KARMA : A Secure Economic Framework for Peer-to-Peer Resource Sharing

Vivek Vishnumurthy, Sangeeth Chandrakumar and Emin Gün Sirer
Department of Computer Science, Cornell University, Ithaca, NY 14853

Abstract

Peer-to-peer systems are typically designed around the assumption that all peers will willingly contribute resources to a global pool. They thus suffer from freeloaders, that is, participants who consume many more resources than they contribute. In this paper, we propose a general economic framework for avoiding freeloaders in peer-to-peer systems. Our system works by keeping track of the resource consumption and resource contribution of each participant. The overall standing of each participant in the system is represented by a single scalar value, called their karma. *A set of nodes, called a* bank-set, *keeps track of each node's karma, increasing it as resources are contributed, and decreasing it as they are consumed. Our framework is resistant to malicious attempts by the resource provider, consumer, and a fraction of the members of the bank set. We illustrate the application of this framework to a peer-to-peer filesharing application.*

1 Introduction

Recent years have seen the introduction of peer-to-peer systems, whose design relies centrally on exchange of resources between peers. The utility of such systems is proportional to the aggregate amount of resources that the peers are willing to pool together. While many peer-to-peer systems have implicitly assumed that peers will altruistically contribute resources to the global pool and assist others, recent empirical studies have shown that a large fraction of the participants engage in freeloading: 20 to 40% of Napster and almost 70% of Gnutella peers share little or no files [1, 2]. This is not surprising, since there is little concrete incentive for peers to contribute resources.

This paper outlines the design of a peer-to-peer system that incentivizes participating nodes to contribute resources to a global pool, and illustrates how this economic framework can be used in a filesharing system. Our system, called KARMA, is economic, that is, it works by keeping track of the resource purchasing capability of each peer. A *resource* in KARMA can be anything exchanged between two peers, such as files, messages, or the result of a computation. A single scalar value, called *karma*, captures the amount of resources that a peer has contributed and consumed, and represents the user's standing within the global system. Groups of nodes, called *bank-sets*, keep track of the karma belonging to the users. A user is initially awarded a seed amount of karma when he joins the system. The karma balance is adjusted upwards whenever the user contributes resources, and downwards whenever he consumes resources. A transaction is not allowed to proceed if the resource-consumer has less karma than it takes to make the payment for the resources involved. Thus, participants are ultimately forced to achieve parity between the resources they contribute and those they consume.

The economic framework presented in this paper provides the properties of non-repudiation, certification, and atomicity. That is, KARMA protects against malicious providers that promise a resource but do not deliver it completely, against malicious consumers that receive a resource but claim that they did not, and against transient states of the system where participants can observe intermediate states in the process of transferring karma from one account to the other. KARMA uses an atomic transaction scheme that provides the resource consumer with the key to decrypt the resource simultaneously as it provides the provider with a certificate of receipt. Also, KARMA limits the effects of large-scale inflation and deflation by applying periodic corrections to the outstanding karma in the system.

2 Overview

In this section, we describe the basic operation of KARMA in the context of a file-sharing application. While filesharing is useful as a tangible example, we note that the basic transfer protocols in KARMA can be used equally well with other kinds of resources, such as file blocks instead of whole files, messages in a publish-subscribe system, or the results of a computation in a grid computing system. KARMA maintains its internal state in p2p fashion, distributed across the participants, and employs the secure routing primitive [4] for reliable delivery of messages. While our prototype implementation is layered on top of Pastry [3], our design can be extended to work on top of any Distributed Hash Table(DHT) with the corresponding secure routing property.

