Making P2P Accountable without Losing Privacy

Mira Belenkiy, Melissa Chase, C. Chris Erway, John Jannotti,
Alptekin Küpçü, Anna Lysyanskaya, Eric Rachlin
Department of Computer Science, Brown University
Providence, RI, USA
{mira,mchase,cce,jj,kupcu,anna,eerac}@cs.brown.edu

ABSTRACT

Peer-to-peer systems have been proposed for a wide variety of applications, including file-sharing, web caching, distributed computation, cooperative backup, and onion routing. An important motivation for such systems is *self-scaling*. That is, increased participation increases the capacity of the system. Unfortunately, this property is at risk from selfish participants. The decentralized nature of peer-to-peer systems makes accounting difficult. We show that *e-cash* can be a practical solution to the desire for accountability in peer-to-peer systems while maintaining their ability to self-scale. No less important, e-cash is a natural fit for peer-to-peer systems that attempt to provide (or preserve) privacy for their participants. We show that e-cash can be used to provide accountability without compromising the existing privacy goals of a peer-to-peer system.

We show how e-cash can be practically applied to a file sharing application. Our approach includes a set of novel cryptographic protocols that mitigate the computational and communication costs of anonymous e-cash transactions, and system design choices that further reduce overhead and distribute load. We conclude that provably secure, anonymous, and scalable peer-to-peer systems are within reach.

Categories and Subject Descriptors: E.3 [**Data**]: Data Encryption; C.2.2 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Protocols*

General Terms: Algorithms, Design, Economics, Security

1. INTRODUCTION

Peer-to-peer systems leverage the cooperative exchange of resources between peers to provide useful services, but lack strong mechanisms to enforce this cooperative behavior. Their decentralized nature provides many advantages, including fault-tolerance and scalability, yet seems to preclude simple enforcement mechanisms such as quotas and strong reputations.

Measurement studies and theoretical analyses have shown selfish behavior common among users of peer-to-peer file-sharing systems, and effective for obtaining an unfair share of resources [1, 29, 35, 34]. The most popular file-sharing system, BitTorrent, employs a "tit-for-tat" mechanism aimed at encouraging fairness, but a recent study has suggested that BitTorrent's effectiveness is due largely to the altruistic behavior of a small number of high-bandwidth nodes [42] rather than fair contributions from all participants. Most nodes leave the system as soon as they finish downloading; only the most altruistic continue to donate upload capacity.

BitTorrent's apparent dependence on a small percentage of altruistic nodes indicates that it may not exhibit the fault-tolerance and scalability that one would expect when considering the protocol in the abstract. If a small number of altruistic users change their habits or fail, BitTorrent's effectiveness would be at risk. Worse, in other peer-to-peer contexts such as distributed backup, altruism is less likely since non-renewable resources such as disk space are exchanged. A repeat of BitTorrent's success in these applications appears unlikely without techniques to eliminate selfish behavior and reliance on altruism.

We would like to build systems that provide fungible benefits to cooperative nodes, rewarding those that continue to provide service with credit towards future service. These credits and debits eliminate free-riders by forcing every user to contribute as much as he receives from the system. Equitable exchange would make the system significantly more robust, as it would no longer be dependent on the contributions of altruistic users. These credits should be secure, anonymous, and fungible. Further, the complete system should continue to self-scale.

In this paper, we focus on how a BitTorrent-like file-sharing system might use endorsed e-cash, because the dynamics of BitTorrent are better understood than less popular peer-to-peer systems. However, we believe that e-cash is more widely applicable, and outline how to apply endorsed e-cash to applications such as distributed lookup, computation, storage, and onion routing.

Our contribution is an accountability mechanism for file-sharing applications. Our mechanism is provably secure, reasonably efficient in terms of computational and network overhead, avoids centralized knowledge of client behavior, and provides fungible benefits so that contributions can be recouped without requiring pair-wise exchanges. We believe that this currency can be used to provide practical robustness and accountability in peer-to-peer systems without compromising privacy.

This paper is organized as follows. We begin with a re-

WPES'07, October 29, 2007, Alexandria, Virginia, USA.

view of fairness and incentives in BitTorrent in Section 2. We review the requirements on our currency and introduce *endorsed e-cash* in Section 3. The application of our accounting mechanism to BitTorrent-like systems is described in 4. Our novel protocols for buying and bartering data are presented in Section 5. In Section 6 we evaluate the performance costs of our privacy goals, and describe practical measures to mitigate them, providing simulations to measure their effectiveness. We discuss economic issues introduced by our currency-based design in Section 7. Other possible applications of e-cash to distributed systems are described in Section 8. We provide an overview of related work in reputation and currency-based resource allocation in Section 9. Finally, we conclude in Section 10.

2. BACKGROUND

To motivate our call for better accountability in peer-to-peer systems, we consider the popular file distribution system BitTorrent [15].

BitTorrent. A BitTorrent download begins with a metadata file, or *torrent*, listing the name and size of the file(s) for distribution, the hash value of each data block (typically sized 64KB–2MB) needed to assemble the file(s), and the address of a *tracker* assigned to manage the swarm of participants. A node enters the network by announcing itself to the centralized tracker server. The tracker's reply provides the Internet addresses of a random subset of active nodes.

Using this list, a node makes neighbors by attempting to directly connect to other nodes. Upon connecting, neighbors start to exchange block-availability advertisements and issue block requests, using a local "rarest-first" heuristic designed to improve data availability. Nodes typically maintain many neighbor connections, but at each round upload requested blocks only to a handful of *unchoked* neighbors, selecting those that have recently provided the fastest rate of service. The rest are *choked* until the next round, typically ten seconds later, when they will be re-evaluated for reciprocation.

Incentives. BitTorrent relies on a tit-for-tat bandwidth reciprocity mechanism whereby cooperative nodes are rewarded for uploading by their neighbors, who *unchoke* them. Thus nodes are encouraged to upload to their neighbors, since doing so offers the possibility of faster download rates. However, there are still opportunities for selfishness [42, 34] and free-riding [29, 35]. In search of ever-better neighbors, nodes select two at random for *optimistic unchoking* every three rounds; this also helps to bootstrap new users, who may not have enough data to reciprocate. Once a download is complete, nodes receive no benefit from participating, and most depart [42]; nodes that persist become altruistic *seeders*. By providing unaccounted service, both optimistic unchoking and seeding may potentially enable free-riding.

The tracker has limited ability to perform resource accounting: when nodes periodically re-announce themselves (*e.g.*, to check for new nodes), they report the total amount of data uploaded and downloaded for the torrent so far. Online communities have emerged around registration-only trackers that enforce sharing ratios by totaling user activity across multiple torrents, leading to increased seeding activity [3]. However, the basis for these ratios comes from self-reported client data and can be trivially forged [35, 25].

Fairness and performance. Studies have found the tit-for-tat choking mechanism highly effective in discouraging free-riding, encouraging the clustering of similar-bandwidth peers, and fully utilizing participants' uplink bandwidth.

