

Decentralized Auction-based Pricing with PeerMart

*D. Hausheer*¹

¹*ETH Zurich, TIK*

Gloriastrasse 35

CH-8092 Zurich

Switzerland

{hausheer, stiller}@tik.ee.ethz.ch

B. Stiller^{2,1}

²*University of Zurich, IFI*

Winterthurerstrasse 190

CH-8057 Zurich

Switzerland

stiller@ifi.unizh.ch

Abstract

Auction-based pricing mechanisms offer the flexibility of setting prices for goods dynamically and efficiently based on current supply and demand. This paper presents PeerMart, a new mechanism which combines the economic efficiency of auction-based pricing with the technical performance and resilience of P2P networks. PeerMart supersedes the need for a central auctioneer by distributing broker functionality over all peers in the network. A structured and redundant P2P overlay network design is applied to achieve scalability and robustness even in the presence of malicious peers.

Keywords

Peer-to-Peer (P2P) Networks, Distributed Auctions, Pricing Mechanisms

1. Introduction

Online auctions like eBay [5] are becoming increasingly popular market places for trading any kind of goods or services over the Internet. Auction-based markets benefit from the flexibility to adjust prices dynamically and enable to achieve efficient supply allocations [7], [22]. However, those markets usually rely on a central component, i.e. the auctioneer which collects price offers of all participants and performs matches. Such centralized systems are vulnerable against attacks and may suffer from overload or failures.

At the same time, an increasing number of systems and applications in the Internet benefit from the scalability and robustness of emerging peer-to-peer (P2P) networks (cf. [10] for an overview). P2P networks implement the idea that peers jointly achieve a task such as key lookup [14] or file storage, without relying on central components. By means of appropriate aggregation and replication techniques P2P-based systems can provide much higher reliability and performance than traditional client/server-based applications [13].

Combining the advantages of those two areas, the proposed approach was termed PeerMart. It implements an economically efficient pricing mechanism, a variant of

the Double Auction (DA), on top of a P2P overlay network. The basic idea of PeerMart is to distribute the broker load of an otherwise centralized auctioneer onto clusters of peers, each being responsible for brokering a certain number of goods. PeerMart is different to existing work, such as [4] and [16], since it applies a structured rather than a random P2P overlay network, which enables deterministic location of brokers and is more efficient and scalable. Moreover, through the use of redundancy PeerMart can provide a high reliability even in the presence of malicious or unreliable peers.

The remainder of this paper is organized as follows. While Section 2 derives the most important requirements for a distributed pricing mechanism, Section 3 outlines the design space, summarizes related work, and discusses major shortcomings. Section 4 presents the key design and characteristics of PeerMart. Furthermore, main building blocks of a prototype implemented are outlined in Section 5, while results obtained from various experiments are discussed in Section 6. Finally, Section 7 concludes the paper.

2. Requirements

To be able to apply a distributed pricing mechanism within real-world applications, key requirements need to be met during its design. Any approach has to consider economic as well as technical and social aspects in an integrated manner. The main economic goal of a pricing mechanism is to achieve an efficient allocation of the goods being traded. At the same time, key technical objectives with respect to performance and scalability should not be violated. In addition to that, it is important to take into account major social aspects such as malicious or selfish behavior of individuals and groups. The following detailed requirements determine the important design objectives:

- **Economic efficiency:** The adopted pricing mechanism should lead to an efficient allocation and use of goods being traded among market participants (traders). Economic efficiency is reached if goods are allocated in a way which maximizes the benefit through their use, i.e. no further gains are possible. The Double Auction (DA) approaches very close this ideal case, and outperforms many other trading mechanisms, such as bilateral search, posted offer, or single-sided auctions [7].
- **Technical performance:** As far as the technical design of the pricing mechanism is concerned, a high performance has to be achieved through an efficient use of technical resources like network capacity, memory space, and processing power. For a distributed implementation of such a mechanism, required network resources determine the main bottleneck. Thus, size and number of exchanged messages have to be reduced to a minimum.
- **Scalability:** With respect to the technical performance of the pricing mechanism, the solution should be capable to operate under any load, i.e. any number of price offers for any goods being traded. This load typically increases as more traders make use of the pricing mechanism. A solution is scalable if the system performance does not decrease as the load increases. A centralized system does not scale well under these circumstances as the load on it increases linearly with the number

of traders. Therefore, a central system can quickly become overloaded, especially if no centralized load balancing concepts are applied. In contrast, P2P systems benefit from the characteristic that the load caused by a participating peer can be compensated by those additional resources provided by that peer. Emerging P2P overlay infrastructures like Chord [20] or Pastry [17] benefit from this advantage and provide, in addition, scalable and efficient routing mechanisms which can be used for object replication and load balancing purposes.

