Market Design & Analysis for a P2P Backup System*

Sven Seuken†
School of Engineering & Applied Sciences
Harvard University, Cambridge, MA
seuken@eecs.harvard.edu

Denis Charles, Max Chickering, Sidd Puri
Microsoft
One Microsoft Way, Redmond, WA
{cdx,dmax,siddpuri}@microsoft.com

ABSTRACT

This paper presents the design and theoretical analysis of a P2P resource exchange market, a novel application of markets to the domain of P2P backup. While the long-term goal is an open market using real money, here we consider a system where monetary transfers are prohibited. We first describe the design of the market and the user interface we developed. Second, we prove theorems on equilibrium existence and uniqueness. Third, we present a price update algorithm that uses daily supply and demand information to move prices towards the equilibrium. The market design described in this paper is already implemented as part of a Microsoft research project on P2P backup systems and an internal alpha version of the software has been released. We will thus complement the theoretical analysis with discussions of implementation challenges that arise in practice.

1. INTRODUCTION

The increasing dependence of our lives on information technology has resulted in a dependence on the continual access to our data. Regularly, users lose valuable data because their hard drives crash, their laptops are stolen, etc. Already in 2003, the annual costs of data loss in the US was estimated to be $18.2 Billion[1]. With internet broadband connections becoming faster and cheaper, *online backup systems* are becoming increasingly attractive alternatives to traditional backup solutions. There are hundreds of companies offering online backup services, e.g., SkyDrive, Idrive, Amazon S3. Most of these companies offer some backup storage for free and charge fees when the free quota is exceeded.

All of these services, however, rely on large data centers and thus incur immense costs. The motivation for P2P backup systems is that idle resources on the computers of millions of users can be used and thus these costs for centralized data centers can be avoided. The main idea is that users provide some of their resources (storage space, upload and download bandwidth) in exchange for using the backup service. We use *erasure coding* to minimize the storage overhead while providing the same reliability guarantees as centralized servers. The system we propose uses a hybrid P2P architecture where all backup data is transferred directly between peers, but a dedicated server coordinates the operations and maintains meta-data. The role of the server in this system is so small that standard load-balancing techniques can be used to avoid scaling bottlenecks.

A Brief Macroeconomic View

Just considering a resource costs analysis, there are multiple reasons to expect the P2P solution to be economically more efficient than the centralized alternative, even though it requires more storage space and generates more internet traffic. First, the largest cost factor in centralized data centers are the immense energy costs for cooling the servers. In a decentralized system, these cooling costs disappear and this saving significantly outweighs the additional energy costs for running previously idle hard drives. Second, many users have hard drives that are much larger than they really need. Using this unused space on users' home machines is more efficient than buying new hard drives for data centers. Third, internet bandwidth is particularly expensive at the peak usage. In the P2P system, the bandwidth load is naturally more distributed and thus usage peaks on individual wires can be avoided. Fourth, the maintenance costs for data centers are also significant and can be saved using the P2P solution because individual users buy a new machine every few years, even if they don't use the P2P backup system.

1.1 The Market Design Problem

Using decentralized peers to store data raises two market design challenges regarding *incentives* and *efficiency*. For the backup system to be reliable and sustainable, it must provide proper incentives to its users. Every user must provide a certain amount of all three resources, even if he currently only consumes one or two resources. For example, a user who only contributes storage space and no bandwidth is useless to the system because no files could ever be sent or received from that peer. Furthermore, it is natural that each user has different preferences regarding how much of each resource he wants to supply. Some users might be very constrained regarding the free space on their hard drive but might have a lot of bandwidth. Other users might have large hard drives but use their bandwidth for many other services like VOIP, file-sharing, etc. Thus, an accounting system that enforces the same resource ratios across all users is undesirable. The P2P backup market addresses this by associating each resource with a price such that the relative prices reflect the relative scarcity of the resources.

*We are thankful to David Parkes, Kamal Jain, and Alex White for helpful discussions on this work.

†Most of this work was done while the author was a research intern at Microsoft Live Labs.

[1]http://gbr.pepperdine.edu/033/dataloss.html

NetEcon'09, July 7, 2009, Stanford, California, USA.