Legout *et al.* [32] highlight these properties through experiments on PlanetLab, limiting each node's maximum upload rate to define different classes of participation. Their measurements show that, in a "flash crowd" scenario with a fast seed, nodes are more likely to exchange data with neighbors of their own class, forming similar-bandwidth clusters. This follows from the intuition that, to a faster node, a slower neighbor's unchokes are less useful, so are less often reciprocated. As a result, higher-class participants are rewarded with faster downloads. The simulation-based study of Bharambe *et al.* [6] also finds the choking mechanism effective in fully utilizing upload bandwidth.

However, both studies also provide evidence that high uplink utilization (and the resultant fast average download times) comes at the expense of *fairness*: fast nodes typically contribute much more data than they upload. In [6] the authors find that in heterogeneous settings, some high-bandwidth nodes upload more than seven times more data than they download. The experiments of [32] also observe similar unfairness, and show it is exacerbated as clustering breaks down in the presence of a slow seed. Piatek *et al.* [42] observe and model unfairness as *altruism*—their term for any data uploaded in excess of that downloaded—which they exploit with a selfish client implementation.

Unfairness can be partially explained by the slow search process nodes undertake when looking for better neighbors, with optimistic unchokes occurring only once every 30 seconds. Converging on a set of similar-bandwidth peers in a large, high-churn torrent may take a long time (or forever), and along the way many high-capacity nodes provide unmetered service to slower neighbors [42]. In response, [6] considers modifying the tracker, allowing it to induce clustering by matching similar-bandwidth nodes. The authors report that reducing bandwidth-mismatched pairings leads to improvements in both uplink utilization and fairness.

This has led some researchers to design alternate choking mechanisms that provide stronger fairness to participants. A strict block-level tit-for-tat choking policy, bounding the number of excess blocks transferred to a neighbor without reciprocation, is considered as a replacement for the default rate-based policy in [6]. It is shown to reduce unfairness at the expense of lower average upload utilization, slowing overall performance (though its performance fares better in conjunction with a bandwidth-matching tracker, or at high node degree). Fan *et al.* [20] present an analytical model characterizing the design space of rate assignment strategies in BitTorrent-like systems, demonstrating a fundamental trade-off between optimal performance and fairness.

We aim to provide strong fairness with e-cash, incentivizing user contributions with fungible credits redeemable for future service. This approach offers benefits beyond even an optimally fair local barter policy: currency fungibility allows nodes to participate fairly without the constraint of mutual coincidences required by bartering. However, requiring strict data volume fairness may fundamentally change BitTorrent's performance. Free-riders accustomed to unmetered service from altruistic nodes would instead be required to contribute at least as many resources as they use. Though a currency-based approach would clearly eliminate free-riding (by requiring nodes to contribute enough service to maintain financial solvency), it may impact performance

by limiting the amount of altruism available for slow nodes. This may lead some to ask why strong accounting, or even fairness, is desirable.

We argue that fairness is essential to providing scalable incentives for greater participation. Dependence on a beneficent minority makes a system more vulnerable to failure and performance less predictable. Fair incentives are especially important in systems like BitTorrent, where users can select their level of contribution. Presented with a system that does not provide service in proportion to one's contribution, a user will rationally prefer to provide less service, rather than waste resources for diminishing gains. Further, users may wish to avoid a system perceived as unable to prevent "selfish" or free-riding activity; the availability of software for this purpose [42, 35, 25] reinforces this view. In contrast, strict fairness is simple to understand and leverages the contributions of all users, not just the fastest and kindest. We believe that a system based on this idea—that all nodes contribute at least what resources they use—provides incentives inherently more scalable than one that allows free-riding.

3. OVERVIEW OF E-CASH

We need to choose an appropriate currency to incentivize peer-to-peer systems. The currency must be *fungible*: a user must be able to be paid for service and spend this payment for service from any other user. The payment protocol should ensure a *fair exchange* of money for the content: either the seller gets paid and the user gets the content, or neither of them gets anything. Users should be able to spend the currency *anonymously*: there should be no way to link an e-coin to the user that spent it, even if the bank colludes with all sellers. The currency must also be *unforgeable*: users should not be able to forge money (or at least be caught and punished when they try). Finally, we want our currency to be *efficiently implementable*.

3.1 E-cash

One currency that satisfies all our requirements is *e-cash* [12, 13], which offers the following properties:

Anonymity. E-cash ensures that it is impossible to trace an e-coin to the user who spent it, even when the bank colludes with all the sellers. It is not even possible to tell *if* two different e-coins were spent by the same user. The only exception is if a user tries to *double-spend* an e-coin. In this case, the bank can learn the identity of a cheater (and in some cases even trace *all* e-coins the dishonest user ever spent), and punish him accordingly.

Unforgeability. A dishonest user might try to spend the same e-coin more than once. The simplest solution to this problem is to use *on-line* e-cash: the seller consults the bank during every transaction to ensure the promised e-coin is new. However, this approach places a heavy burden on the bank and compromises the privacy of the user (if a fair exchange fails, new e-coins can be linked to old ones), and thus is not preferable.

In *off-line* e-cash, a buyer and seller perform a transaction without consulting the bank. At some later point, the seller can deposit all of the coins he has received. The bank will then check whether any of these coins have been spent before. If a user spends the same e-coin more than once, the bank can use the forged e-coin to identify the user. We must note that, at this point, the user has already carried out multiple transactions and obtained more content than to which he is entitled. It is up to the system to devise a punishment sufficient to deter forgery.

The bank can limit the ability of dishonest users to forge e-coins indefinitely, by publishing an Authorization List of users who are permitted to spend money off-line. The list would in reality be condensed to single value, called an accumulator [8]. Users can prove that they are in the accumulator without revealing their identities. As long as sellers receive frequent updates to the Authorization List, a user would not be able to double-spend many times. If a user double-spends, the bank can simply remove him from the accumulator and punish him (*e.g.*, by banishing his account).

Fair exchange. To be useful in our system, the e-cash scheme must enable users to perform a fair exchange of e-coins for digital content. All fair exchange protocols require a trusted third party (TTP) that is responsible for resolving disputes. In our system, we refer to the TTP as the *arbiter*, to emphasize its potential difference from the *bank*. In an optimistic fair exchange protocol, the arbiter is only involved when one party cheats. There has been extensive prior work on fair exchange, however none of the existing protocols is sufficient for our application.

We need to provide a protocol for exchanging an e-coin for a file. Jakobson [28] and Reiter, Wang, and Wright [43] provide protocols for exchanging e-coins for data. However, with both the user loses his e-coin if the exchange fails. Asokan, Shoup and Waidner [4] provide a protocol which allows a user to reuse his e-coin after a failed exchange, but unlinkability is no longer guaranteed for the reused coin. Furthermore, all of the above work requires that e-coins be withdrawn one by one from the bank.

The best solution to date is Camenisch *et al.*'s endorsed ecash [9], which allows users to withdraw and store many coins at once, and to respend coins after failed exchanges with the guarantee that these coins will still be unlinkable. However, this protocol has the disadvantage that in case of a conflict, the arbiter must download and verify an entire file. We modify this protocol so that the arbiter only has to examine a short proof provided by one of the parties.

As generating e-coins is somewhat expensive, we also wish to provide a more efficient protocol for bartering files. There is no existing protocol that addresses this situation. Bao, Deng, and Mao [5] provide protocols for fair exchange of signatures, but not of other forms of data. Asokan *et al.* [4] provide a protocol for fair exchange of data, but they assume that there is an online verifier who verifies each block of an encrypted file and provides a signature on that file. We provide an efficient protocol which allows two users to exchange files given only that both know the correct hashes on those files. See section 5 for more details.