- **Reliability:** It is important that the pricing mechanism is available continuously and performs correctly and securely even in the case of individual failures. Centralized systems are highly vulnerable against total failures or Denial-of-Service attacks which can basically make a system unusable. P2P systems are by design more robust against such failures or attacks. But at the same time they can suffer from the fact that those peers (traders) are autonomous entities, which may not behave as intended by the designer of the mechanism. The behavior of a peer can, *e.g.*, be selfish or malicious. Thus, it might not be willing to cooperate or even more try to actively harm the system, *e.g.*, for its own benefit. The pricing mechanism has to minimize the impact and prevent or discourage such behavior.

Further desirable properties, such as accountability, privacy or anonymity, exist and a complete market system has to comply with them, depending on the dedicated target application. It is important to find the right trade-offs, as different requirements often contradict each other, *e.g.*, it is difficult to guarantee accountability and anonymity at the same time.

To achieve accountability, it is assumed that an accounting or payment system exists which provides the notion of a common currency in exchange for the goods being traded. This may, *e.g.*, be a scalar value, which can be aggregated over time and represents the current credit of a trader. Ideally, such a system should also try to avoid any central components to not reintroduce any bottlenecks. One of the main challenges is clearly to bind the accounting information to a real identity or making re-entries of traders under a new identity costly and thus unattractive. Potential systems that could be used for this purpose are, *e.g.*, Karma [21], PPay [23], or Token-based Accounting [8]. A similar mechanism is needed to keep track of a trader's reputation, considering its behavior in the past, such as cheating, freeriding, or running malicious attacks. There are trust mechanisms like EigenTrust [11] or NICE [12] which are able to aggregate such information in an efficient way. The trust metric is needed to be able to exclude misbehaving traders from the system.

For a P2P-based solution it is additionally important to have a closer look at specific characteristics of a P2P environment. One of the main obstacles in a P2P systems is the problem that peers (traders) may not be willing to cooperate and do not behave correctly according to a particular pre-defined mechanism. Firstly, cooperation is costly. Thus, peers might simply have no incentives to cooperate, *i.e.* behaving correctly as a broker. But even worse, as peers are users and providers of the pricing mechanism at the same time, they might be competitors for a particular good and thus benefit from behaving maliciously. For instance, a peer could lose the opportunity to sell or buy a particular good by forwarding the price offer of another peer. In any sit-

uation considered it has to be taken into account that peers are autonomous and act in a rational and selfish way [18]. The use of additional mechanisms like accounting or reputation can solve this problem only, if a particular behavior is accountable, *e.g.*, the provision of a resource. However, for a functionality like forwarding or caching requests the problem is much more complicated. *E.g.*, it is very hard to detect that a peer did not forward a particular message. This problem becomes even more essential when business sensitive information like price offers for goods need to be processed or stored. Applying accounting mechanisms as an incentive mechanism to provide such functionality may not be feasible due to the enormous technical effort required. Also, it might not solve the problem, since the incentive for forwarding or caching a message would need to be higher than the potential benefit for not doing it. In these cases the use of redundancy can help to improve the robustness of such a distributed mechanism [19].

3. Design Space and Related Work

Based on those main categories of requirements for a distributed pricing mechanism, many alternative approaches exist to set prices and disseminate them to all participants. Starting with non-market-based mechanisms, the simplest approach covers fixed prices, which are determined by a central authority, *e.g.*, the system designer. While such an approach would technically be very efficient, it implies that the central authority has complete information to set optimal prices in advance.

A good overview on the large design space of market-based pricing mechanisms (called market institutions) is given in [7]. Thereby, many pricing mechanisms rely on centralized components, *e.g.*, a central broker. In the absence of such a central authority the problem becomes technically complex. Prices need to be communicated to other market participants in a reliable manner. A simple approach includes traders being connected in a random P2P network, asking all participants for their prices (bilateral search) or regularly sending their price offers for goods to all other participants in the network (posted offer). In the latter case, an offer could directly be answered with a counteroffer by any participant. An approach similar to this is adopted in [4]. Obviously, such a solution does not scale and there are no guarantees, whether all market participants can be reached. A more scalable but rather complex approach is to store offers for later use in form of a routing table at intermediate peers in the P2P network. Those could use this information to route requests to the provider (consumer) currently asking the lowest price (bidding the highest price) rather than broadcasting them. However, it is unclear how long prices should stay valid before being dropped from such a price-based routing table. Also it is difficult to make a prediction about the reliability of this approach in the presence of malicious peers.