The design of our system is guided by three fundamental properties stemming from the peer-to-peer domain. First,

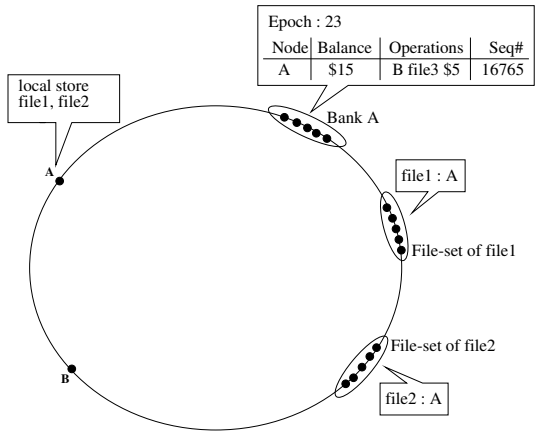

Figure 1: Overview of system state: $Bank_A$ stores A's account: A has a balance of \$15 karma, and has recently paid B \$5 for file $file3$; A's sequence number is 16765, and the current epoch number is 23. File-sets of $file1$ and $file2$ store the list of nodes that store the respective files.

since KARMA is designed to complement peer-to-peer systems, the system itself needs to be completely distributed and require no centralized functionality or trust. Second, since there are no failure-proof components in a loosely-organized network of peers, account data needs to be replicated, possibly extensively, to insure against loss and tampering. Third, since a transaction system needs to perform well, the coordination among the replicas must be kept to a minimum. Karma's design strives to achieve these goals.

KARMA relies principally on replication to deter nodes that might try to subvert the protocol. It assumes that there are at least k nodes in the system at all times, and uses protocols to ensure that the system will operate correctly unless a substantial fraction of these nodes are malicious.

2.1 Maintenance of bank-set information

The bank-set of a node maintains the karma balance of the node, and updates the account after each upload and download. We define the bank-set *Bank*$_A$ of a node A as the leaf-set of the node closest in the nodeId-space to $HASH(nodeId(A))$; all nodes in the bank-set are equally responsible for maintaining the required information. It is critical that this assignment of nodes to a bank-set be secure against targeted attacks to take over the bank-set. While any other mapping scheme can be used, this particular approach allows us to layer our implementation on top of an existing DHT like Pastry [3], using the secure routing primitive suggested in [4]. We use k to denote the size of the leaf-set, and *bank-root* to denote $HASH(nodeId(A))$.

Each member of *Bank*$_A$ stores the amount of karma in A's account, and information regarding all recent payments A has made to other nodes for files. This extra information acts as proof of A's payment, and comes into play when the other party in the transaction has not sent A the file for which the payment was made. The bank-set corresponding to each node also stores (i) the last used $sequence-number$, which is part of the message sent by a node authorizing its bank-set to transfer karma from its account to the account of some other member, and (ii) the current epoch number. Each epoch spans a fixed length of time, typically several months, and at the end of each epoch, currency adjustments are made so that the per-capita karma in the system is constant, thereby eliminating the effects of inflation and deflation (see Section 2.3). The sequence number used by a node is incremented after each transaction, and eliminates the possibility of replay attacks. Figure 1 shows a snapshot of KARMA.

2.2 Maintenance of file information

For each file in the system,the filesharing application uses the DHT to store a list of nodes which have a copy of that particular file under the $fileId$ of the file. The fileId is derived by taking the MD5 hash of the file name, and the file information is replicated at a set of nodes called the $file-set$ of the file. We define the file-set as the set consisting of the $k/2$ nodes closest to the fileId in either direction in the DHT. Again, as earlier, this is only one possible definition of the file-set. When a node A joins the network, it sends messages to the file-sets of each of its files; nodes that receive these messages add A's nodeId to the list of nodes that store the file. A's name is dropped from this list after a certain period of time; it is up to A to renew its files' registrations at the different file-sets. Arrival of new nodes into a file-set and departure of nodes from a file-set are handled easily because of the periodic file refresh messages.