3.2 Endorsed e-cash

In endorsed e-cash, a central bank maintains an account for each user. Users can withdraw a wallet of multiple e-coins from the bank in one efficient operation. Users can spend their e-coins anonymously by performing an efficient fair exchange of e-coins for digital content. However, users must deposit e-coins they have earned back into their bank accounts before spending them again.

An endorsed e-cash transaction proceeds as follows: to spend an e-coin, the buyer chooses a random value, called the *endorsement*, and uses it to encrypt the e-coin to get

coin′. The buyer gives the seller *coin′*, but retains the *endorsement*. The seller verifies that (1) *coin′* is a valid unendorsed e-coin and (2) the buyer knows an *endorsement* that will decrypt the e-coin. Once the seller is satisfied, the buyer and seller perform a fair exchange of the endorsement for the content. It is possible to simultaneously exchange several e-coins for multiple goods and services. If a fair exchange fails, the buyer re-encrypts the e-coin with a new randomly chosen *endorsement′*. The new *coin″* cannot be linked to the original coin, the previous *coin′*, or the user, *unless the coin is double-spent*.

Fair exchange. Endorsed e-cash allows us to perform an efficient optimistic fair exchange on e-coins, which is the unique feature of endorsed e-cash. We used endorsed e-cash to create two new fair exchange protocols, for buying and exchanging files, that place a significantly smaller load on the arbiter than prior work. To ensure fairness, the arbiter needs to (1) download the unendorsed e-coin (this requires it to download just one integer) and (2) verify that the seller's supplied content is correct. In the case of the buy protocol, the arbiter randomly tests parts of the file to verify correctness. In the case of the barter protocol, the aggrieved party supplies the arbiter with a short proof that a segment of the file is corrupted. The details of our new, efficient fair exchange protocol using endorsed e-cash is given in Section 5.

Enforceable contracts. Each endorsed e-coin is explicitly associated with a contract. Only the owner of the e-coin can create the contract. The buyer gives the contract to the seller along with the encrypted *coin′*. The seller can either accept the terms of the contract and initiate the fair exchange protocol, or reject the contract and simply terminate. Endorsed e-cash [9] primarily use the contract to provide some necessary randomness for the e-coin. In this work, we explore how to use this contract to explicitly ensure fairness, via the arbiter. Our two new fair exchange protocols in Section 5 use specially structured contracts to allow peers to buy and exchange files. In Section 8, we explore how to use contracts to enforce fairness in distributed look-up, distributed storage, and distributed computation.

4. CURRENCY-BASED DESIGN

We now describe a peer-to-peer content distribution system that uses e-cash to provide strong accountability. We draw inspiration from BitTorrent, but strengthen its loose barter accounting with two new protocols allowing a node to *buy* and *barter* encrypted data blocks from its neighbors. These protocols, detailed in Section 5, enable these transactions through the fair exchange of decryption keys for e-coins, or for other decryption keys.

4.1 Participants

Bank and arbiter. Our design requires the existence of two trusted entities: a *bank*, which provides secure resource accounting, and an *arbiter*, which ensures the fair exchange of e-cash for data. These nodes are trusted to be fair, but otherwise are not trusted with private information, and may operate separately.

The *bank* maintains each user's bank account, handling and validating deposits and withdrawals. The centrally-administered bank is also responsible for administering monetary policy, which we discuss in Section 7. For our application, a user's accumulated savings represents the amount of data uploaded in excess of that downloaded; it also bounds the maximum number of simultaneous buying and bartering transactions a user may undertake. Like the tracker, the bank cannot link activity between peers. The application of unlinkable currency makes BitTorrent's design—while not originally privacy-preserving—no *less* private by e-cash.

The *arbiter* protects the fair exchange of keys for e-coins by resolving aborted transactions in cases of node failure or intentional misbehavior. (We expect the prospect of future exchanges, especially in a system like BitTorrent that prizes high-bandwidth neighbors so highly, to be a strong incentive against the latter.) The arbiter seems well-suited for distribution: distributing it would require only the system's escrow key and a put/get database, similar to the design the email postage system DQE [48].

The introduction of these two centralized entities, required by our goals of secure accountability, may seem to contradict the fully decentralized nature of some peer-to-peer systems. But it fits our goal of enhancing the scalability of these systems with proper accounting. We note that previous work on incentives and accountability in P2P systems have also relied on centralized entities, whether by providing a central reputation service [7] or employing trusted agent(s) to punish misbehavior [33, 39]. BitTorrent also uses a trusted, centralized tracker to coordinate peer activity. These systems reinforce the view that the value of P2P design is in its scalability, not necessarily in its decentralized nature.

Users. Each user must establish an account with the bank before she can withdraw and deposit e-cash. Rather than provide new users with a starting balance, we look to social networks: new users are invited by friends with existing accounts, who transfer some e-cash from their own bank account to the invitee's account. This technique limits the utility of Sybil attacks [19]; we discuss bootstrapping new users further in Section 7.

Our protocols require users to deposit each earned coin before it can be re-spent, presenting a potential performance bottleneck. The computational requirements necessary for our cryptographic protocols may weigh heavily on both users and the bank; we describe practical approaches to lessening this burden in Section 6.

4.2 Interactions

Obtaining blocks. Suppose Alice requests a block from Bob. Bob encrypts the block using a randomly chosen key, and sends the ciphertext to Alice. Alice responds with an unendorsed e-coin and a contract describing the file she wants. Having spent bandwidth on transferring the ciphertext, both parties now have an incentive to perform an optimistic fair exchange of the decryption key for the e-coin's endorsement. If Alice tries to avoid paying Bob, then Bob can give his key to the arbiter and prove that it decrypts the ciphertext correctly, as specified by the contract. The arbiter would endorse the e-coin for Bob. If Bob fails to give Alice a proper key, then neither Alice nor the arbiter will endorse the e-coin.

Alice and Bob may also barter blocks when mutual coincidence exists, *e.g.*, when each has received the other's encrypted blocks. To begin, they exchange unendorsed e-coins, along with the coins' endorsements encrypted under escrow. They can now perform fair exchange for decryption keys indefinitely, with the escrowed coin as collateral. Details on both protocols are provided in Section 5.

These two protocols are designed to correspond with the unbalanced and balanced exchanges used in a system like BitTorrent. We envision their use as follows: instead of relying on altruism, nodes buy blocks at the outset, and later switch to bartering once they have acquired enough data to participate fully in the torrent. We describe in Section 6 how the performance benefits of bartering over buying provide sharing incentives. After completing, nodes may continue to sell blocks to earn credit for concurrent or future torrents.

Key to the suitability of our protocols for this use is how our design *decouples data transfer from accounting*: nodes download encrypted blocks and pay for decryption keys later. This allows us to accommodate the "optimistic" behavior inherent in BitTorrent's choking protocol, since a node may optimistically send neighbors encrypted blocks and reasonably expect to be paid for their keys later. A pair of nodes might delay reckoning debts some number of rounds in order to wait for reciprocation and bartering opportunities, or to improve performance by paying for multiple keys with higher-denomination coins. Section 6 examines this further.