In contrast, auction-based approaches are promising in terms of both economic and technical efficiency. While single-sided auctions are simple to implement, they have the drawback to be either provider- or consumer-oriented and thus asymmetric and not optimal. Double auctions enable both providers and consumers to offer prices. An attempt to implement a double auction in a P2P environment is proposed in [16]. Their algorithm is based on agents which are connected in a random P2P network. It

is considered that only one commodity good is being traded. Agents randomly join to build clusters and assign a single agent as the cluster center which keeps a map of all agents in the cluster. Thus, the maximum cluster size has to be limited which implicates that the solution does not scale well. In addition, it is assumed that messages are never lost or delayed, which is not a realistic assumption for a P2P network in practice.

PeerMart, as presented in detail in the following section, has been designed with a strong focus on the technical feasibility *and* efficiency of implementing a double auction mechanism on top of a P2P network. An analysis of the economic efficiency of double auctions is out of the scope of this paper. Justifications of this fact can be found in [15] and [22].

4. PeerMart Design

The core economic mechanism used in PeerMart's design and subsequent implementation is the Double Auction (DA) which has also been adopted by [4] and [16]. Today, variants of the DA are widely used in stock markets. Other than in single-sided auctions such as the Second-Price Auction, in a DA both providers and consumers can quote prices (bids or asks) in a symmetric way. These are matched by the corresponding broker following a certain matching strategy. The core technical mechanism in PeerMart is based on a structured P2P overlay network such as Chord [20] or Pastry [17]. Structured P2P overlay networks enable deterministic object location and are more scalable and efficient than random or hybrid overlay structures.

4.1 Basic Mechanism

The basic pricing mechanism in PeerMart works as follows: Providers and consumers which are interested in trading a particular good, initially send a price request to the responsible broker. As described later on, brokers are realized by clusters of peers. In the following, the term *service* is used to refer to the good being offered by a peer, *e.g.*, a content service. The responsible broker answers requests for a particular service with the current bid price (ask price), which is the current highest buy price (lowest sell price) offered by a peer. Based on this information, providers and consumers send price offers to the broker, using a particular strategy which can arbitrarily be chosen by the peer (a potential bidding strategy that is close to human behavior can be found in [2]). Continuously, a broker runs the following matching strategy:

- Upon every price offer received from a provider (consumer), there is no match if the bid (ask) is lower (higher) than the current ask price (bid price). However, the offer may be stored in a table for later use.
- Otherwise, if there is a match, the offer will be forwarded to the consumer (provider) that made the highest bid (lowest ask).

The resulting price for the service is set to the mean price between the two matching price offers.

4.2 Underlying Infrastructure

To implement this DA mechanism in a decentralized manner, PeerMart uses a structured P2P overlay network as underlying infrastructure. Currently, Pastry [17] is applied, but in principle any other P2P overlay infrastructure could be used. PeerMart uses Pastry for peers joining and leaving the system, and to find other peers (brokers) in the network. In Pastry every peer is given a unique 128-bit node identifier (nodeId), which can be calculated from a peer's IP address or public key using a secure hash function. In PeerMart it is assumed that every peer has a public/private key pair, which is also used to sign and verify messages. The public keys are bound to the nodeIds using certificates which are signed offline by a trusted third-party. An alternative, lightweight method to acquire a public/private key pair without having to rely on a public key infrastructure is described in [1]. That method is based on crypto puzzles and limits the rate at which new peers can join the system.

4.3 System Design

It is assumed that each service being traded over PeerMart has a unique service identifier (serviceId). Thus, a separate mechanism is required which maps services onto serviceIds. For content services this can be achieved, *e.g.*, by calculating the hash value of the content file together with a fixed set of parameters describing further service aspects, such as the allocated upload rate, maximum jitter, etc. The serviceId needs to have at least the same length as the nodeId, to be able to map the services onto the address space of the underlying DHT. The only parameter considered at this stage that can vary is the price. A set of n peers (called *broker set*) which are numerically closest to the serviceId are responsible to act as broker for that service. As it is assumed that multiple services are available, multiple broker sets will exist concurrently, each being responsible for a dedicated service. Furthermore, as it is assumed that serviceIds are uniformly distributed, every peer on average will be responsible for an equal number of services. Initially and for implementation performance reasons of the DHT-based P2P infrastructure, the broker set is congruent with the leaf set in Pastry. It consists of $n/2$ numerically closest larger nodeIds and $n/2$ numerically closest smaller nodeIds for a particular serviceId. Afterwards, the broker set size n may dynamically be increased, depending on the actual service popularity or for robustness reasons. Furthermore, each peer keeps a table for every service for which it acts as broker. This table has a fixed size of m rows and is used to store at most $m/2$ highest bids and $m/2$ lowest offers.

