29. Peer-to-Peer Market Management

Jan Gerke, David Hausheer (ETH Zurich)

This chapter focuses on the *economic* aspects of Peer-to-Peer networks. In particular, it describes the most important requirements and presents a basic architecture for a *market-managed* Peer-to-Peer system supporting commercial applications going beyond pure file sharing. In general Peer-to-Peer systems can be used to share *any* kind of resource, including but not limited to, computing power, storage space, network capacity, files, and any combination thereof, *e.g.*, online games or multimedia streams. In the following, the term *service* will be used to refer to the individual provision of goods or resources by peers. The main goal is a completely decentralized and generic marketplace for efficient trading of such services among peers.

It has been observed in Peer-to-Peer file sharing applications, that in the absence of appropriate economic and social mechanisms, Peer-to-Peer systems can suffer from the behavior of selfish peers, which do not cooperate [11]. This is also known as the *free-rider problem.* As peers are autonomous entities, they need to be given the right *incentives* for offering services to other peers and to behave correctly. Without any central regulator, this is clearly a non-trivial problem to be solved. Any approach has to consider both economic and technical aspects in an integrated manner. This chapter tries to investigate how a technically viable *and* incentive-compatible Peer-to-Peer system can be built. Two case studies are presented which show how particular parts of this given problem can be solved.

The remainder of this chapter is organized as follows. Section 29.1 derives key requirements for a market-managed Peer-to-Peer system based on the main problems identified. Section 29.2 describes the architecture of such a system and outlines its core elements and mechanisms. Section 29.3 then presents two specific approaches towards a Peer-to-Peer market. First, a Peer-to-Peer middleware is presented which enables the architecture and focusses on service negotiation and management aspects. Second, Peer-to-Peer double auctions are described, which make it possible to extend the system with a completely decentralized pricing mechanism. Finally, Section 29.4 concludes this chapter.

29.1 Requirements

A set of key requirements need to be met in order to support real-world applications by the design of a market-oriented Peer-to-Peer architecture. The main goal is to support market mechanisms, while maintaining the technical benefits of Peer-to-Peer networks. The core functional and non-functional requirements for such an architecture are derived from the various problems currently observed in many Peer-to-Peer systems.

29.1.1 Main Problems

Peer-to-Peer systems are based on the idea that peers offer services to other peers. Ideally and in the absence of a monetary payment system, each peer should contribute as much as it uses from other peers. However, as peers are autonomous entities acting in a rational way, it is unlikely that such cooperation is going to happen without appropriate incentives for peers to share their resources. In fact, it was shown in [11] that 70% of Gnutella users share no files at all. Thus, many users in Gnutella compete for the resources offered by only a few peers, which leads to a major degradation of the overall system performance.

Some Peer-to-Peer systems use specific accounting or reputation mechanisms to deal with this problem, such as BitTorrent's tit-for-tat mechanism [128], eMule's credit system [589], or KaZaA's peer points [558]. However, most of these mechanisms are purely file sharing-oriented and can thus hardly be used for other types of services. Moreover, due to weak security measures these mechanisms can usually not be applied to commercial purposes.

Another major problem faced in Peer-to-Peer networks, is the fact that individual peers are usually unreliable, i.e. they may be faulty or even act maliciously. Peers may often join and leave the system, lose messages or stored data, or deliberately misuse or harm the system. Replication can help to increase data availability and reliability, however, without appropriate synchronisation techniques replicated data may quickly become inconsistent. In a commercial environment this problem becomes even more essential. A peer may increase its own benefit by acting maliciously against potential competitors, *e.g.*, by not forwarding other peers' service offers. It is hardly feasible to use accounting mechanisms or payments as an incentive to fulfill such basic tasks, as it may be difficult to check whether a particular task has been performed correctly or not. Also, the necessary accounting effort may quickly exceed the effort for the actual task.