2.3 Offsetting Inflation and Deflation

With time, the per-capita karma, i.e., the total karma divided by the number of active users varies. It inflates when nodes use up their money and go down, and deflates when nodes accrue karma and go down. If uncontrolled, the value of a unit of karma could go out of bounds. To prevent this, the outstanding karma in the system is periodically re-valued so that the per-capita karma is maintained at a constant level. The *Correction Factor*(ρ) applied to the karma is computed at the end of every epoch, according to $\rho = \frac{Karma_{old}.N_{new}}{Karma_{new}.N_{old}}$. Here $Karma_{old}$ is the total karma at the beginning of this epoch and N_{old} is the total active nodes at the beginning of the epoch. At the end of an epoch, each node in a bank-set transmits to all nodes a message containing (i) the number of nodes in the bank-set that went inactive in this epoch and their unused karma balance, (ii) the number of new nodes that joined the system in this epoch.

When a node receives these messages from all nodes in the system, it computes the current number of nodes in the system (N_{new}) and the current total karma in the system ($Karma_{new}$). Using the previously stored values of $Karma_{old}$ and N_{old}, the node computes the adjustment to be applied, applies it to accounts for which it is part of the bank-set and increments the epoch number. Because of the distributed nature of the correction, nodes could be in differ-

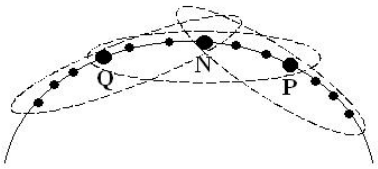

Figure 2: New node N joining the system: N gets the required account information by querying the $k-1$ nodes above and below it, i.e., the leaf-sets of P and Q.

ent epochs at the same time. When two such nodes engage in a transaction, appropriate currency conversion is made to maintain consistency. This scheme needs $O(N^2)$ messages to be transmitted at the end of each epoch, where N is the number of nodes in the system, but since each epoch typically spans several months, the cost of the global correction is acceptable.

3 Initialization

This section describes how a new node becomes part of KARMA. When a node enters the overlay, it has to be assigned a bank-set. This assignment has to be performed securely, as manipulating the bank-set assignment may allow a node to adjust its karma balance at will. A cryptographic puzzle [5] ensures that the assignment is random, and limits the rate at which a given node can join the system. To join KARMA, each new node selects a random K_{public} and $K_{private}$ key pair, and a value x such that $MD5(K_{public})$ equals $MD5(x)$ in the lower n digits, where n is a parameter that can be used to limit the difficulty of the puzzle. The nodeId is then set to $MD5(K_{public}, x)$, and the node certifies that it completed this computation by encrypting challenges provided by its bank-set nodes with its private key. Thus each node is assigned an id beyond its immediate control, and acquires a public-private key pair that can be used in later stages of the protocol without having to rely on a public-key infrastructure.

When node A enters the system, its potential bank-set members check to see if A was already a member of the system by looking for an entry for A in their databases. Each bank-set node sends to every other member of the bank-set (i)a message with A's account information if it finds A's entry (ii) a message indicating that A is a new member if it does not find an entry. These messages are signed by the private keys of the corresponding bank-set members, and therefore cannot be forged. If a majority of the bank-set indicates that A is a new node, then each node initializes A's account with a system-wide constant amount, and a sequence number of zero. Otherwise, each bank-node uses the balance and sequence number values supported by a majority of the bank-set. Consequently, the karma assignment is persistent, and previous solutions to the cryptographic puzzle cannot be reused to acquire new karma.

When a new node N comes up (Fig.2), it has to start functioning as a bank node for all nodes whose bank-sets now include N. In order to receive account updates from all relevant bank-nodes, N sends messages to the $k-1$ nodes that are above and below it in the identifier space and informs them of its entrance to the system. In response, the nearby nodes report the karma balances they are holding for each node that maps to a bank-set that includes N. N analyzes all reported bank balances corresponding to an account, and picks the value supported by the majority of the k nodes in that bank-set. Note that non-malicious members of the bank-set engaged in simultaneous karma transfers and are at different stages of the protocol may legitimately disagree on the current value of the account balance. Hence, if a majority consensus is not reached, the newly joining node waits a period of time before selectively polling that account value, until a majority consensus is established. A similar majority voting protocol is used to establish the sequence number.