An example for the double auction mechanism in PeerMart is given in Figure 1. Providers ($P1, P2$) and consumers ($C1$) which are interested to trade a particular service (x) first lookup the responsible broker set using Pastry routing. Pastry routes the request to the corresponding root node (the node numerically closest to the serviceId). The root node returns the responsible broker set, which can now be contacted directly to get the current bid or ask price. If the root node is faulty or malicious and fails to return the broker set, the following fallback method is applied (Note that Pastry does not notice this and therefore does not assign a new root node). Recursively, peers on the path to the serviceId are contacted, until a member of the broker

set is found. This peer then temporarily takes over the responsibility of the root node and returns the broker set. Continuously, a peer can now send price offers to the broker set. Apart from the price, every offer contains a sequence number and a valid time and is signed by the peer's private key. For every peer only the newest offer is kept. The valid time cannot be larger than a maximum time t . After an offer becomes invalid it will be removed from the table.

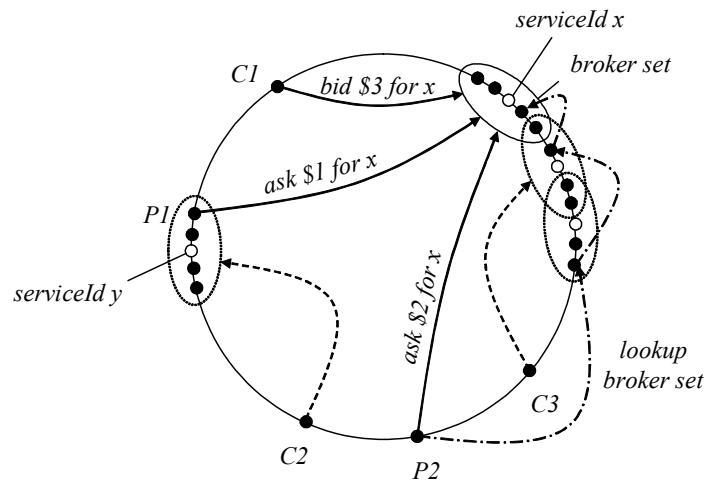


Figure 1: Double Auction in PeerMart

Concurrently, other broker sets exist which are responsible for other services. Note that a peer can act as a provider, a consumer, and as a broker for several services at the same time. Note also, that only the first request (to identify the broker set for a particular service) is routed through the overlay network. All subsequent messages (namely price offers) are sent directly over the underlying IP network.

4.4 Broker Set Maintenance

As mentioned above, initially the broker set for a particular service is congruent with the leaf set in Pastry. When a new service is offered for the first time, the corresponding root node has to notify the other peers in its leaf set about the new service. Again, if the root node fails to do that, the same fallback method as described above can be applied. The next closest node to the serviceId will now act as root node and notify the other peers in the leaf set about the new service.

Furthermore, every peer keeps a list of nodeIds of other peers which are in the same broker set for a particular service. This list is updated regularly based on changes in the leaf set notified by Pastry. Depending on the service popularity (offer rate) or for robustness reasons, the broker set may be increased beyond the size of a leaf set. However, this should normally not be necessary as the services (and hence their different popularities) are uniformly distributed over all peers. Nevertheless, if the popularity for a particular service increases beyond a certain threshold t , and a majority of broker peers vote for an increase of the broker set size, the broker set is

extended by the nodes in the leaf sets of the two outermost brokers. Similarly, the broker set can be decreased again. The majority vote prohibits that a malicious peer can wrongly cause a change of the broker set size.

4.5 Broker Set Redundancy and Synchronization

As a countermeasure against faulty or malicious peers, initial price requests and subsequent price offers are always sent to f randomly selected broker peers in parallel ($1 \leq f \leq n$). f is a design parameter that has to be set very carefully with respect to the ratio of malicious peers, desired reliability, and message overhead, for which there is always a trade-off. Broker peers receiving an offer either reject it or store it in their tables according to the strategy described in Section 4.1 above. Every broker peer forwards pairs of locally matching offers to all other peers in the broker set. Based on the signature of an offer, brokers can easily verify its validity. In addition to the offers matching locally, a broker also forwards the current highest bid and lowest ask, if it has not already been sent earlier. Thus, only potential candidates for a match are synchronized among peers in a broker set. Based on the offers received from other brokers the current bid price (ask price) can be determined and a globally valid matching can be performed by every broker. Asks and bids matching globally are finally forwarded to the corresponding peers by those broker peers which initially received them.