1.2 Research Goals & Overview of Results

The motivation for this work is to develop a market design for a P2P backup service that can ultimately be used by millions of users. Thus, it was our goal to hide as many aspects of the market from the user as possible, including prices, account balances, etc. Our first contribution is the design of a P2P resource exchange system that works with minimal user interaction. Second, we introduce a new equilibrium concept that we term "buffer equilibrium". We explain why a traditional Walrasian equilibrium is not suitable for our domain and why the buffer equilibrium is desirable. Third, we provide theoretical proofs that under mild conditions, a buffer equilibrium exists in our market and is unique. Fourth, we then present a novel price update algorithm that takes system-wide supply and demand information to update prices such that the market is driven towards a buffer equilibrium over time. Our analytical and experimental analysis has shown that the price update algorithm converges reasonable quickly if the initial price vector is chosen close enough to the equilibrium. Whenever possible we complement the theoretical analysis with challenges that we encountered in practice.

1.3 Related Work

In recent years there has been much research on P2P storage systems, electronic markets, distributed accounting, resource exchange systems, etc. Almost 10 years ago, the research projects OceanStore [7] and FarSite [3] already investigated the potential of distributed file systems using P2P. Both projects, however, did not do any kind of market design. More recently, researchers have looked at the incentive problem, often with the primary goal to enforce fairness (i.e., you get as much as you give). Samsara [5], for example, is an accounting scheme that allows for fairness enforcement. In contrast to our design, however, their scheme is fully distributed. While such a design has some advantages, it prevents the use of sophisticated pricing and payment mechanisms.

The idea to use electronic markets for the efficient allocation of resources is even older than ideas regarding P2P storage systems. Already in 1996, Ygge et al. [14] proposed the use of computational markets for efficient power load management. More recently, grid networks and their efficient utilization have gotten more attention [8]. Fundamental to these designs is that participants are sophisticated users able to specify bids in an auction-like framework. While this assumption seems reasonable in energy markets or computational grid networks, we are targeting millions of users with our backup service and thus we cannot assume that users are able to directly act as traders on an exchange market.

The two papers most similar to our work are by Aperjis et al. [2] and Freedman et al. [6]. They analyze the potential of exchange economies for improving the efficiency of file-sharing networks. While the domain is similar to ours, the particular challenges they face are quite different. They use a market to balance supply and demand with respect to popular or unpopular files. However, in their domain there is only one scarce resource, namely upload bandwidth, while we must design an exchange market for three different resources (space, upload and download bandwidth).

To the best of our knowledge, there are currently only two companies offering a P2P backup service: *Wuala* and *AllMyData*. However, both companies do not have a market-based system, they do not elicit the users' preferences and they do not allow different users to provide different resource ratios. Thus, these systems exhibit large economic inefficiencies compared to our exchange market.

2. A P2P RESOURCE MARKET

2.1 Suppliers & Consumers

Each user in the P2P backup system is simultaneously a supplier and a consumer of resources. For every backup, there are multiple peers on the supplier side offering their resources and a single peer on the consumer side, needing these resources. A peer on the consumer side cannot simply use one supplier because unreliable storage is useless. Instead, the production process of the server (bundling multiple peers and coordinating them) in the middle is essential. Effectively, the server turns unreliable storage into reliable storage. Note that each peer on the supplier side offers a different bundle of resources (i.e., different storage space, availability, and bandwidth limits) while each peer on the consumer side gets the same product, i.e., a backup service with the same reliability. Each resource has a price at which it can be traded and in each transaction, the suppliers are paid for their resources and the consumers are charged for consuming services. Prices are updated regularly according to current aggregate supply and demand in the system, to bring prices into equilibrium.

2.2 Minimizing Replication: Erasure Coding

One natural concern about P2P backup systems is that individual P2P users have a much lower average availability than dedicated backup servers. Thus, a P2P system must maintain a higher file redundancy to guarantee the same file availability as server-based systems. Simply replicating the file multiple times would be very inefficient. Fortunately, we can significantly reduce the replication factor by using *erasure coding* (see [9] for an application of erasure coding to P2P storage). The erasure code splits up a file into k fragments, i.e., $F = (F_1, ..., F_k)$, and produces $n > k$ new fragments, i.e., transforms F into $G = (G_1, ..., G_n)$. The resulting ratio $\frac{n}{k}$ is called the *replication factor*. In contrast to simple replication, erasure coding ensures that *any* k of the n fragments are enough to reconstruct the file F which results in significant savings. For example, using simple file replication and assuming an average user availability of 12h/day, we would need approx. 17 replications to guarantee a file availability of 99.999%. In contrast, using erasure coding we can achieve the same availability with a replication factor of approx. 3. The use of erasure coding is reflected in the market prices for consuming and supplying space. For example, a consumer with an online time of 12h/day would have to pay approx. 3 times as much for consuming 1 GB of space than he would earn for supplying 1GB of space.