As a consequence of decentralisation and the potentially large size of a Peer-to-Peer network, the efficient and scalable design of appropriate search mechanisms and other distributed tasks is another difficult problem. Existing Peer-to-Peer overlay infrastructures such as Pastry or Chord (cf. Chapter 8) allow for efficient request routing, but have limited support against malicious peers or insecure networks. In fact, many Peer-to-Peer mechanisms currently do not consider malicious behavior at all.

29.1.2 Functional Requirements

The following three main functional goals form the basis for a completely decentralized and generic Peer-to-Peer marketplace:

Service Support

The targeted Peer-to-Peer system needs to support completely different services, including purely resource-based services such as processing power, storage, or transportation, as well as higher level services such as content or software applications going beyond pure file sharing. Also, combinations of existing services offered by different peers need to be supported, to create new, value-added services. The service usage model described in Section 29.2.2 illustrates how such combinations of distributed services may look like.

Market-Based Management

The most important goal is the creation of a marketplace for trading different services, managed by true market mechanisms which provide appropriate incentives. On the one hand, the traditional way for this is by introducing a currency that can be used in exchange for the services being provided. On the other hand, barter trade is a suitable alternative which has to be supported, too. Barter is a simple form of trade where services are directly exchanged against other services, *e.g.*, a peer may only download a file if it also provides one. The market model described in Section 29.2.1 further details market-related aspects of the Peer-to-Peer architecture.

Decentralization

Today, many Internet-based marketplaces such as eBay [183] are based on centralized infrastructures. However, a true Peer-to-Peer-based system must use only Peer-to-Peer mechanisms which must be able to function without any central components. Only this type of approach offers the full advantage of the Peer-to-Peer concept and ensures that no central point of failure exists.

29.1.3 Non-functional Requirements

Apart from the key functional goals, a market-based Peer-to-Peer system is subject to the following non-functional requirements:

Efficiency

The adopted core mechanisms should lead to an economically efficient allocation and use of the services being traded among the market participants. Economic efficiency is reached when the services are allocated in a way which maximizes the overall social benefit of all participants. Additionally, as far as the technical design of the mechanisms is concerned, an efficient use of technical resources like network capacity, memory space, and processing power has to be achieved. In a distributed system, network resources (i.e. communication bandwidth) determine clearly the main bottleneck. Thus, size and number of exchanged messages have to be minimized.

Scalability

With respect to the technical performance, a solution should be capable to operate under any load, i.e. any number of market participants or services being offered. A system is scalable if the performance does not decrease as the load increases. A centralized system does not scale well under these circumstances, because the load on it increases as more participants make use of it. Therefore, a central system can quickly become overloaded, especially if no centralized load-balancing concepts are applied. In contrast, Peer-to-Peer 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 Peer-to-Peer overlay infrastructures (cf. Chapter 21) 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 a system designed for real-world applications 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. Peer-to-Peer 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 are autonomous entities, which may not behave as intended by the designer of the mechanism as mentioned earlier. A solution has to minimize the impact and prevent or discourage such behavior.

Accountability

Making the services being traded among the peers *accountable*, is another inevitable requirement for a market-managed Peer-to-Peer system. An accounting or payment mechanism is required which provides the notion of a common currency that can represent the value of the individual services. This may be a scalar value, which can be aggregated over time and thus represents the current credit of a peer. Peer-to-Peer accounting systems are discussed in detail in Chapter 32. One of the main challenges of an accounting system is clearly to bind the accounting information to a real identity, thus making re-entries of peers under a new identity costly and therefore unattractive. Karma [608], PPay [636] or PeerMint [283] are potential systems that may be used for this purpose. 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 [334] which are able to aggregate such information in an efficient way. The trust metric is needed to be able to exclude misbehaving peers from the system.

Further desirable properties, such as privacy or anonymity, exist, which may contradict accountability, as it is difficult to guarantee accountability and anonymity at the same time. It depends on the dedicated target applications, if a system has to comply with them.