Handling of a change in the bank-set due to a bank-node failure is similar to the case when a new bank-node comes in. When a bank-node P goes down, a new node R becomes part of the bank-set. The underlying DHT detects P's failure, and R initiates a similar discovery mechanism for accounts whose bank-sets now include R.

4 The Karma-File Exchange

The karma for file exchange forms the heart of our system. This exchange has to be karma-conserving and fair, i.e., the file-receiver's(say A) account has to be decremented by the karma-amount and the file-sender's(say B) account incremented by the same amount if and only if B sends A the required file. This is ensured by first making a *provable* karma-transfer from A's account to B's account, and then making a provable file-transfer from B to A.

When node A wishes to download a file F, it submits a download query to F's file-set, and starts a set of auctions for the file. The root-node of the file-set forwards the query to all nodes that store the required file(say nodes B and C). When B and C receive the query, they submit bids in the auction, and A chooses one of the bids, say the one submitted by node B, and the karma-for-file exchange is initiated. First karma is transferred from A's account to B's account, which is followed by the file chunk transfer. Auctions are continued to be held for each chunk of the file, till the entire file has been downloaded.

4.1 Karma Transfer

A karma transfer from a payer to a payee entails the deduction of a given karma amount from the payer's account, and the deposit of exactly the same amount to the payee's account. This section describes how this is done securely, and satisfying other properties required by KARMA.

The initialization procedures described in Section 3 require that a majority of the bank-nodes agree on the account value of each node. This requirement is satisfied by

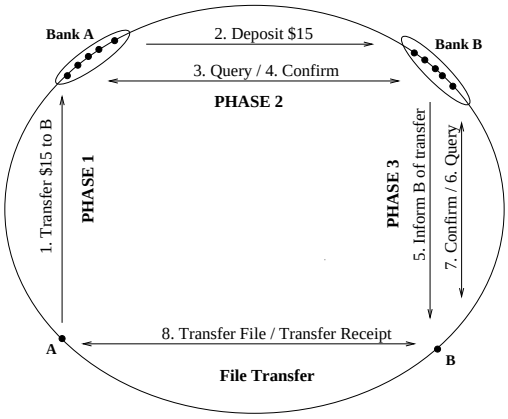

Figure 3: Karma-File exchange

ensuring that the required property is maintained whenever the account values change, i.e., at the end of each karma transfer, through the use of a simple synchronization mechanism.

A feature of the karma transfer protocol described below is that throughout the protocol, each bank set node decides whether to proceed with the transfer independently of all other nodes in the same bank set. KARMA takes advantage of the properties of the credit/debit interface to tolerate temporary inconsistencies between bank-set members. This obviates the need for expensive Byzantine consensus protocols.

A karma-transfer from a node A to node B involves A, B, $Bank_A$ and $Bank_B$. The transfer can be broadly split into three phases of communication between different pairs of entities: (1)A and *Bank*$_A$, (2)*Bank*$_A$ and *Bank*$_B$, and (3)*Bank*$_B$ and B (see Fig.3). A first sends *Bank*$_A$ a request to transfer a given amount of karma to B's account. *Bank*$_A$ then deducts this amount from A's account, and communicates with *Bank*$_B$. *Bank*$_B$ credits the same amount to B's account and informs B of the karma-transfer, so that B can proceed with the file-transfer to A. For security, the protocol has to take care to see that every one of these messages is authenticated. We now explain how this authentication is carried out at each step of the protocol.

The first transfer request sent by A is signed using A's private key, and the request includes a unique sequence number to avoid message replay. Also, after the first phase, *Bank*$_A$ nodes generate and store a log that contains details about the transfer; this makes the karma-transfer provable.