2.3 Operations in the Backup System

We consider the following five high-level operations:

1. **Backup**: When a user performs a backup, file fragments are sent from the consumer to the suppliers.
2. **Storage**: The suppliers persistently store the fragments they receive (until they are asked to erase them).
3. **Retrieval**: When a user retrieves a backup, file fragments are sent from the suppliers to the consumer.
4. **Repair**: When the server determines a backed up file to be unhealthy, the backup is repaired.
5. **Testing**: When sufficient availability information about a peer is missing, another peer backs up a file to the peer in question or retrieves a file from that peer.

Each of these operations requires a particular set of resources from the suppliers (see Table 1).

Operation	Resources Required from Suppliers
1. Backup	Download Bandwidth
2. Storage	Space
3. Retrieval	Upload Bandwidth
4. Repair	Download and Upload Bandwidth
5. Testing	Download and Upload Bandwidth

Table 1: Operations and their Required Resources.

2.4 Currency, Trading & Work Allocation

All trades in the market are done using "virtual currency", thus no real money is required. Trading is enabled via a centralized accounting system, where the server plays the role of a bank. The server maintains an account balance for each user starting with a balance of zero and allowing each user to take on a certain maximal deficit. In every trade, consumers pay credits to the server and the server pays credits to the suppliers. The primary purpose of the virtual currency is to allow users to do work at different points in time. Users have a steady inflow (from supplying resources) and outflow (from consuming services) of money which can vary over time. In *steady state*, when a user has been online sufficiently long, his flow consumption of backup services must be balanced by his flow provision of resources. However, when a users goes offline for a few days, he cannot earn money during that period but he still has to pay for his backed up files. Thus, his account balance will continuously decrease during that time period. Once the user reaches the maximum deficit level, he will not be allowed to make further backups before his balance increases again.

The server is involved in every operation and stores all relevant meta data for each file. For the backup, retrieval, testing and repair operations, the server either directly chooses which peers act as suppliers or sends a list of potential suppliers. Thus, the server has significant influence on how the work is allocated in the system. In the current implementation, the server chooses suppliers based on their account balance. In particular, the server allocates work to those users with the lowest account balance to drive all accounts (back) to zero over time. This is possible because the user's steady-state income must equal his expenditure. Thus, when a user has been online for a sufficiently long time, his account should be close to zero. One important implication for the user interface is that it is not necessary to show the user his account balance because it will usually be close to zero.

2.5 Resources, Services & Prices

It is important to understand the distinction between the three resources that must be supplied (storage space, upload and download bandwidth) and the three services that can be consumed (backup operation, storage, retrieval operation). Table 2 provides an overview and denotes the abbreviations:

Supplied Resources	1. Storage Space (S) 2. Upload Bandwidth (U) 3. Download Bandwidth (D)
Consumed Services	1. Backup (B) 2. Persistent Storage (S) 3. Retrieval (R)

Table 2: Resources vs. Services.

At first sight, it might be confusing that we did not list the user's online time, i.e., his availability a_i, as a separate resource. Note that suppliers are paid for their bandwidth per fragment they send or receive independent of how often they are online. However, their availability matters a lot for the payments they receive for the supplied storage space. Obviously, the higher a user's online time, the more useful is his supplied space, and the more will he get paid per fragment stored. In practice, the relationship between availability and payments is complex and not necessarily linear, however, the general idea is that a user gets paid for his "effective space supply" which is the product of the space he supplies, his availability, and an overhead factor (of approx. 2/3 in practice). For example, a user who supplies 30GB of space and is online 50% of the time has an effective storage supply of approximately 30GB $\cdot 0.5 \cdot 2/3 = 10$GB and that's what he gets paid for. To simplify notation we will let s_{iS} denote user i's effective storage supply. Together with the supplied upload bandwidth s_{iU} and the download bandwidth s_{iD} this defines a user's supply vector $s_i = (s_{iS}, s_{iU}, s_{iD})$. Using analogous notation we let $d_i = (d_{iB}, d_{iS}, d_{iR})$ denote the service consumption vector of user i. We use linear prices for the resources and services in the system. We let $p = (p_S, p_U, p_D)$ denote the price vector for supplied resources and $q = (q_B, q_S, q_R)$ denote the price vector for consumed services. Remember that in steady state (when the user is online and his account balance is close to zero), a user must be able to pay for his consumption with his supply. We can now express this *income/expenditure flow constraint* more formally using prices and the user's supply and demand. In steady state we require that: $s_i \cdot p = d_i \cdot q$.