5. BUYING AND BARTERING PROTOCOLS

In this section, we present two new cryptographic protocols. In Section 5.1, we create a new fair exchange protocol that lets Alice buy a block of a file from Bob. In Section 5.2, we create a new fair exchange protocol that lets Alice and Bob trade two blocks: we call this process bartering. In both protocols, when users are honest, the arbiter will never be involved. If something goes wrong, the aggrieved party will ask the arbiter to resolve the dispute. In the barter protocol, this process is straightforward; in the buy protocol, conflict-resolution is more involved and we describe it in Section 5.3.

We minimize the amount of work the arbiter performs so that a few malicious users cannot overload it. Our buy protocol improves on the e-coin fair exchange protocols in Camenisch *et al.* [9] and Asokan *et al.* [4] because the arbiter never has to download the entire block (nor the encryption of it). Instead, the arbiter uses the VerifyKey protocol in Section 5.3 to randomly test the correctness of the block. Our barter protocol also improves on the Asokan *et al.* [4] digital content exchange protocol, which requires some trusted party to verify that each block is correct and to sign the encryption of each block together with the commitment to the encryption key. Our protocol only assumes that both parties know the desired hash of each block; thus no trusted signer is required. The barter protocol assumes that Alice and Bob maintain a continuous relationship; as a result, Alice and Bob only have to create one e-coin each to initiate a relationship, and can reuse the e-coins indefinitely.

Our protocol requires a symmetric block cipher: we write $\text{Enc}_K(block)$ to denote encrypting *block* with key K; $\text{Dec}_K(ctext)$ means decrypting *ctext* using key K. We assume that a block is large enough to be divided into chunks and $\text{Enc}_K(block)$ encrypts each chunk separately. We also use verifiable escrow [10]: $\text{Escrow}_{Arbiter}(data, contract)$ this encrypts *data* under the public-key of the arbiter. The decryption key to the escrow is a combination of the arbiter's secret-key and the *contract*; this lets the arbiter ensure he decrypts the escrow only when the terms of the *contract* have been fulfilled. Anybody who knows the arbiter's public-key and the *contract* can verify that the escrow is valid. Finally, we write $\text{Commit}(data)$ to denote a commitment that can only be opened to *data*. In practice, these would be implemented as Pedersen commitments [41].

We use Merkle hash trees [38] to create short descriptions of a block (or ciphertext). We write $\text{MHash}(block)$ to denote a Merkle hash of *block*. A person who knows the entire file can publish $(chunk, proof, \text{MHash}(block))$ to prove that *chunk* is in (or is not in) *block*; *proof* is short, efficiently calculated, and includes the position of the chunk in the block. (We require a collision-resistant hash function h.)

5.1 How to buy a file

We present our protocol that lets Alice buy a file block from Bob. Before the start of the protocol, Alice acquires $bhash = \text{MHash}(block)$ from a trusted authority (*i.e.* tracker). Alice and Bob will agree on a *timeout* by when Bob must provide Alice with the block. The protocol works as follows:

1. Bob chooses a random key K and sends $ctext = \text{Enc}_K(block)$ to Alice.
2. Alice constructs an endorsed e-coin $(coin', endorsement)$. Alice calculates $chash = \text{MHash}(ctext)$, chooses a random value r and calculates the exchange ID $v = h(r)$. Alice sets $contract = (bhash, chash, timeout, coin', v)$. She escrows the *endorsement* under the arbiter's public-key: $escrow = \text{Escrow}_{Arbiter}(endorsement, contract)$. Alice sends Bob $(coin', contract, escrow)$.
3. Bob verifies that $(coin', contract, escrow)$ is formed correctly. If he is satisfied, he establishes a secure connection to Alice using standard techniques and sends the key K. Otherwise, Bob terminates.
4. If Alice receives a K that lets her decrypt *ctext* correctly before *timeout*, she responds with *endorsement*. Otherwise, Alice waits until *timeout* and then calls AliceResolve(r) on the arbiter, as in Algorithm 5.1.
5. If Bob does not receive a correct *endorsement* before *timeout*, he calls BobResolve$(K, escrow, contract)$ on the arbiter, as in Algorithm 5.2.

We sketch why our protocol ensures a fair exchange. Suppose Alice wants to avoid paying. If Bob calls BobResolve before *timeout*, he is guaranteed to be paid, as long as the unendorsed e-coin is valid (recall that Alice cannot abort the protocol before *timeout*). If the unendorsed e-coin is invalid (either badly formed $coin'$ or incorrect *contract*), Bob would not accept it and terminate in step 4 without giving Alice the key K.

Suppose Bob wants to avoid giving Alice a correct key. If he calls BobResolve after *timeout* he will not get paid. If he calls BobResolve before *timeout*, due to the *contract* associated with the *escrow*, he can only get paid if he deposits the correct key K. Alice can then retrieve K at her convenience.

5.2 How to barter

We present a new protocol that lets Alice and Bob perform a fair exchange of two files. The exchange proceeds in two phases. First, Alice and Bob give each other an unendorsed e-coin and an escrow of the endorsement. This establishes a collateral that an aggrieved party can collect if something goes wrong. In the second phase, Alice and Bob perform a fair exchange of the file. If the exchange fails, the wronged party can ask the arbiter to endorse the e-coin. As long as Alice and Bob are honest, they can continue in a bartering relationship indefinitely using the same e-coin as collateral.

Suppose Alice has $block_A$ and she wants $block_B$ which is owned by Bob. They both get $hash_A = \text{MHash}(block_A)$,

$hash_B = \text{MHash}(block_B)$ from a trusted authority (*i.e.* the tracker). They perform the exchange as follows:

1. Alice chooses a new signing key $(sk_{\mathcal{A}}, pk_{\mathcal{A}})$ and gives $pk_{\mathcal{A}}$ to Bob. Bob does the same, responding with $pk_{\mathcal{B}}$.
2. Alice creates an endorsed e-coin $(coin'_A, endorsement)$ and calculates $escrow_A = \text{Escrow}_{Arbiter}(endorsement, contract)$, where the *contract* states that the arbiter can endorse $coin'$ for anyone who presents some $contract'$ that is (1) signed by $pk_{\mathcal{A}}$ and (2) whose terms are fulfilled. Alice gives $(coin'_A, escrow_A)$ to Bob, who performs the corresponding operation and gives Alice $(coin'_B, escrow_B)$.
3. Alice calculates a ciphertext $ctext_A$ and a commitment to the decryption key $K'_A = \text{Commit}(K_A)$. Alice gives $(ctext_A, K'_A)$ to Bob. Bob similarly computes $(ctext_B, K'_B)$ and gives it to Alice.
4. Alice and Bob both compute $contract' = (pk_{Arbiter}, pk_{\mathcal{A}}, K'_A, \text{MHash}(ctext_A), hash_A, pk_{\mathcal{B}}, K'_B, \text{MHash}(ctext_B), hash_B)$. This contract states that to collect collateral, one of two conditions must be met: (1) the owner of $pk_{\mathcal{B}}$ can prove that the opening of K'_A does not decrypt a ciphertext corresponding to $\text{MHash}(ctext_A)$ to a plaintext corresponding to $hash_A$, or (2) the owner of $pk_{\mathcal{A}}$ can prove that the opening of K'_B does not decrypt a ciphertext corresponding to $\text{MHash}(ctext_B)$ to a plaintext corresponding to $hash_B$. This can be proved using standard techniques from Merkle hashes.
5. Alice gives Bob her signature on $contract'$ and Bob gives Alice his signature on $contract'$.
6. Alice and Bob execute a fair exchange protocol where Alice gets K_B (the opening of K'_B) and Bob gets K_A (the opening of K'_A). This can be done using Asokan *et al.* fair exchange [4].
7. If K_B does not decrypt $ctext_B$ correctly, Alice goes to the arbiter with the signed $contract'$, $escrow_B$, K_B, a proof showing that $ctext_B$ did not decrypt correctly, and a proof that she knows the secret key corresponding to $pk_{\mathcal{A}}$. The arbiter would give Alice the endorsement to Bob's e-coin and his signature on the e-coin. Alice can bring the endorsed e-coin to the bank and deposit it in her account. Bob would do the same if K_A is incorrect.