29.2 Architecture

The previous section introduced the concept of service markets based on Peer-to-Peer networks. It also stated the main requirements that a system has to fulfill to enable such a market. This section describes the architecture of such a system (cf. [240]). The architecture primarily consists of three models. The market model describes the roles of service providers and service consumers in the market. The service usage model describes the different ways of using services. The peer model describes the architecture of a single peer. It is extended through the description of key mechanisms which have to be implemented on every peer, in order to enable the Peer-to-Peer-based service market.

29.2.1 Market Model

The classical market is a place where sellers and buyers meet to exchange goods against payment, *e.g.*, money. While in old times this market corresponded to a closed physical location, nowadays the term is used in a much broader sense, *e.g.*, to describe a national or even the global market.

The goods traded in the market described here are services (cf. Chapter 14). Thus, the sellers and buyers are *service providers* and *service consumers*. However, participants in the market are not restricted to either provide or consume a service. Rather, they can take on any of these *roles* at any point of time. This means, that they can provide a service to a second participant and later use a service from a third participant or vice versa. They can even do both at the same time, as shown in Figure 29.1.

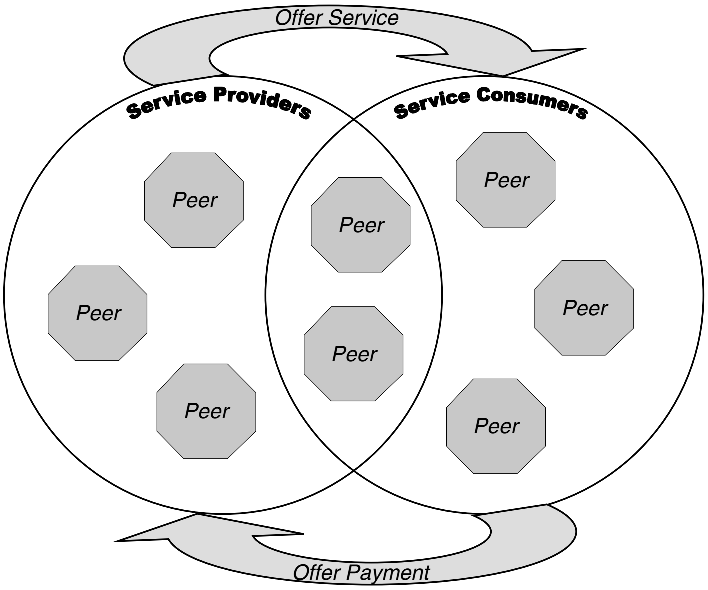

Fig. 29.1: The Market Model

The participants of the service market are peers of an underlying Peer-to-Peer network, so the market covers the complete Peer-to-Peer network. When properly implemented, such a Peer-to-Peer-based market offers low barriers to entry to potential service providers as they do not have to spend a lot of money on marketing first. Rather, Peer-to-Peer search mechanisms allow consumers to easily find any service offering, not just the most visible ones.

As any market, this market shall lead to an equilibrium of supply and demand. Especially, the competition between service providers shall lead to the deployment of new services as well as low prices for existing services. In order to achieve this, the market must give incentives for not overcharging a service and for providing a proper Quality of Service (QoS). On the one hand, the long-term development of the market provides some incentives by itself. If the price of a service is considered to be too high noone will use it and the provider will be forced to lower the price of the service. Similarly, consumers are not likely to use the services of a provider who previously provided them a low QoS. However, the basic market forces are not enough if prices are to be created dynamically in accordance to the current market situation. Neither do they prevent a provider from providing low QoS if the user base is large enough. Thus, additional incentives have to be provided to stimulate a proper behavior in the market. *E.g.*, auction mechanisms (cf. Section 29.3.2) allow to create prices for services dynamically on short notice. Reputation mechanisms allow users to judge the quality of service providers and the services they offer, while Service Level Agreements (SLAs) allow to specify the QoS and terms of service usage in a legally enforceable way.