3. THE USER

3.1 User Interface

The user interface is an essential aspect of the market design: it defines how the user sees the current market prices and how the server learns about the user's preferences. The resulting UI is novel, however, still in beta testing and needs to be examined in more detail. A more detailed description of the UI design can be found in [11]. Here, we only provide a brief description as necessary to motivate our user model.

The challenge regarding the UI design for this application is that the majority of the users of a P2P backup system will be non-experts without an understanding of the underlying market. We adopt as our goal that of *hidden markets*: the concept of prices and account balances must be hidden from the users as much as possible. This is in contrast to much of the previous work on electronic market design, where users are generally asked to specify bid/ask prices to interact with an auction protocol. Figure 1 displays our current implementation of the UI, a "settings window" where a user can control his resources and look at his resource balance.

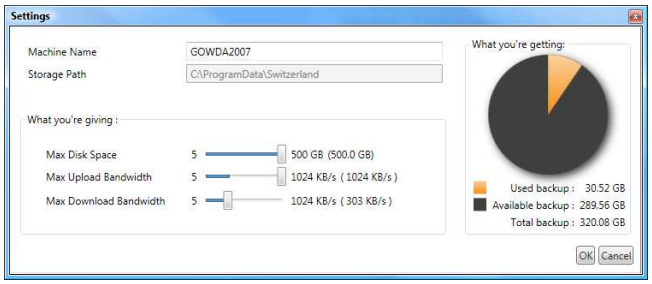

Figure 1: Screenshot of Current User Interface.

This window has two distinct areas: on the left side, the user can control his resources (his supply vector) and on the right side, the user sees his "resource balance" via a pie chart (an aggregate view of his current and maximally possible

consumption/demand vector). For each of his resources, the user can set a maximum cap for that resource by moving the sliders to specify how much of that resource the system should maximally use. The software will never use more of each resource than the user specifies via the sliders.[2]

When the user moves the sliders, his resource balance on the right changes simultaneously. The resource balance is determined by two numbers: 1) the user's current usage of online backup storage and 2) the additional backup storage space available to that user. The second value is effectively a prediction regarding how much more data the user could backup before hitting his resource caps. Here, we make use of the income/expenditure flow constraint again. Thus, by looking at the ratio of his current supply vector s and his maximum supply vector S, we can determine how much more he could consume. Note that this UI allows users to express their preferences. If a user wants to have a total available backup space of 100GB, there are different settings that will allow this. Some users might specify to give more space and less bandwidth, others might specify it the other way around, depending on the resources they have available and their individual preferences.

One highlight of this UI is that it allows the user to interact with the market, and see/experience the current market prices simply by moving the sliders and looking at the pie chart. If the user moves the slider a little and the resource balance only moves a little, this means that the current price for that resource is relatively low (because there already is a lot of supply for that resource). If the user moves a slider a little and the pie chart moves a lot, this implies that the current market price for that resource is relatively high (because this is a scarce resource).

Enforcing Resource Ratios
A challenge that we have ignored so far is the fact that we need all three resources from each individual user. The resources prices are automatically updated to bring demand and supply into balance, but only on an aggregate level. Given any price vector, a particular user might still want to choose to supply very little of one particular resource and very much of another. Thus, we also have to enforce that each user stays within certain resource ratios. In general, the available online space increases when the user increases one of his sliders. However, this is only true for a subset of possible slider positions. In particular, if a user keeps increasing one slider towards the maximum while the other two sliders are relatively low, at some point the online backup space stops increasing. For example, if a user limits his upload bandwidth to 5KB/s, increasing his space from 50GB to 100GB should not increase his online backup space. The reason is that we would never store 100GB on this user because 5KB/s would not be enough to have a reasonable retrieval rate. In the UI, we display the area where the slider has an effect on the resource balance as a thick line. Once the user moves one slider beyond the point where the line is thick, the resource balance on the right stops increasing. Moreover, when the user moves one slider the "effective slider regions" for the other resources (i.e., the thick lines) change accordingly. When one slider is moved up, the effective slider regions for the other two resources increase; when one slider is is moved down, the effective slider regions for the other two resources decrease. At which point the sliders must lose their effect and how the interaction between sliders and effective slider regions should be is a non-trivial question. In our implementation, we use system-wide information regarding the demand for each of the three resources in combination with a "slack factor" that tells us how flexible we can be in using a particular user's resources.