Note: Showing that $ctext_B$ does not decrypt correctly can be done efficiently. Alice gives the arbiter the signed $contract'$, K_B, and a *chunk* that does not decrypt correctly. The arbiter can check that K_B is the promised opening of K'_B. Then the arbiter can test if (1) *chunk* is in $ctext_B$, (2) $\text{MHash}(ctext_B)$ is in $chash_B$ and (3) $\text{Dec}_{K_B}(chunk)$ is *not* in $hash_B$.

Steps 1 and 2 of the protocol only have to be done once to establish a bartering relationship between Alice and Bob. Subsequently, Alice and Bob can perform steps 3–7 to exchange a block. The bartering protocol has more efficient conflict resolution. If Bob cheats Alice, Alice can show the arbiter which chunk decrypted incorrectly. As a result, (1) conflict resolution is more efficient and (2) a cheating Bob is caught with *overwhelming probability*. Finally, we note that, bartering two files is more efficient for the users than executing two purchase protocols; this is because the Asokan *et al* [4] fair exchange only has to be performed once instead of twice (per block). This is true even if Alice and Bob decide to preserve anonymity by using new signing keys and e-coins each time they exchange files.

Algorithm 5.1: AliceResolve, run by the arbiter

Input: Exchange ID r (ensures only Alice can resolve)
$v \leftarrow h(r)$;
if $< v, K > \in DB$ **then**
  send K to Alice.
end

Algorithm 5.2: BobResolve, run by the arbiter

Input: key K, *escrow*, and contract { *bhash*, *chash*, *timeout*, $coin'$, v }) (all sent by Bob)
if $currentTime < timeout$ **then**
  $endorsement \leftarrow \text{Decrypt}(escrow)$;
  if *endorsement not valid for coin'* **then**
    return error.
  end
  Run VerifyKey with Bob for K, using $(bhash, chash)$ from the contract. // ensures Alice sent them
  if K *verifies* **then**
    add $< v, K >$ to DB.
    send *endorsement* to Bob.
  else
    return error.
  end
end

The main security challenge is to ensure that Alice cannot deposit the e-coin she put up for collateral (and that Bob cannot deposit his collateral e-coin). We have to make an assumption about the endorsed e-cash deposit protocol: the bank can verify the contract associated with an endorsed e-coin. Specifically, the bank will have to verify that the arbiter signed the e-coin (the arbiter's signing key is included in the contract, so the bank does not have to know the arbiter's identity in advance). As a result, to deposit the e-coin under $contract'$, Alice has to get the arbiter's signature. Alice cannot enforce clause (1) of the contract because she does not know Bob's secret key. Alice can enforce clause (2) only if Bob cheats, in which case she is entitled to get her e-coin back. The only other option Alice has is to deposit her e-coin under a new contract. If Alice does this, and Bob later deposits the endorsed e-coin under the old contract, the bank will see that Alice double-spent an e-coin. Due to the construction of endorsed e-cash, if the same e-coin is deposited under two different contracts, the bank can trace the owner of the e-coin. Thus Alice cannot deposit her own collateral e-coin unless Bob cheats. The same argument shows that Bob also cannot deposit his own collateral e-coin unless Alice cheats.

5.3 How to resolve disputes

We give a protocol that lets a seller prove to the arbiter that he provided the buyer with the correct ciphertext and decryption key.

Recall that when Bob calls BobResolve in Algorithm 5.2, he has to prove to the arbiter that he has provided it with the correct key. Specifically, he has to show that K decrypts $ctext = c_0||c_1||\ldots||c_n$ to $block = b_0||b_1||\ldots||b_n$, where *ctext* is in *chash* and *block* is in *bhash*. Bob and the arbiter execute VerifyKey, as shown in Algorithm 5.3.

Algorithm VerifyKey will detect if chunk c_i does not decrypt correctly. We call such a chunk *corrupted*. If Bob cor-

Algorithm 5.3: VerifyKey

Arbiter's Input: Two Merkle hashes *bhash* and *chash*, key K

Bob's Input: Ciphertext $c_0||\ldots||c_n$), key K

`Step 1: Arbiter's challenge`

The arbiter sends Bob a set of indices I.

`Step 2: Bob's response`

Bob replies with $(c_i, cproof_i, bproof_i)$ for every $i \in I$, where $cproof_i$ proves that c_i is the Merkle tree corresponding to *chash* and $bproof_i$ proves that $b_i = \mathrm{Dec}_K(c_i)$ is in the Merkle tree corresponding to *bhash*.

`Step 3: Verification`

The arbiter *accepts* the key if Bob responds with valid $(c_i, cproof_i, bproof_i)$ for every $i \in I$, and *rejects* otherwise.

rupts an nth fraction of the chunks, and the arbiter verifies k chunks, then the arbiter will catch Bob with probability $1-(1-n)^k$. Suppose Bob corrupts 10% of the chunks. To catch Bob with 90% probability, the arbiter needs to check 22 chunks; to catch Bob with 80% probability, the arbiter needs to check 16 chunks. This approach might not deter a *malicious* Bob who just wants to perform a denial of service attack. However, a *selfish* Bob who wants to try to get paid for a bad file block would be deterred. This is good enough for our purposes.

6. EVALUATION

The requirements of anonymity and security that we have placed on our currency and associated protocols impose computational and communication overheads. This section describes these overheads, and explains how they can be mitigated by careful system design.

Cost of privacy. Our design assumes that users value the privacy of their interactions in peer-to-peer systems, and would thus prefer not to use an accounting system that might record an audit trail of all user transactions. We assessed the cost of this privacy by comparing our protocol with a simpler contract fair exchange, as in [4]. There, the contract specifies that the buyer pays an e-coin to the seller for a block, indicated by hash value. Provided a contract, the bank validates it and transfers the noted amount to the seller's account. This scheme provides less privacy than BitTorrent, granting the bank complete knowledge of all transactions.