In the second phase, each member of *Bank*$_A$ sends messages to all members of *Bank*$_B$ requesting that the given amount be credited to B's account. *Bank*$_B$ nodes then send out a query to *Bank*$_A$ nodes, asking them to confirm whether they sent the previous messages. *Bank*$_A$ nodes respond with positive acknowledgements(ACKs) if they did, and with negative acknowledgements(NACKs) if they did not. If a *Bank*$_B$ node (say C) receives more than $k/2$ ACKs and less than $k/2$ NACKs, it proceeds with the transfer. If it receives less than $k/2$ ACKs, it aborts the transaction. If C receives more than k ACKs and NACKs, it means that spurious nodes that are not currently part of the bank-set are trying to influence the outcome of the transaction. To filter out such spurious responses, C sends to each *Bank*$_A$ member a random challenge, and authentic *Bank*$_A$ members respond with the challenge decrypted with their private keys, along with the (K, x, nodeId) tuple used during the nodeId generation (see Section 3). From the valid responses to the challenges, C picks those that are sent from nodeId's that are among the $k/2$ closest to A's bank-root in either direction. *Bank*$_B$ nodes now allow the transaction if the valid responses contain at least $k/2$ ACKs.

In the third phase, *Bank*$_B$ nodes inform B of the transfer. B verifies that a majority quorum exists using a mechanism similar to the one described above. B proceeds with the file-transfer to A if the verification succeeds.

To maintain the requirement of a majority agreement over the account values at the end of the karma transfer, members of each of the two involved bank-sets synchronize themselves by sending account values to one another and choosing the value suggested by a majority of the bank-set. If this synchronization is not done as part of each karma transfer, a chain of transfers with some message losses will, with high probability, result in correct bank-nodes disagreeing over account values, and the property of majority agreement being violated.

At every stage of the protocol, bank nodes independently decide whether to proceed with the transaction. To prevent malicious nodes from exploiting the lack of complete synchronization among different bank nodes, we incorporate the following features into the first phase of the protocol: (i)Automatic deduction of karma from the account, whether or not the node has enough karma to pay for the transfer. (This means that the account balance could fall below zero.) (ii) An attempt to ensure that every other bank-node in the bank-set gets the transfer request: Randomly pick another bank-node in the bank-set and forward the request to it.

These features make the transfer process commutative in the presence of multiple requests: the account balance is the same irrespective of which transfer request is seen first by a bank node. The features prevent malicious users from trying to have illegally high account balances by sending their requests to exactly a majority of the bank-set, thus keeping other bank nodes in the dark about the transaction. This allows us to execute a karma transfer without resorting to expensive agreement protocols among bank nodes.

An obvious observation that can be made from the preceding discussion is that the KARMA system requires the participating nodes to perform work on behalf of other nodes, and KARMA itself may suffer from freeloaders who keep accounts in the system without shouldering its load!

To prevent this, KARMA can compensate bank-set members for taking part in transactions by awarding them with a small amount of karma. However, care must be taken to avoid two potential problems. First, performing more than one transaction in response to a single transaction will create a chain reaction and grind the system to a halt. A suitable dampening function, e.g. awarding nodes extra karma only after a node has performed 10^4 transactions, can address this problem. Second, providing extra karma to participants will violate the zero-sum properties of karma transactions and exacerbate inflation, so taxing the resource provider, or consumer, or both, might be a simpler solution that preserves the zero-sum property.

4.1.1 Karma Transfer Without Overlay Routing

The time required for a karma transfer can be greatly reduced if, instead of sending messages over the overlay, as suggested earlier, the messages are transmitted directly between the communicating parties, bypassing the overlay. To run the first phase of the transfer without resorting to overlay routing, we would now need each node to explicitly know the IP addresses of the members of its bank-set. This is realized via a challenge-response protocol when a node joins a bank, where each bank-node responds to a challenge provided by the node, thereby proving its nodeId and its membership in the node's bank.

During the second phase, each node in payer A's bank-set proves to payee B's bank-set its nodeId by responding to challenges. The failure test suggested in [4] is used by B's bank-nodes to ascertain that the communicating set of nodes form the legitimate bank-set of A. Once a node in B's bank-set is sure that it has received messages from a majority of A's bank-set, it sends a message to B informing it of the transfer. B knows that the transfer is successful when it receives such messages from a majority of its bank-set.