Handling Low Account Balances
The maximum deficit (in terms of credits) a user can take on is bounded. This implies that if a user has backed up some files, the user can only go offline for a limited period of time before his account will be frozen. If a user's account is low, the system alerts the user by sending him an email informing him of actions he could take.

[2]While the user has full control over his supply of space, upload and download bandwidth, a user's availability cannot be set explicitly but will be determined over time depending on how long the user is online. In our implementation, a complex algorithm estimates a user's availability over time.

3.2 The User Model

Naturally, each user has an endowment of the resources space, upload and download bandwidth. User i's endowment vector is denoted $\omega_i = (\omega_{iS}, \omega_{iU}, \omega_{iD})$. The UI gives the user the option to specify the *maximum supply* he is willing to give up. We let $S_i = (S_{iS}, S_{iU}, S_{iD})$ denote the maximum supply vector chosen by user i and we let $D_i = (D_{iB}, D_{iS}, D_{iR})$ denote the maximum consumption vector of user i. Given S_i, the flow-constraint $S_i \cdot p = D_i \cdot q$ automatically determines D_i. We assume that when the user sets the maximum supply vector S_i, he is forward-looking, i.e., he is planning for demand D_i in the future. In practice this means the user sets his supply such that he can continue using the backup software for a certain time period without running out of online backup space and thus without having to adjust the supply settings again. For every target amount of services D_i there are many different combinations of supplied resources that will allow the user to consume D_i, and different users might choose different trade-offs according to their preferences. To make the preferences monotone, we define the vector of resources that user i keeps as $K_i = \omega_i - S_i$. Now, we define each user i's preference relation as $\succeq_i$ over the six-tuple $(K_{iS}, K_{iU}, K_{iD}, D_{iB}, D_{iS}, D_{iR})$ and we make the following standard assumptions:

Assumption 1. Each user's preferences are (i) continuous, (ii) convex, and (iii) monotone.

Furthermore, we make a few well-known (cf. [10]) observations that will be useful later:

Observation 1. (Continuous, Quasi-Concave Utility Function and Homogeneity of Degree Zero)

(i) Given that the user's preferences are continuous, there exists a utility function $u_i(K_{iS}, K_{iU}, K_{iD}, D_{iB}, D_{iS}, D_{iR})$ that represents the preference relation and this utility function is continuous.

(ii) Given that the user's preferences are convex, we know that u_i is quasi-concave.

(iii) Given that we are in a closed economy, the supply and demand function of the individual users are homogeneous of degree zero, i.e., only relative prices matter. This implies that the aggregate supply and demand functions are also homogeneous of degree zero.

3.3 The Utility Maximization Problem

Definition 1. (Utility Maximization Problem) Given the endowment ω_i, the demand d_i, the current price vectors p and q, we model user i's UMP as follows. Let K_i be the resources he keeps, and let D_i be his expected future demand.

Maximize $u_i(K_i, D_i)$ under the following constraints:

$$S_i = (\omega_i - K_i) \quad (1)$$
$$D_i \cdot q = S_i \cdot p \quad (2)$$
$$d_i \cdot q \leq S_i \cdot p \quad (3)$$
$$S_{iX} \leq S_{iY} \cdot r_{XY}, \text{ where } X \neq Y \in \{S, U, D\} \quad (4)$$
$$S_i, K_i, D_i \geq 0 \quad (5)$$

Explanation: Line (1) defines the supply vector. Line (2) expresses the constraint that the amount a user supplies defines how much he can maximally consume. The UI will show the user how much he can consume, i.e. D_i, given his current supply choice S_i. Line (3) makes sure that the user chooses to supply enough such that he can at least afford his current demand (otherwise he would quickly run into a deficit). Line (4) denotes the *ratio constraints*, i.e., a user's supply of each resource has to be within a certain ratio of the supply of both other resources. The coefficients r_{XY} are calculated based on the aggregate demand ratios and a slack factor that depends on how flexible the system's work allocation method is. Finally, line (5) makes sure that all choice variables are ≥ 0.

