two parts, namely the client and the server.

Client Assuming the client acts as a bidder, the only critical part is, if the client currently was in the process of bidding. A client's crash is handled as a leave of the auction (albeit not gracefully) and hence the bid is not considered. Due to the use of the adjusted ISIS algorithm, the state of the auction is consistent between replicas, even if a replica crashes.

As an auctioneer, the client's critical part is in the initial 'outsourcing' of the auction to the replicas. If the auctioneer crashes during this process, the replicas will release themselves after a certain time. This is due to the use of a reliable unicast between the replicas and the replication manager. Therefore, if a message is not delivered, the receiver is believed to have crashed and the corresponding replica is removed from the list of candidates. On the other hand, a crashed auctioneer (i.e. replication manager) results in the replicas releasing themselves after a certain time, as the auctioneer is not able to share the auction state.

We can conclude that the system is fault-tolerant to the crash of clients.

Server By use of a heartbeat, the system is able to detect crashed replicas and remove them from the auction. These replicas are then not further considered in the auctions processes (e.g. leader election, ISIS) and their corresponding peers are adjusted to not consider these replicas. Furthermore, if a replica itself is the leader, other replicas will notice the crash and start a new election, guaranteeing that a leader exists between the replicas in an auction. If the replica number falls below a certain threshold, the leader will introduce new replicas to the auction (cf. section 3.1). This is done to ensure that the auction can continue, even if replicas crash.

The server is only responsible for managing the replicas and hence does not have to be considered in the fault tolerance of the system.

Overall, the system can be considered to be fault-tolerant to the crash of clients and replicas and hence as a whole.

4 Assumptions, Shortcomings and Possible Fixes

During the development of the system, several assumptions were made, which we will discuss in this section.

Authentication Service The system assumes that a user has a unique identifier, like a username, to be able to track and uniquely identify bids to a user. This is not implemented in the system, and hence, the system can not guarantee that a user is unique. To remedy this, a fixed and known service can be introduced over which initial authentication can be done. This service can in itself be a distributed system, which is known to all replicas and clients.

Communication over Secure Channel The system assumes, that the communication between replicas and clients is secure. In its current implementation, however, the system does not guarantee this. To remedy this, a public-key encryption scheme could be used, where the public-key is distributed over the fixed and known service, just as with the authentication service. This would allow for secure communication between replicas and clients.

Storage of Auctions The system does not store auctions after they are finished, only a last message is sent to the multicast group, announcing the end of the auction and the winner. This means, that an auctioneer is unable to look into its auctions if it failed during the process, nor is a bidder able to look into the auctions he took part in. This can be remedied by introducing a storage facility, which stores the overall auction state, including the bids and the winner of the auction. In combination with the authentication service, this would allow a user to look into their auctions and the auctions he took part in.

Consensus Under the assumption of consensus, it would seem that a guarantee of Byzantine Fault for an auction bid is possible, as all replicas must hold the correct state by construction. In combination with a consensus in the release of final auction states to the bidders, this would lead to the correct state for the bidders in an auction. However, due to the auction manager process of the leader of the replicas, which governs the process of moving the auction to the next state, this is not guaranteed. For the auction finish state transition, this might be acceptable, as this is determined by the time the auction ends, yet, the leader also governs the start of the auction and its announcement to the discovery multicast group. Hence, if the leader misbehaves in this part, the auction can not start, and the auction is stuck. By design of the system, this is a problem, and state changes should be moved into the replicas themselves, such that the leader can not block the auction from starting or finishing.

Word count: 4806

References

1. Gerwin, C., Mieth, R., Dvorkin, Y.: Compensation mechanisms for double auctions in peer-to-peer local energy markets. Current Sustainable/Renewable Energy Reports **7**(4), 165–175 (Dec 2020). https://doi.org/10.1007/s40518-020-00165-1, https://doi.org/10.1007/s40518-020-00165-1
2. Krishna, V.: Auction theory, san diego; london and sydney (2002)
3. Okwuibe, G.C., Zade, M., Tzscheutschler, P., Hamacher, T., Wagner, U.: A blockchain-based double-sided auction peer-to-peer electricity market framework. In: 2020 IEEE Electric Power and Energy Conference (EPEC). pp. 1–8 (2020). https://doi.org/10.1109/EPEC48502.2020.9320030
4. Pereira, H., Gomes, L., Vale, Z.: Peer-to-peer energy trading optimization in energy communities using multi-agent deep reinforcement learning. Energy Informatics **5**(4), 44 (Dec 2022). https://doi.org/10.1186/s42162-022-00235-2, https://doi.org/10.1186/s42162-022-00235-2

5. Teixeira, D., Gomes, L., Vale, Z.: Single-unit and multi-unit auction framework for peer-to-peer transactions. International Journal of Electrical Power & Energy Systems **133**, 107235 (2021). https://doi.org/https://doi.org/10.1016/j.ijepes.2021.107235, https://www.sciencedirect.com/science/article/pii/S0142061521004749
6. Xu, S., Zhao, Y., Li, Y., Cui, K., Cui, S., Huang, C.: Truthful double-auction mechanisms for peer-to-peer energy trading in a local market. In: 2021 IEEE Sustainable Power and Energy Conference (iSPEC). pp. 2458–2463 (2021). https://doi.org/10.1109/iSPEC53008.2021.9735918