This strategy eliminates all transmissions over the overlay during a karma-transfer, at the cost of some computational burden at the participating nodes, and should lead to a significant improvement over the earlier method.

4.2 File Transfer

We use the Certified Mail Scheme [6] for a provable file transfer mechanism. The proof of delivery here is the receipt for the delivery of the file signed with the receiver's private key. Briefly, the sender first sends the receiver the file encrypted with a secret DES key, and then the sender and the receiver run the protocol, through which the receiver gets the key to decrypt the file if and only if the sender gets the required receipt. This transfer is carried out directly between the two nodes involved, and not over the overlay.

If node A makes a payment to node B for a certain file, but B does not send A the file, A informs $Bank_A$ of this; $Bank_A$ talks to $Bank_B$, and $Bank_B$ asks B to produce the appropriate receipt. Since B did not send A the file, it would not have the required receipt either; so $Bank_B$ would transfer the karma back from B to A.

Note that the use of this mechanism is not limited to file-sharing applications alone; it can be used in any scenario where the required resource can be expressed as a sequence of bytes. This sequence of bytes could be a file in a file-sharing application, or the result of a computationally intensive function in a grid-computing system. The same mechanism can still be used to transfer the end-result after the karma transfer. The use of a currency independent of any single type of resource means that KARMA can be implemented to work with different resource sharing applications at the same time.

5 Possible Attacks

We now present a list of possible attacks that can be launched against the system, and describe how our system handles these attacks.

Replay Attacks: Replay attacks are ruled out by the use of sequence numbers and signatures when a node authorizes its bank-set to transfer karma in the first step of the karma transfer protocol, and the verification mechanism employed by any bank-set when some other bank-set wants to deposit karma.

Malicious Provider: A provider that accepts payment but fails to complete the transaction can be contested, and the karma repaid back to the consumer.

Malicious Consumer: A malicious consumer who fraudulently claims that he did not receive services even though he did is thwarted by the use of certificates. The provider simply provides the certificate to his bank-set when the transaction is complete.

Corrupt Bank-set: The use of the secure entry algorithm ensures that it is not feasible to target a bank-set. Assume that an attacker has compromised 10% of a 10^6 node network. Denoting by X the number of nodes controlled by the attacker in a $given$ bank-set, we have: $Exp(X) = 6.4$, and the probability of this attacker acquiring the majority of a 64-member bank-set: $P(X > 32) = P(X > (1+4)6.4) < (\frac{e^4}{5^5})^{6.4} = 5.6 \times 10^{-12}$ The probability that the attacker controls the majority in $some$ bank-set is less than the above value multiplied by the total number of bank-sets, i.e., 5.6×10^{-6}.

Attacks against DHT routing: Secure routing [4] , with the use of appropriate signing of messages, ensures reliable message delivery even when up to 25% of the nodes in the system do not adhere to the prescribed routing protocol.

Denial of Service Attack: Malicious nodes that send dummy NACKs to break a karma-transfer are thwarted by the checks employed to see if the NACKs originate from the authentic bank-set.

Sybil Attacks: In a peer-to-peer domain without external identifiers, any node can manufacture any number of identi-

ties [7]. This is a fundamental problem in any P2P system. The use of an external identifier, such as a credit card number or unique processor id, would address this problem at the loss of privacy. We permit Sybil attacks but limit the rate at which they can be launched through our secure entry algorithm

Spurious Files: Malicious members could put up dummy files giving them popular names, leading to downloads of the spurious files by unsuspecting users. While it is inherently difficult to counter this attack, some security can be achieved by generating the fileId based on the content hash of the file as well as the name of the file. Users seeking a particular file could then choose the most common fileId associated with the file name.