It turns out that, if we know the overhead resulting from the erasure coding and the amount of repair and testing operations the system needs to perform, then the supply price vector p automatically determines the service price vector q. Thus, to simplify notation we will only use p for the remainder of this paper. Furthermore, for the equilibrium analysis, we will consider the aggregate *demand of all resources* $d = (d_S, d_U, d_D)$ which can be calculated analogously from the aggregate demand of all services. In the remainder of the paper we will often just refer to *supply* or *demand* and it will be clear from the context whether we mean the *currently used* aggregate supply and demand s and d, or the *maximum* aggregate supply and demand S and D.

4. EQUILIBRIUM ANALYSIS

A standard equilibrium concept in General Equilibrium Theory is the Walrasian equilibrium where demand equals supply. While this equilibrium is reasonable for one-time market models, it is not suitable for an ongoing market like ours where users only slowly react to price changes. In particular, we do *not* want to balance the market. We must guarantee that the backup system can always satisfy new requests (demand) which implies that we must always have some excess supply of all resources. Thus, a Walrasian equilibrium is certainly not the correct concept for this system.

4.1 The Buffer Equilibrium

To guarantee that the backup system always has enough supply for each individual resource to satisfy new incoming requests, we must always aim for a large enough buffer between the aggregate demand and the aggregate supply. Remember that the idea of market prices is that the prices reflect the scarcity of the resources in the system. For example, if all users value upload bandwidth highly and would rather supply storage space or download bandwidth instead, then upload bandwidth should have a high price such that the users are incentivized to supply more upload bandwidth. When we are updating prices, we are effectively increasing the buffer for some resources and decreasing it for others. Given that we cannot predict which resource demand will increase the most in the future, we assume a uniform distribution. Thus, to minimize the risk of ever not having enough supply for any resource we maximize the minimum buffer between supply and demand over all resources. This naturally leads to the following new equilibrium concept:

Definition 2. (Buffer Equilibrium) A buffer equilibrium is a price vector $p = (p_S, p_U, p_D)$, an aggregate supply vector $S(p) = (S_S(p), S_U(p), S_D(p))$ and aggregate demand vector $d(p) = (d_S(p), d_U(p), d_D(p))$ such that

$$\frac{S_S(p)}{d_S(p)} = \frac{S_U(p)}{d_U(p)} = \frac{S_D(p)}{d_D(p)}$$

i.e., the buffer between aggregate demand and aggregate supply is the same for each resource.

It is straightforward to show that in a buffer equilibrium, for every resource, the aggregate supply is at least as high as the aggregate demand. How much higher the supply is depends on how much extra supply the users are provisioning on average (beyond what's necessary to afford their current consumption). See [12] for more details.

4.2 Existence of the Equilibrium

We don't want to assume that the user's preferences are strongly monotone and strongly convex w.r.t. to all resources because we believe that this is not the case for the service products. Instead, we will work with the following, more restrictive assumption:

Assumption 2. (Service Products are Perfect Complements) We assume that the user considers backup, storage and retrieval to be service products that are perfect complements. A utility function that induces the perfect complements property is given by

$$u_i(D_i) = \min\{\beta_{i1} D_{iB}, \beta_{i2} D_{iS}, \beta_{i3} D_{iR}\}.$$

Discussion: Note that this assumption does not mean that every user consumes the services in the same fixed ratios. What it means is that each user has a certain fixed usage pattern that defines the ratio of the three services for this individual user. If prices or the user's endowment change, the demand for the three service products changes, but the ratios remain the same.

THEOREM 1. *A buffer equilibrium exists in the P2P exchange economy, given that users' preferences are continuous and convex, given that they are monotone w.r.t. service products, strongly monotone w.r.t. to supply resources, and given that service products are perfect complements.*

Due to space constraints we could not include the proof in this paper. Please see our working paper [12] for the full proof.[3]

4.3 Uniqueness of the Equilibrium

For the uniqueness theorem, we need the following assumption:

Assumption 3. (Supply Resources are Gross Substitutes) We assume that the aggregate supply function $S(p)$ satisfies the gross substitutes condition [1], i.e., whenever p' and p are such that, for some l, $p'_l > p_l$ and $p'_k = p_k$ for $k \neq l$, we have $S_k(p') > S_k(p)$ for $k \neq l$.

THEOREM 2. *The buffer equilibrium is unique, given that users' preferences are continuous, convex, and monotone, that services are perfect complements (Assumption 2) and that supply resources are gross substitutes (Assumption 3).*

[3]The working paper is available online at `http://www.eecs.harvard.edu/~seuken/P2PBackup.pdf`.