We compared only the buyer-seller part of our protocol to this simple contract fair exchange scheme. We conducted tests on a Pentium M 1.6 GHz CPU with 1 GB of RAM, using our Java implementation of our protocols. Our protocol requires about 12.5 KB of data exchange (per block, which can be as big as 2 MB in BitTorrent protocol), and takes about 2.5 seconds for the buyer and 1.5 seconds for the seller. The contract fair exchange protocol requires about 4 KB of data exchange, and takes about 1 second for the buyer and less than 0.5 seconds for the seller. Thus, the cost of our system is roughly two times more than in simple centralized accounting system.

We also measured operations involving the bank. The withdrawal of a wallet of e-coins took 1.2 seconds and 1.7 KB of data, independent of the withdrawal amount (number of coins). Unfortunately, to guarantee our currency's unlinkability, users cannot deposit multiple e-coins in one operation, and so for each e-coin, the bank must perform the verification equivalent to the seller's job in the buy protocol.

Scaling by bartering. Our currency system requires the bank to perform expensive verification for each e-coin presented for deposit, performing roughly as much work as the seller in the buy protocol. This high transactional overhead implies that a distributed, load-balanced bank service would be required to meet the demands of a large system. A mixed hardware-software approach [16] also offers a way to accelerate each bank node's performance.

We should also consider reducing the number of buy-sell exchanges (and thus deposits) performed in the system. One may begin by selecting a large block size, lowering the number of transactions required, to the detriment of fine-grained accounting. This provides a guideline for uses in other applications: using e-cash to account for a very large number of small transactions may not yet be practical.

Our barter protocol, described in Section 5.2, sidesteps this potential bottle-neck by avoiding the bank altogether. Bartering requires expensive computation only for the first exchange, when both parties create escrow coins.Successive bartering between the same two parties involve only key exchanges, not heavy computation: the bank becomes involved only in the case of a dispute, when one of the parties may deposit the other's escrow coin. As discussed in Section 2, BitTorrent's choking mechanism and clustering behavior suggests bartering would be fairly common, and made even more so with a block-level tit-for-tat policy [6] or a choking policy designed for fairness [46].

Of course, bartering may only be used when a pair of participants have mutually desirable content. We evaluate the likelihood of these exchanges using the discrete event simulator from [6]. We attempted to replicate the heterogeneous "flash crowd" scenario from their published results, using 1000 nodes split evenly between cable (6 Mbps down; 3 Mbps up), DSL (1.5 Mbps down; 400 Kbps up) and slow DSL (784 Kbps down; 128 Kbps up). The participating nodes exchanged a 400-block file with the help of a 6Mbps seed node.

The simulator provides a complete log of node interactions; we analyzed logs generated from both the default choking policy and the block-level tit-for-tat (TFT) policy. For each piece received, we searched through nearby events, looking for a reciprocating piece in the opposite direction. If blocks were exchanged in both directions during a short window, we assume that the peers involved could have bartered for the blocks in question.

We present the results of our simulations in Figure 1, charting the fraction of blocks found eligible for barter (out of a total of 400,000 possible transactions) as we increase the barter window length. Each line represents a simulation run using a different choking policy. We find that with a 30-second barter window (three rounds, the length of an optimistic unchoke), the block-level TFT and default policies reveal roughly equivalent opportunities for bartering, at a little more than half of all transactions. With longer barter windows, the fairer block-level TFT policy produces more opportunities, but not as many as with the quick bandwidth estimation (QBE) scheme [6]. The authors designed QBE to obviate the need for optimistic unchoking (a major source of unfairness), by simulating the effect of instantaneous, accu-

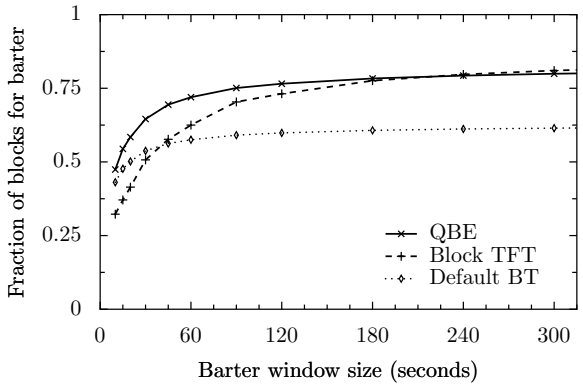

Figure 1: Bartering opportunities for different barter windows. Choking policies designed for fairness and clustering produce more opportunities for bartering.

rate bandwidth probes between nodes. This allows nodes to immediately begin unchoking their fastest neighbors, without need for the trial-and-error method of tit-for-tat.

These simulation results indicate that more than half of peer transactions between BitTorrent nodes involve reciprocation soon after. Finding more bartering opportunities requires modified choking policies designed for fairness and clustering; interestingly, the currency overhead aligns greater fairness, through barter, with reduced computational burden. However, achieving these results in practice may be more difficult: we do not simulate node churn, failure, nor do we attempt to integrate currency budgets or debt-reckoning at certain intervals into the choking policy. We leave this for future work.

Batched transactions. Bank deposits may also be reduced by grouping transactions over time: the bank may issue different coin denominations (*i.e.*, \$1, \$2, \$4, etc.), which can be used to pay for multiple blocks at once. The bank must publish different keys corresponding to each coin denomination, so that they may be withdrawn and recognized by all users. A buyer could then negotiate a contract to pay for n decryption keys with $log(n)$ coins. We expect peers may use larger coins once a long-standing relationship has been established (so that in case of a dispute, not many resources would have been wasted), thereby decreasing load on themselves and the bank.

We note that these techniques may apply to other currency systems as well, regardless of the underlying implementation. For example, Karma [47] builds a currency atop a distributed hash table (DHT); as a result, its karma-for-file exchange protocol requires many DHT lookups. Here, the overhead is defined by network latency, rather than computation, but this overhead may nevertheless be similarly reduced by bartering and batching.

7. ECONOMIC ISSUES

In a monetized P2P network selfish peers serve as agents in a virtual economy. Peers that behave correctly not only earn money, but add value to the network. As peers join, the economy grows, requiring the creation and distribution of currency. The bank must implement a monetary policy and peers must navigate the marketplace. In this section we consider some simple solutions to these problems.

Entering the network. Users in a monetized file sharing network make money by sharing the files they acquire. This presents a bootstrapping problem: new users must obtain the resources (either e-cash or files) required to obtain files.

If the bank simply gives new users e-cash, the network becomes highly vulnerable to Sybil attacks [19]. Users will join repeatedly and e-coins will become worthless. A more reasonable option is to give away blocks from a randomly selected file. Unfortunately this places significant demand on the bank's bandwidth, and requires that the bank obtain a steady stream of desirable (and legal) content.

To avoid incentivizing Sybil attacks, we provide new users with neither coins nor files. Instead we assume an existing social network capable of distributing files and e-coins. It is in a P2P network's interest to admit productive users. We expect existing users to give (or lend) e-coins and files to trusted friends.

If a potential user cannot obtain resources through friends, they can also obtain e-coins using some real world resource. The bank may either sell e-coins for real money, or facilitate an auction between new and existing users. Alternatively, users can be given coins when they perform a bandwidth intensive task (for example, users look up webpages and compute hashes, which the bank then crosschecks). If the payoff of this task is small compared to the payoff of sharing files, users will not waste bandwidth to join multiple times.