6 Related Work

Fair-sharing of Resources in P2P Systems: Ngan et al in [8] present a design that enforces fair-sharing in P2P storage systems. Their goal is to ensure that the disk-space a user is willing to put up for storing other users' files is greater than the space consumed by the user's files on other disks. Whether a user is really storing the files he says he is storing is verified by random audits. This design makes use of the fact that the resource in contention is spatial in nature: any user's claim that he is storing files for other users can be verified after the claim is made. This design cannot be extended to the scenario we are concerned with, namely the contention for temporal resources like bandwidth; here the resource contribution is not continuous across time.

Micropayment Schemes: A number of micropayment schemes [9] have been proposed to support lightweight transactions over the internet, such as making a small payment for accessing a page at a restricted site. The primary aim of these schemes is to enable a level of security commensurate with the value of the transaction, while having almost negligible overhead. Some schemes also provide a degree of anonymity to the parties in a transaction via trusted common brokers. Unfortunately, almost all these schemes require a trusted centralized server. Also many micropayment schemes assume the existence of brokers that give out currency to users, and then handle the deposit of currency from the vendors. These assumptions of trusted parties do not translate well into a peer-to-peer domain.

Microeconomic Models for Resource Allocation in Distributed Systems: Various decentralized microeconomic schemes have been proposed to solve resource allocation problems such as load balancing and network flow problems in computer systems [10]. The KARMA economy presented in our paper is similar to the pricing economic models proposed in these systems. In these systems, different resource consumers and resource consumers act as independent agents in a selfish manner to maximize their respective utility values. The proposed strategies that maximize individual utility values can be overlaid on top of the KARMA economy as well.

Applying Mechanism Design to P2P systems: Shneidman et al in [11] advocate the use of mechanism design in p2p systems to make users behave in a globally beneficient manner. KARMA, by tracking each user's resource contribution, aims to do the same.

7 Conclusions

In this paper, we propose an economic framework for discouraging freeloader-like behavior in a peer-to-peer system, and provide the design of a file-sharing application based on this framework. In this framework, each node has an associated bank-set that keeps track of the node's karma balance, which is an indicator of its standing within the peer community. The bank-set allows a resource consuming operation by the node only if the node has sufficient karma in its account to allow the operation. Safeguards protect the system against malicious nodes that may attempt to manufacture karma, acquire services from peers without providing them with karma, or acquire karma and refuse to provide services. Built on top of a peer-to-peer overlay, the proposed design can complement other peer-to-peer services and force nodes to achieve a parity between the resources they provide and the resources they consume.

8 Acknowledgements

We thank the referees for their helpful comments.

References

[1] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proc. MMCN 2002*, San Jose, Jan. 2002.

[2] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), Oct. 2000.

[3] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems.In *Proc. IFIP/ACM Middleware 2001*, Heidelberg, Germany, Nov. 2001.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Secure routing for structured peer-to-peer overlay networks. In *Proc. OSDI02*, Boston, Dec. 2002.

[5] S. Goel, M. Robson, M. Polte, and E. G. Sirer. Herbivore: A Scalable and Efficient Protocol for Anonymous Communication. *Cornell Univ. CIS Tech. Rep.*, TR2003-1890, Feb. 2003.

[6] B. Schneier *Applied Cryptography*, John Wiley and Sons, 2nd edition, 1995.

[7] J. Douceur. The Sybil attack. In *Proc. IPTPS 02*, Cambridge, Mar. 2002.

[8] T. Ngan, D. S. Wallach, and P. Druschel. Enforcing Fair Sharing of Peer-to-Peer Resources. In *Proc. IPTPS 03*, Berkeley, Feb. 2003.

[9] P. Wayner. *Digital Cash: Commerce on the Net.*, Morgan Kaufmann, 2nd edition, Apr. 1997.

[10] D. F. Ferguson, C. Nikolaou, J. Sairamesh, and Y. Yemini. Economic Models for Allocating Resources in Computer Systems. In S. Clearwater, editor, *Market Based Control of Distributed Systems*. World Scientific Press, 1996.

[11] J. Shneidman, and D. Parkes. Rationality and Self-Interest in Peer to Peer Networks. In *Proc. IPTPS 03*, Berkeley, Feb. 2003.