In this redundant approach message loss is implicitly considered. When a message is lost accidentally between two brokers, it appears as if one of the brokers would act maliciously. However, so far timing issues have not been dealt with, and it was assumed that all messages are sent without any delay. In PeerMart a slotted time is used for every individual auction to tackle the problem of message delays. Time slots have a fixed duration which has to be longer than the maximum expected round trip time between any two peers. Every time slot has a sequence number starting at zero when a service is traded for the first time. Price offers from providers (consumers) are collected continuously. At the end of every even time slot, the potential candidates for a match are forwarded to the other brokers and arrive there during an odd time slot. Candidates arriving during even time slots are either delayed or dropped, depending on the sequence number. At the end of every odd time slot, the final matches are performed and notified to the corresponding peers. Since after this synchronization process all broker peers have the same information needed to match offers and bids, no matching conflict can occur. In the rare case that more than one peers quoted exactly the same price within the same time slot, a broker peer gives priority to the one that came in first. After synchronization, the price offer which was prioritized by most brokers is selected.

5. Implementation

PeerMart has been implemented as a prototype on top of FreePastry [6], an open source implementation of Pastry (cf. [9] for more information about the prototype). The common API [3] is used to interact with FreePastry as shown in Figure 2. On every node, an instance of PeerMart is registered as an application. As such it imple-

ments the three main methods *forward*, *deliver*, and *update*, through which a FreePastry node notifies forwarded and received messages, and changes in its leaf-set. Both interfaces are completely generic, which simplifies alternative implementations in future.

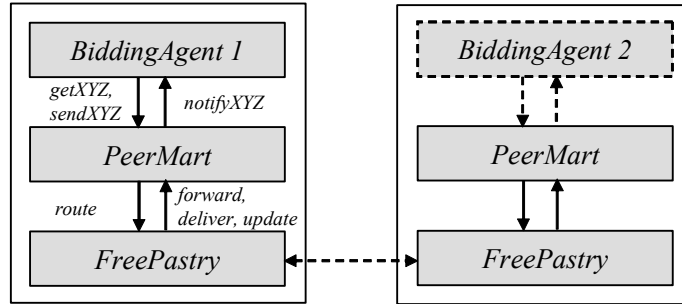


Figure 2: PeerMart Implementation

PeerMart itself offers four main methods, *getBid*, *getAsk*, *sendBid*, and *sendAsk*. The first two methods are used to request the current highest bid price (lowest ask price) for a particular service. If no ask (bid) is available, infinite (zero) is returned. The latter two methods are used to send own price offers. They return true, if the offer could successfully be entered into the table. Otherwise, if the price is lower (higher) than the $m/2$ -highest bid ($m/2$ -lowest ask), they return false. The return values are notified asynchronously through the corresponding callback methods *notifyBid*, *notifyAsk*, *notifySendBid*, and *notifySendAsk*. If a price offer has matched, the result is notified through *notifySuccessfulBid* or *notifySuccessfulAsk*, respectively. The *notify* methods have to be implemented by any application that makes use of PeerMart, e.g., a bidding agent. The implementation of PeerMart’s four main methods, however, is completely transparent for the overlying application.

5.1 Messaging and Threading

All interactions between instances of PeerMart are encapsulated in Pastry messages, which are either routed over the overlay network (broker set lookup) or sent directly to the corresponding destination node (all other tasks). For each task there is a dedicated message. Based on their type, the messages are dispatched remotely and the corresponding methods are executed. Whenever a reply message is expected, e.g., containing a return value or an acknowledge, a new thread is created. The corresponding thread is started by the *WaitingThreadList* class, when the expected message has arrived or after a certain time-out has passed. In that case, the corresponding message may be sent again.

The core task of PeerMart, the maintenance of auction tables including timely exchange of matching candidates and forwarding of successful offers, is performed by a dedicated *AuctionThread*. The *AuctionThread* contains a sorted set of *AuctionTasks* which are executed at a specific time. Additional *TimerTasks* are created, to remove offers from the auction table after becoming invalid.

6. Evaluation and Experimental Results

PeerMart's applicability in today's Internet applications is envisioned as soon as analytical and performance results show a gain and clear trade-off between efficiencies, distribution, and reliability. Therefore, the proposed mechanism is analyzed with respect to overhead, scalability, and reliability in terms of resistance against attacks or unreliable peers.