Creation of money. In any monetized P2P network, there is a time and cost associated with each transaction. The bank must provide enough e-cash to satisfy the network's demand for simultaneous transactions. If the bank produces too few e-coins, the network cannot operate near full capacity, and if the bank produces too many e-coins inflation potentially leads to a monetary crash [30].

One proposed heuristic for currency creation is to grow the number of e-coins linearly with the number of users [47]. This heuristic ignores the fact that certain users add more value to a network than others. Furthermore, it does not address how new e-coins should be distributed. Simply paying users interest on their bank balances is problematic since it allows wealthy users to continually earn money without participating.

A simple, yet robust approach to wealth distribution is for the bank itself to participate in the network. The bank can inject new e-coins when making purchases, or destroy existing e-coins after making sales. To determine when currency should be created or destroyed the bank can monitor the rate at which files are bought and sold, as well as the rate at which e-coins are withdrawn and deposited. If currency becomes sparse, purchase requests will slow. If currency is overly abundant, purchase requests will increase but finding sellers will become difficult. In either case withdrawal and deposit rates with slow.

A more analytic approach to currency regulation is pursued in [30]. Using a restricted economic model, the authors demonstrate that bank balances (*e.g.* knowledge of wealth distribution) can predict how much currency can be added to a monetized P2P network without resulting in a crash. The model also allows the authors to bound the impact of altruistic behavior and money hoarding. Though their approach is promising, their results assume that all users have equal file-sharing resources (*i.e.* bandwidth, files to share). The authors believe that more realistic assumptions do not fundamentally change their results.

Variable vs. fixed pricing. In our protocol, the price of a BitTorrent block is fixed at one e-coin. This minimizes the overhead associated with purchasing files, but requires the value of an e-coin to remain relatively constant; it also assumes that all content is priced the same per block. A protocol with variable prices permits sellers to correct for inflation and deflation. It also allows buyers to pay more for faster delivery of high-priority blocks (this would be useful for streaming). To enable variable pricing, the bank can produce multiple denominations of e-coins. If multiple coins are involved in a transaction, however, the overhead associated with depositing these coins grows linearly. Furthermore, client behavior becomes much more complex, since prices must be intelligently negotiated.

Our approach also provides incentives for the BitTorrent's "rarest first" heuristic. Even with fixed pricing, the rare blocks are more likely to be sold. Participants have an incentive to buy rare blocks first, making them more common, and helping to solve the problem of blockage at the end.

8. OTHER APPLICATIONS

We believe e-cash can be securely and anonymously applied to many other distributed peer-to-peer systems. This section gives some insights on how to make use of e-cash to incentivize such applications. Even though the technical details are missing, the ideas are worth investigating.

Distributed lookup. In BitTorrent, users contact a centralized tracker for a list of neighbors who are likely to have the file. We want to remove any central trust points (except possibly the bank). We need the system to be efficient: lookup queries should not flood the whole network. We want to introduce incentives to *encourage* peers to help other users find files and to *discourage* unnecessary lookup queries.

One approach is to pay each node along the query path; the interaction between search depth and query price has been studied in random-tree networks [31]. In a fully incentivized DHT, payments would insure an efficient lookup structure and honest responses to queries. Suppose Alice wants to find some file (or a peer responsible for tracking the file). She asks Bob, the closest peer she knows to the file, for the next hop. Alice pays for this information using endorsed e-cash only if the next hop node is at least half-way closer to the file than Bob (which can be checked using hashes), as specified in the contract. Then Alice would contact the next node and repeat the process. Alice pays all peers that help her find the file, encouraging peers to cooperate, and discouraging fake lookups.

Distributed storage. A distributed storage system allows a user to backup her data on another peer's machine. Suppose Alice wants to backup her data, and this will be done on Bob's machine. If Alice pays Bob upfront, then Bob has no incentive to store the data, he already got the money. However, if Alice pays him upon retrieval, then if she decides she no longer needs the data, she would never pay and Bob would have wasted his resources storing her data.

We propose that Alice pays Bob upfront, but Bob gives her a *warranty* check in return (via a fair exchange protocol) that contains a Merkle hash of the data (for the arbiter to easily verify without downloading the whole data), an unendorsed e-coin, and an escrow of the endorsement. If Bob ever loses or corrupts the data, the arbiter would decrypt the escrow and pay Alice for her damage. The bank can ensure that Bob always maintains a balance large enough to cover his liability. When Alice gets her data, the check is invalidated.

Distributed computation. In current distributed computing projects, like Rosetta@Home [44], people voluntarily donate their excess CPU cycles to perform computation-heavy tasks. We can transform this one-way system into a mutually beneficial computing cluster. Users can accept outside jobs when their CPU is not fully loaded, and pay other users to perform some computations when they need more resources.

Suppose Alice wants some computation-heavy task to be done (although I/O-heavy tasks may also be considered). We propose that using e-cash can provide an incentive for other uses to not only contribute in this computation but also perform it correctly. For some problems, such as graph coloring (in NP), verifying the correctness of the answer is easy, and so can be a part of the contract. In optimization problems, the contract may specify that the best answer within a deadline will be paid (the most), and again the verification is easy. Unfortunately, this is not true for some other types of problems.

Onion routing. In onion routing (*i.e.* anonymous remailing) [11, 23, 18], participants route their messages through randomly chosen intermediate peers to disguise the origin and destination of messages. Each router only knows about the next hop, and so the real sender and receiver is hidden. Selfish peers might want to use the system to send and receive messages, but refuse to route other peers' packets. As an incentive, a router gets paid only if it passes a message to the next router in chain, by using threshold endorsed e-cash as explained by Camenisch *et al.* [9]. For a router to get paid, it needs to collaborate with both the previous and the next hops.

9. RELATED WORK

This paper is not the first to propose currency-based approaches to resource allocation in distributed systems. Previous work has seen currencies used to allocate resources in grids [27] and sensor networks [14], pay for distributed storage [26], and optimize queries in distributed databases [45]. Game-theoretic analyses of peer-to-peer currency systems and associated equilibria are detailed in [24, 22, 30]. Mojo Nation [49] used a currency for incentives that was not provably secure or privacy-preserving. Karma [47] describes a currency built atop a DHT, which places the responsibility of maintaining each node's bank account and processing transactions with a "bank-set" of randomly-chosen nodes from the DHT. This decentralization makes it difficult to effectively manage the currency (see Section 7), requiring $O(N^2)$ messages. Though these systems provide useful examples of currency-based design, they do not address issues of privacy and security that we consider essential.

Local and centralized reputation systems have also been used to provide incentives in peer-to-peer systems [50, 37]. Some even quantify local reputation values in terms of non-fungible credit [40] or currency [2], extending the barter economy by identifying chains of indebted nodes that can be used for transitive exchanges. However, reputation schemes are less effective when discredited pseudonyms can be easily shed, limiting their applicability in peer-to-peer and privacy-enhancing systems [17, 36]. Friedman and Resnick [21] show

that in this environment, participants must distrust new users until they have earned enough reputation, leading to inefficiency. Our currency can be thought of as providing fungible reputation independent of identity-based observations, which in our view seems better suited than reputation alone for systems with high churn, or those that use pseudonymous or anonymous identities.