29.2.2 Service Usage Model

The term *service* is defined as functionality which is offered by one peer to other peers, and which can be accessed through input and output interfaces. Services are described through service descriptions, which describe each service's functionality and interfaces, but also its non-functional characteristics such as terms of usage, *e.g.*, prices and methods of payment. *Services instances* can be used by applications, which are programs running on peers which are not provided as services themselves and offer user interfaces. In addition, service instances can be used by other service instances. As mentioned earlier, examples of services include the provision of basic resources like computing power and storage space, as well as higher-level functionality, *e.g.*, the provision of weather forecasts or booking services.

The service usage model shown in Figure 29.2 shows the various uses of services. The most straightforward case is a peer providing a service instance to another peer, where it is used by an application. Such a case is depicted in Figure 29.2 between Peer 2 and Peer 4. Naturally, applications can use several services at the same time, as is done by the application running on Peer 2. Similarily, the same service can be provided to several users at the same time (Peer 4 providing instances of the same service to Peer 2 and Peer 3). Finally, services can also be used by other services. In Figure 29.2 a service running on Peer 3 which is being provided to Peer 1 is using a local service as well as a service provided by Peer 4. The details of this *service composition* are hidden from the application running on Peer 1 which only sees the single service it uses.

29.2.3 Peer Model

Having laid the foundation for a Peer-to-Peer system through its market and service usage models, it is now essential to derive the internal structure of a peer. To ensure a sufficient degree of modularity of the architecture, a layered structure is used for the peer model. The lowest layer are the resources that are locally available at a certain peer node. On top of this layer services are executed which can draw on local resources, such as storage space, computing power or content. They can also access remote resources through other ser-

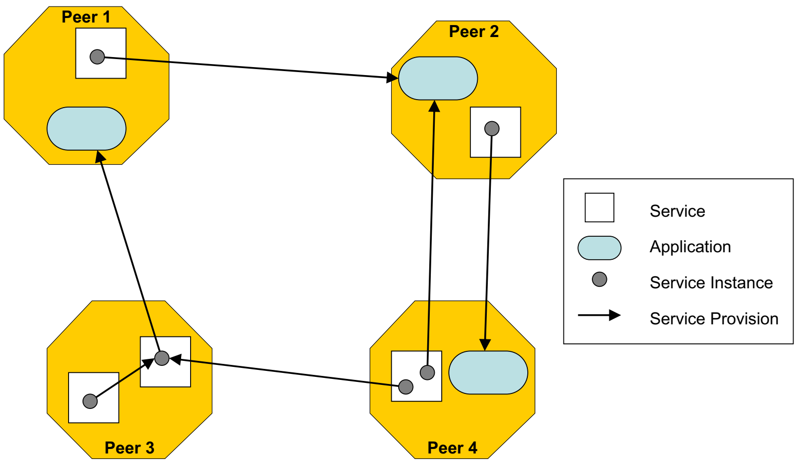

Fig. 29.2: The Service Usage Model

vices. Finally, in parallel to the services layer, core functionality is required to uphold the Peer-to-Peer network and facilitate the smooth operation and interaction of services. Figure 29.3 illustrates the abstract architecture of each peer participating in the market.

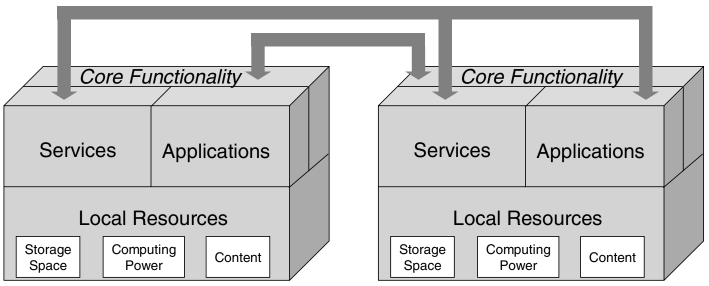

Fig. 29.3: The Peer Model

The core functionality layer on each peer node provides the functionality needed to uphold the peer network services. It is in charge of some basic local functionality like local resource management as well as distributed functionality and protocols like service discovery or reputation. In particular, it includes all functionality needed to enable pricing, metering, and charging and, hence, to support market management of the system. The core functionality layer on each peer accesses and cooperates with the corresponding layers on remote peers but does not access remote or local services.