6.1 Efficiency and Scalability

Efficiency of PeerMart is measured in the amount of overhead in terms of states (storage space) that need to be kept and messages being generated by any peer. There is a basic overhead for maintaining the Pastry overlay infrastructure. For a detailed analysis of the scalability and efficiency of Pastry itself refer to [17]. Beyond that and covering PeerMart, there is an overhead for the exchange of price offers as well as their storage and matching by the broker peers. The number of states which need to be kept by each peer are limited in two ways. Firstly, since the table size is limited, the maximum number of offers and bids per service each broker has to maintain is m . Secondly, it is assumed that the number of services s a peer is concurrently involved in as a trader is limited, as a peer's resources to consume or provide services are physically bounded. Thus, there is an average constant number of $s \cdot n \cdot m$ states each peer has to maintain, where n is the average broker set size depending on the popularity of services. It is assumed, that tables for services in which no peer is involved in anymore (no valid offers exist) can be removed safely.

The number of price messages per service involvement a broker peer needs to deal with are influenced by several factors. Storing/rejecting a price offer and notifying a match generates $2 \cdot f$ messages as only a fraction of broker peers is contacted, which will forward the message. When a price offer leads to a match, an additional number of $n \cdot f$ messages are generated.

To verify and complement those analytical results, the message overhead of PeerMart has been measured in a set of real world experiments with the implemented prototype. The experiments were run on a Pentium 4 CPU 2.4 GHz with 512 MB RAM using Java VM 1.4.2. For all experiments performed there were four times as much peers as services, and each peer was randomly assigned as consumer or provider to two different services. To make the experiments as realistic as possible, the following bidding strategy was applied: While the price bid quoted by a consumer was calculated using $\min(\text{ask price} + a \cdot (\text{bid limit} - \text{ask price}), \text{bid limit})$, the price ask quoted by a provider was set to $\max(\text{bid price} - a \cdot (\text{bid price} - \text{ask limit}), \text{ask limit})$. Thereby, a is a learning parameter which has been set to 0.1 for the experiments. The offer limits (reservation prices) of the peers were normally distributed, such that 80% of the offers were leading to a match. This bidding strategy was motivated by the ZIP strategy presented in [2], which seems to be close to human behavior. The experiments were run until all bidding tasks were either successfully completed once or still unsuccessful after three retries. All messages exchanged between peers were randomly delayed by 100 to 200 ms to simulate the latency of a real network.

Figure 3(a) shows the message overhead per broker peer for a varying number of peers N in the network. It can be seen that the total number of messages per broker increases linearly in a small network, but stays almost constant when the network becomes larger. The messages are broken down into several classes. Auction messages include all messages for the exchange of offers between brokers, while request messages include bid/ask requests, offers sent by peers, and notifications. Finally, other messages include lookup and broker set maintenance messages. The size of all messages is around 1 kB. These results indicate that the system scales very well with the number of peers in the network and has only a small overhead. The use of digital signatures (SHA1 with DSA) has only a minor impact on the system performance. The message size overhead is about 15%, but the total size is still much smaller than maximum size of a UDP packet (64 kB). On the test machine signing and verifying messages took 10 to 20 ms, which is only about 10% of the average RTT in today's Internet.

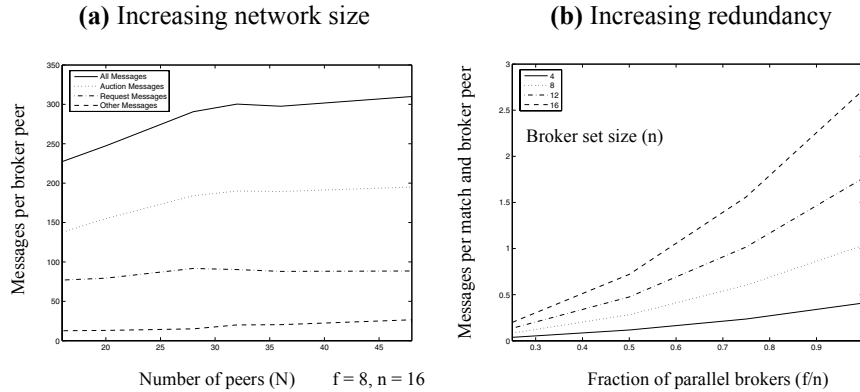


Figure 3: Message Overhead

6.2 Reliability

With respect to reliability PeerMart relies on two concepts, redundancy and reputation. While redundancy helps to resist against unreliable peers and potential attacks by malicious peers, reputation is used as an incentive for peers to stay on-line and to prevent bad behavior. The redundancy of PeerMart can be adjusted through the two parameters f and n , in order to achieve a desired availability and reliability of the system, depending on an expected number of malicious or unreliable peers. The worst-case costs of redundancy (i.e. when all offers lead to a match) are shown in Figure 3(b). As it can be seen, the costs increase quadratically to the fraction of parallel brokers f used within a broker set.