10. CONCLUSION

We have described a practical application of e-cash to the problem of free-loading in peer-to-peer systems. Our design emphasizes provable security without sacrificing anonymity, and fungible, rather than pair-wise, credit for contributions. We have shown that the resulting system remains scalable due to careful protocol and system design. We look forward to applying these ideas to other peer-to-peer systems.

11. ACKNOWLEDGEMENTS

This work is supported by NSF CyberTrust grant 0627553.

12. REFERENCES

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] K. G. Anagnostakis and M. B. Greenwald. Exchange-based incentive mechanisms for peer-to-peer file sharing. In *ICDCS '04*, 2004.

[3] N. Andrade, M. Mowbray, A. Lima, G. Wagner, and M. Ripeanu. Influences on cooperation in bittorrent communities. In *P2PECON '05*, 2005.

[4] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):591–610, Apr. 2000.

[5] F. Bao, R. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line ttp. In *IEEE Symposium on Security and Privacy*, May 1998.

[6] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and improving a bittorrent network's performance mechanisms. In *Proc. IEEE INFOCOM*, Mar. 2006.

[7] A. Blanc, Y.-K. Liu, and A. Vahdat. Designing incentives for peer-to-peer routing. In *Proc. IEEE INFOCOM*, Mar. 2005.

[8] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO 2002*, Aug. 2002.

[9] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *IEEE Symposium on Security and Privacy*, 2007.

[10] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO 2003*.

[11] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[12] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology — CRYPTO '82*, pages 199–203. Plenum Press, 1983.

[13] D. Chaum. Blind signature systems. In *Advances in Cryptology — CRYPTO '83*, page 153. Plenum Press, 1984.

[14] B. N. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. C. Parkes, J. Shneidman, A. C. Snoeren, and A. Vahdat. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proc. of the 2nd IEEE Workshop on Embedded Networked Sensors*, 2005.

[15] B. Cohen. Incentives build robustness in bittorrent. In *Proc. 2nd IPTPS*, Berkeley, CA, Feb. 2003.

[16] A. Daly and W. Marnane. Efficient architectures for implementing montgomery modular multiplication and rsa modular exponentiation on reconfigurable logic. In *FPGA '02*.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Reputation in p2p anonymity systems. In *P2PECON '03*, June 2003.

[18] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *Proc. of the 13th USENIX Security Symposium*, August 2004.

[19] J. Douceur. The sybil attack. In *Proc. 1st IPTPS*, Mar. 2002.

[20] B. Fan, D.-M. Chiu, and J. C. S. Lui. The delicate tradeoffs in bittorrent-like file sharing protocol design. In *ICNP '06*, 2006.

[21] E. Friedman and P. Resnick. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy*, 10(2):173–199, 2001.

[22] E. J. Friedman, J. Y. Halpern, and I. Kash. Efficiency and nash equilibria in a scrip system for p2p networks. In *Proc. of the 7th ACM conference on Electronic commerce*, 2006.

[23] D. M. Goldschlag, M. G. Reed, and P. F. Syverson. Onion routing for anonymous and private internet connections. *Communications of the ACM*, 42(2):84–88, February 1999.

[24] P. Golle, K. Leyton-Brown, and I. Mironov. Incentives for sharing in peer-to-peer networks. In *Proc. of the 3rd ACM Conference on Electronic Commerce*, 2001.

[25] GreedyTorrent. `http://www.greedytorrent.com/`.

[26] J. Ioannidis, S. Ioannidis, A. D. Keromytis, and V. Prevelakis. Fileteller: Paying and getting paid for file storage. In *Financial Cryptography, 6th International Conference*, 2002.

[27] D. Irwin, J. Chase, L. Grit, and A. Yumerefendi. Self-recharging virtual currency. In *P2PECON '05*, Aug. 2005.

[28] M. Jakobsson. Ripping coins for a fair exchange. In *EUROCRYPT '95*, 1995.

[29] S. Jun and M. Ahamad. Incentives in bittorrent induce free riding. In *P2PECON '05*, Aug. 2005.

[30] I. Kash, E. J. Friedman, and J. Y. Halpern. Efficiency and nash equilibria in a scrip system for p2p networks. In *Proc. of the 8th ACM conference on Electronic commerce*, 2007.

[31] J. Kleinberg and P. Raghavan. Query incentive networks. In *FOCS '05: Proc. of the 46th Annual IEEE Symposium on Foundations of Computer Science*, 2005.

[32] A. Legout, N. Liogkas, E. Kohler, and L. Zhang. Clustering and sharing incentives in bittorrent systems. In *Proc. of ACM SIGMETRICS'07*, June 2007.

[33] H. Li, A. Clement, E. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin. BAR gossip. In *Proc. of the 7th OSDI*, Nov. 2006.

[34] N. Liogkas, R. Nelson, E. Kohler, and L. Zhang. Exploiting bittorrent for fun (but not profit). In *Proc. 5th IPTPS*, 2006.

[35] T. Locher, P. Moor, S. Schmid, and R. Wattenhofer. Free Riding in BitTorrent is Cheap. In *Proc. 5th Workshop on Hot Topics in Networking (HotNets-V)*, Nov. 2006.

[36] S. Marti and H. Garcia-Molina. Identity crisis: Anonymity vs. reputation in p2p systems. In *P2P '03*, 2003.

[37] S. Marti and H. Garcia-Molina. Taxonomy of trust: Categorizing p2p reputation systems. *Computer Networks*, 50(4):472–484, 2006.

[38] R. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO '87*, pages 269–278, 1987.

[39] N. Michalakis, R. Soulé, and R. Grimm. Ensuring content integrity for untrusted peer-to-peer content distribution networks. In *Proc. 4th USENIX/ACM NSDI*, Apr. 2007.

[40] A. Nandi, T.-W. J. Ngan, A. Singh, P. Druschel, and D. S. Wallach. Scrivener: Providing incentives in cooperative content distribution systems. In *Middleware 2005*, Nov. 2005.

[41] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91*.

[42] M. Piatek, T. Isdal, T. Anderson, A. Krishnamurthy, and A. Venkataramani. Do incentives build robustness in bittorrent? In *Proc. 4th USENIX/ACM NSDI*, Apr. 2007.

[43] M. Reiter, X. Want, and M. Wright. Building reliable mix networks with fair exchange. In *Applied Cryptography and Network Security: Third International Conference*, 2005.

[44] Rosetta@Home. `http://boinc.bakerlab.org/rosetta/`.

[45] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A wide-area distributed database system. *VLDB Journal: Very Large Data Bases*, 5(1):48–63, 1996.

[46] R. Thommes and M. Coates. Bittorrent fairness: analysis and improvements. In *WITSP '05*, December 2005.

[47] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. Karma: A secure economic framework for p2p resource sharing. In *P2PECON '03*, 2003.

[48] M. Walfish, J. Zamfirescu, H. Balakrishnan, D. Karger, and S. Shenker. Distributed quota enforcement for spam control. In *Proc. 3rd USENIX/ACM NSDI*, May 2006.

[49] B. Wilcox-O'Hearn. Experiences deploying a large-scale emergent network. In *Proc. 1st IPTPS*, Mar. 2002.

[50] B. Zhu and S. Jajodia. Building trust in peer-to-peer systems: a review. *International Journal of Security and Networks*, 1(1/2):103–112, 2006.