29.2.4 Key Elements and Mechanisms

The models presented above determine the general structure of a service-oriented market based on a Peer-to-Peer network. Still, the core functionality within each peer, which is responsible for enabling the complete architecture, must be defined in detail and its mechanisms must be identified. The core functionality must handle the management and interaction of services. Therefore, typical business processes for the management and interaction of businesses in a common market provide a reasonable structure for the definition of the core functionality's mechanisms. Table 29.1 describes this structure. Business processes on the left hand side are translated into necessary core functionality mechanisms on the right hand side. While these mechanisms alone are not able to carry out the corresponding business processes in a completely autonomous manner, they are important tools supporting users and user agents. Section 29.3.1 describes how the identified mechanisms work together within a Peer-to-Peer middleware, in order to enable the architecture.

Business Processes	Required Core Functionality Mechanisms
Strategy development Product management Human resource management	Offline task of the service provider
Budgeting and controlling	Resource management and QoS control
Marketing and selling	Service description, discovery, pricing and negotiation
Contracting	Service level agreements
Order fulfillment	Service execution, accounting, charging
Business development	Service composition uses existing core functionality
External security mechanisms	Security mechanisms included

Table 29.1: Mapping business processes to core functionality

While strategy development, product management, and human resource management are without any doubt highly important processes for any business, they mainly occur in the offline world, in a Peer-to-Peer services environment as well as in a client-server one. It is the service provider's task to think of a successful strategy, draft accurate business plans, employ, train, and retain the right people and develop services accordingly. Hence, the architecture does not need to take these processes into consideration.

Budgeting and controlling, the optimum allocation of resources and the controlling thereof can be translated into local resource management and QoS control.

Marketing, guiding a customer towards a product, is mapped onto service description and discovery, certainly one of the most important mechanisms within the core functionality. Sales, selling the product, then corresponds to service negotiation.

In the normal business world the creation of contracts would be the task of a legal department. In the core functionality the contracts are represented as electronic Service Level Agreements (SLAs), which can be negotiated by the core functionality itself, by applications acting as user agents, or by human users themselves.

After a contract has been finalised, the provider needs to fulfill it, including the capturing of all accounting and charging information for the invoicing and payment steps.

Business development, the cutting of major deals and cooperations to augment the business, corresponds to service composition, the cooperation of several services to deliver a joint result as described in the service usage model. However, this task is not part of the core functionality, but merely uses important parts of it, *e.g.*, service discovery, service negotiation and SLAs.

Finally, while offline businesses can leave security tasks to a high degree to governmental authorities (police, courts), security aspects are crucial for a Peer-to-Peer services architecture, *e.g.*, identification, authentication, authorization, encryption/decryption mechanisms.

29.3 Case Studies

The last two sections introduced the concept of a Peer-to-Peer based service market and a system architecture to enable such a market. Still, this has merely given an overview over the whole topic. Therefore, the purpose of this section is to give a more detailed view on two sub topics to serve as examples of the involved complexity and problems. First, the design of a Peer-to-Peer middleware is introduced. Its purpose is to implement the key mechanisms described in the previous sections, thus enabling the architecture, which in turn enables the service market. This middleware has been developed and implemented within the EU-funded MMAPPS project [591] where it has been successfully used as a basis for various Peer-to-Peer-applications. Second, one key mechanism, namely pricing, is presented in even more detail. Its specific implementation uses the overlay network Pastry to enable auctions for services in a completely decentralized manner.

29.3.1 Peer-to-Peer Middleware

Different implementations of core functionality described in Section 29.2.3 can exist on different platforms and access the local resources through platform dependent interfaces. However, all implementations follow a common standard of protocols to communicate with each other and offer a uniform interface to applications in the form of APIs (Application Programmers Interfaces). Thus, platform dependent details are hidden from the application and its programmers. Together, the core functionalities of all peers act as a middleware (cf. [69]), which makes it possible to treat the collection of all peers as a single service-oriented Peer-to-Peer network.