Figure 4 shows the reliability of PeerMart (measured in the amount of correctly matched offer pairs), depending on the number of parallel brokers f and the amount of malicious peers in the broker set for a broker set size of 64. Malicious peers were modeled as brokers which did not correctly forward offers to other brokers without a consumer or provider noticing. This is much harder to deal with than peers which are

simply off-line, since these can usually be detected after no reply has been received within a certain time. As it can be seen in Figure 4, PeerMart can correctly match bids and offers for up to about 50% malicious peers using, *e.g.*, 8 brokers in parallel (out of 64 brokers in total). Under the assumption that the fraction of malicious peers is below 25%, even 4 parallel brokers provide a fairly good reliability. For comparison, results from (a) an analytical simulation and (b) real world experiments are given, which show a similar behavior. Thus, a high reliability can be achieved, by increasing the number of parallel brokers.

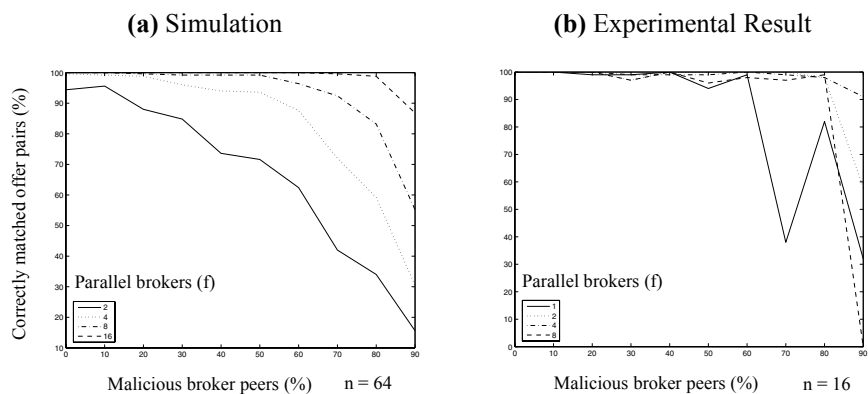


Figure 4: Reliability of PeerMart

The following list outlines additional types of misbehavior and how they are prevented in PeerMart by relying on an external reputation system:

- A peer may continuously send offers and ignore any received match. This can be tackled by decreasing the reputation value of a peer whenever it ignores an offer and increasing it upon every accept. If the reputation value falls below a certain threshold, a peer's offers could simply be dropped, so the peer will be excluded from the system.
- A peer may not stay to its promises, although having previously accepted an offer. This is out of control of PeerMart. Actually, those two peers involved may negotiate the price bilaterally, if both peers agree. PeerMart has no means to prevent this. Otherwise, the same reputation mechanism might be used as above.
- A broker peer may not store or forward an offer. While this is generally not a problem, if at least a few peers behave correctly as shown above, it is still necessary to punish and thus prevent such behavior. A broker peer which does not store an offer cannot immediately be detected. However, if an offer is not forwarded by a broker peer, the receiver will detect it, since fewer than f messages are obtained. Again, the reputation value of the malicious peer can be decreased in this case.

7. Conclusions and Future Work

This paper presented PeerMart, a completely decentralized pricing mechanism based on P2P networks, which can be used by peers to trade services in a double auction. Implemented on top of a redundant P2P overlay network, PeerMart provides a highly reliable, attack-resistant market system at a low overhead of messages and necessary storage space and scales well even for a large number of peers trading services.

Using a redundant number of 8 brokers out of 64 in total, PeerMart could correctly match offer pairs with up to an amount of 50% malicious peers. Based on real world experiments with a prototype, it was shown that the overhead for running PeerMart is small and almost constant independent of the number of peers.

In future work the basic design of PeerMart will further be extended and analyzed and its efficiency will be increased. Especially the synchronization process between brokers which is currently based on broadcast will be optimized. Furthermore, while only the price for a service can be varied at present, the case on how several dynamic parameters can be supported will be investigated.

ACKNOWLEDGEMENTS

This work has been performed partially in the framework of the EU IST project MMAPPS “Market Management of Peer-to-Peer Services” (IST-2001-34201), where the ETH Zürich has been funded by the Swiss Bundesministerium für Bildung und Wissenschaft BBW, Bern, under Grant No. 00.0275. Additionally, the authors would like to acknowledge discussions with all of their colleagues and project partners, in particular Jan Gerke and Vasilios Darlagiannis for lots of helpful comments.