PROOF. From the definition of the equilibrium it is clear that we only care about supply relative to demand. We want to find prices such that the supply vector is a multiple of the demand vector, i.e., $S(p) = \lambda d(p)$ for $\lambda \in \mathbb{R}^+$. Thus, to find equilibrium prices, we can also look at projective space for the aggregate supply and demand vectors where the supply vector must *equal* the demand vector. To derive this directly, we start in affine space where we need: $\exists\lambda \in \mathbb{R}^+ : (S_S, S_U, S_D) = \lambda\,(d_S, d_U, d_D)$. If we normalize supply and demand of storage space to 1 we get: $\exists\lambda' \in \mathbb{R}^+ : \left(1, \frac{S_U}{S_S}, \frac{S_D}{S_D}\right) = \lambda' \left(1, \frac{d_U}{d_S}, \frac{d_D}{d_S}\right)$. This equation can only hold for $\lambda' = 1$. And thus, in $\mathbb{RP}^2$ we get: $\left(\frac{S_U}{S_S}, \frac{S_D}{S_S}\right) - \left(\frac{d_U}{d_S}, \frac{d_D}{d_S}\right) = 0$. We define a new vector-valued function $g(p) = \big(g_1(p), g_2(p)\big)$ where $g_U(p) = \left(\frac{S_U(p)}{S_S(p)} - \frac{d_U(p)}{d_S(p)}\right)$ and $g_D(p) = \left(\frac{S_D(p)}{S_S(p)} - \frac{d_D(p)}{d_S(p)}\right)$. Now, we can re-write the definition of the buffer equilibrium as follows:

Definition 3. (Buffer Equilibrium [2. Alternative]) A buffer equilibrium is a price vector p and $g(p)$ such that

$$g(p) = (0, 0)\,.$$

We have simplified the problem of finding equilibrium prices to finding the root of $g(p)$. The uniqueness of the buffer equilibrium is equivalent to saying that $g(p) = 0$ has at most one (normalized) solution. The remainder of the proof is given in [12]. □

5. THE PRICE UPDATE ALGORITHM

In this section we devise a price update algorithm that is invoked regularly on the server (e.g., once a day), with the goal to move prices towards the buffer equilibrium over time. Remember that the users' utility functions are homogeneous of degree zero. Thus, only relative prices matter and w.l.o.g., we can fix the price for storage space at 1 and only update the prices for upload and download bandwidth.

5.1 The Algorithm

Our price update algorithm is oriented at the tâtonnement process as defined by Walras about 200 years ago [13]. However, with Walras' algorithm, trades were only allowed at equilibrium prices. In our system, however, we must allow trades at all times, even out of equilibrium.

In Section 4.3, we have reduced the problem of finding the buffer equilibrium to finding the root of the function $g(\cdot)$. This is a well-understood mathematical problem. *Newton's method* is probably the best-known root-finding algorithm, which also converges very quickly in practice. However, it requires the evaluation of the function's derivative at each step. However, we don't know the function $g(\cdot)$and thus cannot compute its derivative. Instead, we only get to know individual points in each iteration and can use these points to estimate the derivative. This is exactly what the *Secant method* does for a one-dimensional function.

The problem is that $g(p)$ is 2-dimensional, and thus the secant method is not directly applicable. The appropriate multi-dimensional generalization is *Broyden's method* [4], a quasi-Newton method. Unfortunately, that method requires knowledge of the Jacobian, which we don't know and also cannot even measure approximately. However, we show that one can use an approximation to the diagonal sub-matrix of the Jacobian instead of the full Jacobian matrix. The diagonal sub-matrix of the Jacobian can be approximated by studying changes in the functions $g_X(p)$. This leads to the following quasi-Newton method for multiple dimensions:

Definition 4. (The Price Update Algorithm)

$$p_X^{t+1} = \begin{cases} 1 & \text{for } X = S \\ p_X^t - \frac{p_X^t - p_X^{t-1}}{g_X(p^t) - g_X(p^{t-1})} \cdot g_X(p^t) & \text{for } X = U, D \end{cases}$$

For a real-world implementation of the price update algorithm we have to take care of some special cases. We discuss a modified price update algorithm for these cases in [12].

5.2 Convergence Analysis

We have performed an analytical and experimental analysis of the convergence properties of the price update algorithm. However, due to space constraints we can only briefly discuss some of our main findings. Please see [12] for a detailed discussion and proofs.