Only the API of the local core functionality implementation is visible to a service or application developer and thus the distributed nature of the middleware is hidden from him. Therefore, in this section the single implementation of the core functionality is also referred to as 'middleware'. A modular approach has been chosen for designing this middleware. This approach has the following benefits:

– Details are hidden from modules which are not concerned by them.
– Implementors don't have to consider the complexity of the complete middleware but only the less complex single modules.
– It facilitates maintenance, especially plugging in new modules.

The Peer-to-Peer middleware consists of six modules as shown in Figure 29.4. These modules belong to three different groups. The central group are the Service Support modules, namely the Service Negotiation and Service Management module. These modules are responsible for managing service related information, i.e., service descriptions and SLAs, and for controlling service execution. In order to enable the effects of market forces, additional functionality is provided by Market Management modules, namely Pricing to determine the price of services in the market, and Accounting and Charging (A&C) to collect information about service usage and calculate charges based on this information. Finally, the Enabling modules provide basic functionalities needed to enable the Peer-to-Peer network and offer support to the other modules. The Search module is used to search and find services in the Peer-to-Peer network, and the Security module provides encryption and identity management to all other modules, especially access control to the Service Negotiation module.

In Figure 29.4 each arrow represents a data flow. Dotted arrows denote remote interfaces, i.e., a module of one peer communicating with a module of the same type on another peer. When a new service is deployed, it sends its

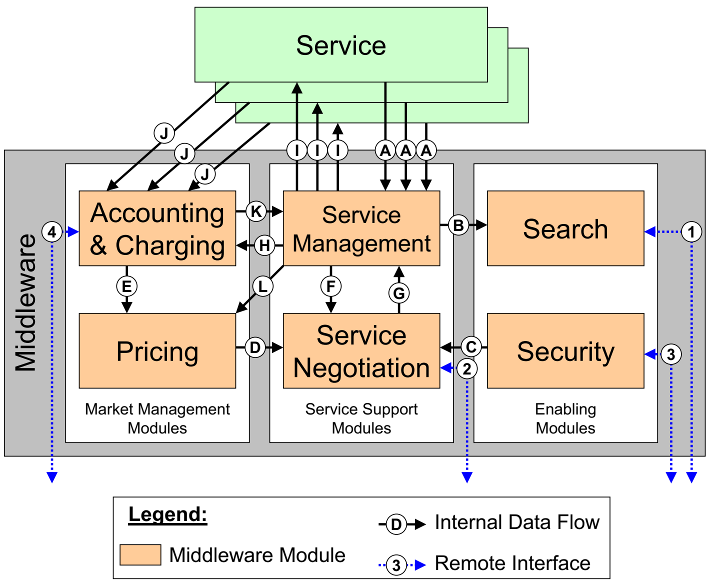

Fig. 29.4: The Middleware Design

description to the local Service Management module via *interface A*. From there the service description is forwarded to the Search module via *interface B*. Thus, whenever a remote peer searches a service via the remote *interface 1*, the Search module can compare the requirements stated in the received service description to the service descriptions it has stored. If this comparison leads to a match, the remote peer can decide to negotiate the terms of service usage via the remote *interface 2*. In this case, the Service Negotiation module will first let the Security module check the remote peer's access rights. The results of this check are returned via *interface C*. In order to perform this check, *i.e.*, to authenticate the remote peer, the Security module may contact other peers via the remote *interface 3*. The Pricing module calculates an appropriate price for the service requested and sends it to the Negotiation module via *interface D*. In order to do so, it can retrieve information about past peer behavior from the A&C module via *interface E*. Furthermore, it can be configured by the Service Management module via *interface L*, *e.g.*, to support a new service or to adapt to a change in the service market. In order to make such decisions, the Service Management module can also retrieve information about the past from the A&C module via *interface K*. If a service negotiation is successful, the negotiation module uses *interface G* to send the final SLA to the Service Management module to start the service delivery.