References

- [1] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach: *Security for structured peer-to-peer overlay networks*; In Proceedings of the Fifth Symposium on Operating Systems Design and Implementation (OSDI'02), Boston, MA, USA, December 2002.
- [2] D. Cliff: *Minimal-Intelligence Agents for Bargaining Behaviors in Market-Based Environments*; Technical Report HPL-97-91, HP Laboratories, Bristol, England, 1997.
- [3] F. Dabek, P. Druschel, B. Zhao, J. Kubiatowicz, I. Stoica: *Towards a Common API for Structured Peer-to-Peer Overlays*; 2nd IPTP'03, Berkeley, CA, February 2003.
- [4] Z. Despotovic, J. Usunier, K. Aberer: *Towards Peer-To-Peer Double Auctioning*; In Proceedings of the 37th Hawaii International Conference on System Sciences, Waikoloa, HI, USA, January 2004.
- [5] eBay Inc., <http://www.ebay.com/>, August 2004.
- [6] FreePastry, <http://www.cs.rice.edu/CS/Systems/Pastry/FreePastry/>, July 2003.
- [7] D. Friedman: *The Double Auction Market Institution: A Survey*, In *The Double Auction Market: Institutions, Theories, and Evidence*, D. Friedman and J. Rust, Eds., pp. 3-25, Addison-Wesley, 1993.

- [8] D. Hausheer, N. Liebau, A. Mauthe, R. Steinmetz, B. Stiller: *Token-based Accounting and Distributed Pricing to Introduce Market Mechanisms in a Peer-to-Peer File Sharing Scenario*; In Proceedings 3rd IEEE International Conference on Peer-to-Peer Computing, Linköping, Sweden, September 2003.
- [9] D. Hausheer: *PeerMart: A Decentralized Auction-based P2P Market*; ETH Zurich, Institute TIK, <http://www.peermart.net/>, September 2004.
- [10] S. Joseph: *Decentralized Meta-Data Strategies: P2P and Agent Perspectives*; Tutorial, 3th International Joint Conference on Autonomous Agents and Multi Agent Systems, New York City, USA, July 2004.
- [11] S. Kamvar, M. Schlosser, H. Garcia-Molina: *The EigenTrust Algorithm for Reputation Management in P2P Networks*; In Proceedings of the Twelfth International World Wide Web Conference (WWW), Budapest, Hungary, May 2003.
- [12] S. Lee, R. Sherwood, B. Bhattacharjee: *Cooperative Peer Groups in NICE*; IEEE Infocom, April 2003.
- [13] A. Loo: *The future of peer-to-peer computing*; Communications of the ACM, Volume 46, Issue 9, pp. 56 - 61, September 2003.
- [14] J. Mischke: *Scalability of Lookup and Search Overlays in Peer-to-Peer Networks*; ETH Zürich, Dissertation, December 2004.
- [15] R. Myerson, M. Satterthwaite: *Efficient Mechanisms for Bilateral Trading*; Journal of Economic Theory, Vol. 29, pp. 265-281, 1983.
- [16] E. Ogston, S. Vassiliadis: *A Peer-to-Peer Agent Auction*; In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Bologna, Italy, July 2002.
- [17] A. Rowstron and P. Druschel: *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. In Proceedings of IFIP/ACM Middleware 2001, Heidelberg, Germany, November 2001.
- [18] J. Shneidman, D. Parkes: *Rationality and Self-Interest in Peer-to-Peer Networks*; 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, USA, February 2003.
- [19] J. Shneidman, D. Parkes: *Using Redundancy to Improve Robustness of Distributed Mechanism Implementations*; In Proceedings of 4th ACM Conference on Electronic Commerce (EC'03), San Diego, CA, USA, May 2003.
- [20] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*; ACM SIGCOMM 2001, pp. 149-160, San Diego, CA, USA, August 2001.
- [21] V. Vishnumurthy, S. Chandrakumar, E. G. Sirer: *KARMA : A Secure Economic Framework for Peer-to-Peer Resource*; Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June 2003.
- [22] B. Wilson: *Incentive Efficiency of Double Auctions*; Econometrica, Vol. 53, pp. 1101-1115, 1985.
- [23] B. Yang, H. Garcia-Molina: *PPay: Micropayments for Peer-to-Peer Systems*; ACM Conference on Computer and Communications Security (CCS '03), Washington, DC, USA, October 2003.