To prove any formal guarantees regarding the convergence of the price update algorithm, we needed a series of technical assumptions, including that the function $g(p)$ defined in Section 4.3 is a continuously differentiable function. However, the more relevant assumption for practical purposes is that the initial price vector p^0 that is used to start the price update algorithm needs to be closes enough to the root of $g(p)$. This is a property that all quasi-Newton methods have in common. If such a starting point can be found then our price update algorithm converges at least Q-superlinearly to the root. Unfortunately, if a price vector too far away from the root is chosen, no formal guarantees can be given.

We also performed an experimental convergence analysis of the price update algorithm, using a simulated market with 100 agents. We used the well-known CES utility functions to make sure that we satisfy the assumptions of our theorems. With the help of an NLP-solver we were able to solve each agent's utility maximization problem after each round of prices updates. Similarly to the analytical analysis, we found that when a price vector somewhat close to the root was chosen, the update algorithm indeed converged very quickly. However, sometimes, when a price vector far away from the root was chosen, the algorithm never converged. Further experiments are necessary, in particular we will study how the price update algorithm reacts to periodic demand and supply shocks, what happens if individual users drop out of the market, and what happens if the users' demand changes exogenously. However, in practice, finding reasonably good starting prices might be the biggest challenge.

6. CONCLUSION

In this paper, we presented the market design for a novel P2P backup application in which monetary transfers are prohibited. One of the design goals was to allow the users to interact with the market without specifying bid or ask prices. In this setting, we proved the existence and uniqueness of a buffer equilibrium. We defined a price update algorithm and we have shown that it converges to the buffer equilibrium, given that the initial starting prices were chosen close enough to the equilibrium. In ongoing work we are augmenting the market design with a payment mechanism, defined in terms of market prices, that will provide for robustness against strategic deviations from users that manipulate the protocol, e.g., by reprogramming their software client. In future work we will extend our current design and allow for monetary payments. Thus, on the one side, users will then be able to pay for their consumption of services by either providing their own resources or by paying with real money, and on the other side, users will then also be able to earn real money by supplying their resources.

7. REFERENCES

[1] J. Alexander S. Kelso and V. P. Crawford. Job matching, coalition formation, and gross substitutes. *Econometrica*, 50(6):1483–1504, 1982.

[2] C. Aperjis and R. Johari. A peer-to-peer system as an exchange economy. In *Proceedings from the Workshop on Game Theory for Communications and Networks (GameNets)*, Pisa, Italy, October 2006.

[3] W. J. Bolosky, J. R. Douceur, and J. Howell. The farsite project: A retrospective. *SIGOPS Operating Systems Review*, 41(2):17–26, 2007.

[4] C. G. Broyden, J. E. Dennis Jr., and J. J.Moré. On the local and superlinear convergence of quasi-newton methods. *J. Inst. Math. Appl.*, 12:223–245, 1973.

[5] L. P. Cox and B. D. Noble. Samsara: Honor among thieves in peer-to-peer storage. In *Proceedings of the nineteenth ACM symposium on Operating systems principles (SOSP)*, pages 120–132, Bolton Landing, NY, 2003.

[6] M. J. Freedman, C. Aperjis, and R. Johari. Prices are right: Managing resources and incentives in peer-assisted content distribution. In *Proceedings of the 7th International Workshop on Peer-to-Peer Systems*, Tampa Bay, FL, February 2008.

[7] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[8] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman. Tycoon: An implementation of a distributed, market-based resource allocation system. *Multiagent Grid Systems*, 1(3):169–182, 2005.

[9] J. Li and C. Zhang. Distributed hosting of web content with erasure coding and unequal weight assignment. In *Proceedings of the IEEE International Conference on Multimedia Expo*, pages 27–30, Taipei, June 2004.

[10] A. Mas-Colell, M. D. Whinston, and J. R. Green. *Microeconomic Theory*. Oxford University Press, 1995.

[11] S. Seuken, D. Charles, M. Chickering, and S. Puri. Designing user interfaces for hidden markets. In *Proceedings of the IJCAI Workshop on Interaction and Intelligence*, Pasadena, CA, July 2009.

[12] S. Seuken, D. Charles, M. Chickering, and S. Puri. Market design & analysis for a p2p backup system. Working Paper, 2009.

[13] L. Walras. *Eléments d'économie politique pure; ou, théorie de la richesse sociale (Elements of pure economics; or, the theory of social wealth)*. Corbaz, Lausanne, 1874.

[14] F. Ygge and H. Akkermans. Power load management as a computational market. In *Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS)*, pages 393–400, Kyoto, Japan, 1996.