After informing the A&C module about the forthcoming service delivery via *interface I*, the Service Management module instantiates, configures and starts a new service instance through *interface I.* During the service delivery, the service instance reports its status by sending events to the A&C module via *interface J.* The A&C module compares these events against the SLA and informs the Service Management module via *interface K* when necessary, *e.g.*, in the case of an SLA breach or when the service instance has finished. The Service Management module controls the service delivery via *interface I*, *e.g.*, stops it in the case of an SLA breach. For special purposes the A&C module can contact remote A&C modules, *e.g.*, to receive an immediate payment through tokens (cf. Chapter 32).

The middleware design has been described in more detail in [239] and [241]. A prototype has been implemented within the MMAPPS project [591] based on the JXTA framework [255]. The prototype serves as a proof of concept, showing the middleware enables the architecture presented in Section 29.2. It specifically fulfills the functional requirements of service support and market-based management, as well as the non-functional requirements of efficiency and accounting (cf. Section 29.1). The other requirements of decentralisation, scalability and reliability depend on the underlying Peer-to-Peer framework, JXTA. The middleware does not impede these requirements, since it does not introduce centralized entities nor unscalable protocols into the Peer-to-Peer environment.

29.3.2 PeerMart: Peer-to-Peer Auctions

Online auctions like eBay [183] are becoming increasingly popular marketplaces for trading any kind of services over the Internet. Auction-based markets benefit from the flexibility to adjust prices dynamically and enable to achieve efficient supply allocations (for an overview on auctions cf. [234]). However, those markets usually rely on a central component, i.e. the auctioneer which collects price offers of all participants and performs matches.

PeerMart, which is the second case study presented in this chapter, combines the advantages of an economically efficient auction mechanism with the scalability and robustness of Peer-to-Peer networks. It is shown, how PeerMart implements a variant of the Double Auction (DA) on top of a Peer-to-Peer overlay network, as an efficient pricing mechanism for Peer-to-Peer services. Other than in a single-sided auction, like the English Auction or the Dutch Auction, in the Double Auction both providers and consumers can offer prices. 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 services. PeerMart differs from existing work, such as [165] and [460], since it applies a structured rather than a random Peer-to-Peer overlay network, which enables deterministic location of brokers and is more efficient and scalable. It resolves the chicken and egg problem between providing incentives for services and being itself dependent on peers' functionality by introducing redundancy. Under the assumption, that a certain amount of peers behave correctly, PeerMart is thus able to provide a high reliability even in the presence of malicious or unreliable peers. Key design aspects of PeerMart are presented briefly in the following.

Basic Design

The basic pricing mechanism in PeerMart works as follows: Providers and consumers which are interested in trading a particular service, have to contact a responsible broker from which they can request the current price. Brokers (realized by clusters of peers) answer such requests with two prices:

– the current bid price, i.e. the current highest buy price offered by a consumer
– the current ask price, i.e. the current lowest sell price offered by a provider

Based on this information, consumers and providers can then send their own price offers (bids or asks) to the brokers. Continuously, brokers run the following matching strategy:

– Upon every price offer received from a peer, there is no match if the offer 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 peer 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.

To implement this mechanism in a decentralized manner, PeerMart uses Pastry [527], a structured Peer-to-Peer overlay infrastructure. The overlay is applied for peers joining and leaving the system, and to find other peers (brokers) in the network. 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. Furthermore, it is assumed that each service has a unique service identifier (serviceId). For content services this can be achieved, *e.g.*, by calculating the hash value of the content data. 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 network. The only varying service parameter considered at this stage is the price.

A set of n peers (called broker set) which are numerically closest to the serviceId are responsible to act as brokers for that service. Each peer in a broker set keeps an auction table for the service to store $m/2$ highest bids and

$m/2$ lowest offers (m can be adjusted depending on the service popularity). As it is assumed that multiple services are available, multiple broker sets will exist concurrently, each being responsible for a dedicated service. A broker set corresponds to a leaf set in Pastry. It consists of $n/2$ numerically closest larger nodeIds and $n/2$ numerically closest smaller nodeIds for a particular serviceId.

When a new service is offered for the first time, the corresponding root node (the node numerically closest to the serviceId) has to notify the other peers in its leaf set about the new service. If the root node fails to do that (*e.g.*, because it is a malicious node), the following fallback method can be applied. Recursively, peers on the path to the serviceId can be contacted, until the next closest node in the leaf set is found. This peer then takes over the responsibility of the root node and notifies the other peers. 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 which are notified to a PeerMart instance by its local Pastry node.

An example for the double auction mechanism in PeerMart is given in Figure 29.5. Two providers (P1, P2) and a consumer (C1) are interested in trading a particular service (serviceId x). Pastry routes their requests to the corresponding root node which returns the list of peers in the broker set. If the root node fails to return the broker set, the same fallback method as described above can be applied. Broker peers can now be contacted directly to get the current bid or ask price and to notify new price offers. Apart from the price, every offer contains a sequence number and a valid time and is signed with the peer's private key. The valid time cannot be larger than a maximum time t. For every peer only the newest offer is kept. After an offer becomes invalid it will be removed from the table.

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.

Broker Set 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 ($1 <= f <= n$). 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 above. Every broker peer forwards pairs of locally

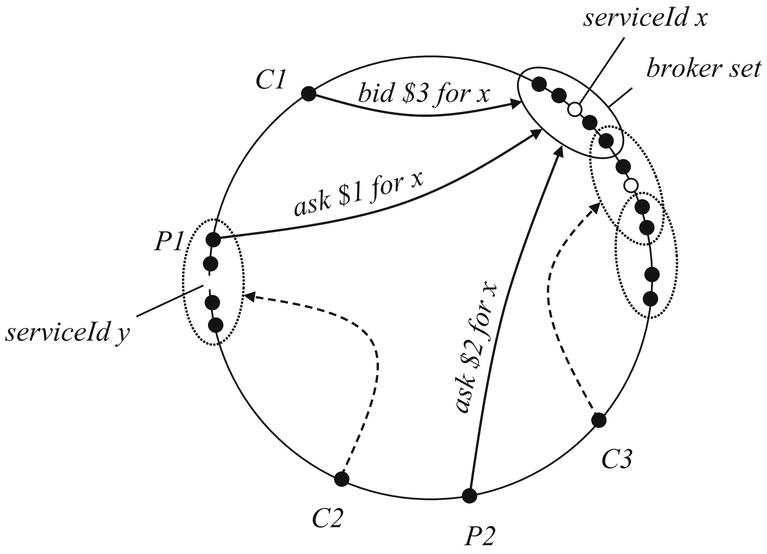

Fig. 29.5: Double Auction in PeerMart.

matching offers to all other peers in the broker set. Based on the signature of an offer, brokers can 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, 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.

A prototype of PeerMart has been implemented and is available as open source software for testing purposes [481]. More details about the implementation and results obtained from various experiments can be found in [282]. These results show that PeerMart provides a reliable, attack-resistant Peer-to-Peer pricing mechanism at a low overhead of messages and necessary storage space and scales well for any number of peers trading services. The mechanism is completely decentralized and suitable for trading any types of services. Hence, it fulfills all the requirements stated in Section 29.1.

29.4 Conclusion and Outlook

In this chapter, the economic and technical aspects of a market-managed Peer-to-Peer system were described in detail. A comprehensive list of requirements for such a system was given, and a generic architecture was outlined containing key components and mechanisms of a Peer-to-Peer-based marketplace. Finally, two case studies were presented, giving a more detailed view on two individual aspects of the overall approach. The first case study described modules of a Peer-to-Peer middleware which is centered around core service negotiation and management functionality. The second case study presented the design of an auction-based pricing mechanisms based on a Peer-to-Peer overlay network. Both studies covered specific parts of the proposed architecture.

Further important aspects exist and have been discussed in this chapter, such as accounting and reputation, which have not been covered in detail. Accounting is described separately in the next chapter.