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 file-sharing 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.

Three fundamental properties stemming from the peer-to-peer domain guide the design of our system. First,

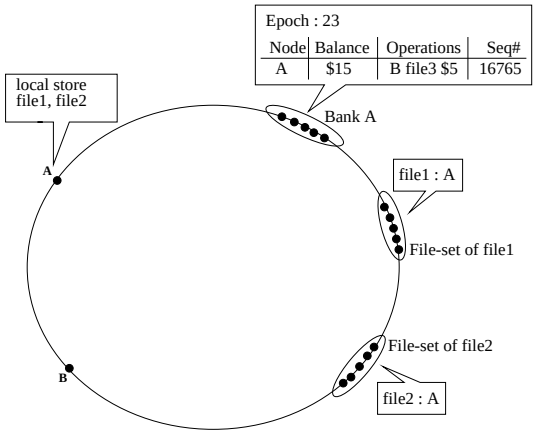

Fig. 1: Overview of system state. A has a balance of \$15 karma, and has recently paid B \$5 for file $file3$.

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

KARMA maintains all of its internal state in a peer-to-peer distributed hash table (DHT). The bank-set *Bank*$_A$ of a node A is a set of k peers that independently maintain the karma balance of that node. KARMA uses the distributed hash table to map nodes to bank-sets. Each participant and each unique peer node are assigned a secure, random identifier in a circular identifier space. The k closest nodes in the identifier space to *HASH(nodeId(A))* constitute the bank-set of A. 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], whose security and resilience to attacks is well-studied [4]. Picking k consecutive hosts for the bank-set allows the secure routing to the bank-set to be performed efficiently.

Each member of *Bank*$_A$ stores the amount of karma in A's account, signed with A's private key, as well as a transaction log containing recent payments A has made to other nodes. Signing of the balance by A ensures that the value is tamper-resistant. The transaction log acts as proof of A's payment, and comes into play if the other party in the transaction does not send 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 roughly constant, as described in Section 2.3 on inflation and deflation. 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

Systems employing KARMA will need to provide their own mechanisms for peering participants to exchange resources, and to agree on a reasonable amount of karma for the requested resource. While KARMA leaves this decision entirely to KARMA-based applications, we illustrate how a typical file transfer application is integrated with KARMA. In a KARMA-based filesharing system, each file is assigned a $fileId$ through some consistent hashing mechanism. The node closest to the $fileId$ serves as a rendezvous point for people who are offering and seeking that file. When a node A joins the network, it publishes its identifier under the $fileId$ of each file it has available for downloads. A node seeking to download a particular file acquires this list and initiates an auction by asking providers to submit a karma bid to transfer the file in question. It then selects the lowest bidder, though other alternatives, such as second-price auctions are also possible. To reduce communication overhead and ensure freshness, file advertisements are lease-based and expire after a certain amount of time unless refreshed.

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 leave the system, and deflates when nodes accrue karma and leave. 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 $\rho = \frac{Karma_{old}.N_{new}}{Karma_{new}.N_{old}}$, where $Karma_{old}$ 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 $Karma_{new}$ and N_{new}, 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 different epochs at the same time. When two such nodes engage in a transaction, appropriate currency conversion is made to maintain consistency. E.g.: if the correction has been applied at the payee's bank-set, but not yet applied at the payer's bank-set, the amount credited to the payee is the amount paid by the payer scaled by the correction factor. 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 does not lead to an unacceptable burden on the network.

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 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 corresponding 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 signed by A and recent transaction history 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 bank-node receives more than $k/2$ such messages agreeing on A's balance and sequence number, it uses the indicated balance for A. Otherwise, the bank-node initializes A's account with a system-wide constant amount, and a sequence number of zero. 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, it has to start functioning as a bank node for all nodes whose bank-sets now include N. N contacts the relevant bank-nodes, and obtains the required account balances using a majority vote with signed data, similar to the procedure described above. 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 on the balance and sequence number is not reached, the newly joining node waits a period of time before selectively polling that account value, until a majority consensus is established.

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 exchange transaction forms the heart of our system. Once a consumer node A selects a provider B according to the procedure outlined in Section 2.2, the file can exchange hands in return for karma. This exchange has to be karma-conserving and fair, that is, file-receiver A's account has to be decremented by the transaction cost and the file-provider B's 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.

4.1 Karma Transfer

Throughout the Karma transfer protocol, each bank set node acts 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.

The karma transfer from A to B is carried out in the following fashion: (see Fig.2) A first sends to B a signed message authorizing $Bank_A$ to transfer a given amount of karma to B. B forwards this message to its bank-set, who contact $Bank_A$ in turn. If A has sufficient karma

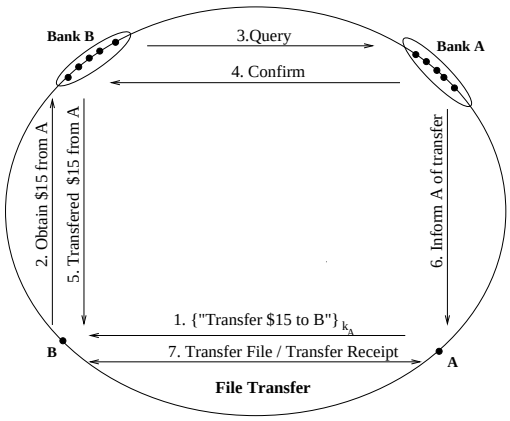

Fig. 2: Karma-File exchange

in its account to fund the transaction, the amount is deducted from A's account and credited to B's account, and 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, and how the protocol operates without the aid of expensive agreement protocols among bank nodes.

The first transfer request sent by A includes the balance A would have at the end of the transaction, and is signed using A's private key. The request includes a unique sequence number to avoid message replay.

B forwards the request along with its own signed balance to *Bank*$_B$. Each *Bank*$_B$ node then sends to *Bank*$_A$ queries that include the original transfer request sent by A. *Bank*$_A$ nodes check if the balance signed by A is indeed the valid balance of A, and if so, reply to *Bank*$_B$ with positive acknowledgements. *Bank*$_A$ nodes then deduct the given amount from A's account, and inform A of the transfer. *Bank*$_B$ nodes that get a majority of positive responses from *Bank*$_A$ credit the amount to B's account, and inform B of the transfer. B can now proceed to transfer the given file to A.

Signing of balances by both A and B maintains the invariant that correctly functioning bank-nodes have the latest signed bank-balances corresponding to each of their client nodes, thus preventing malicious bank-nodes from setting balances to arbitrary values.

At every stage of the protocol, bank nodes independently decide whether to proceed with the transaction. It is possible for a bank node to perceive transactions in a different order from other members of the same bank-set. Commutativity of addition guarantees that, once the node stops initiating new transactions and messages have propagated, bank members will agree on the same bank account balance. This observation greatly simplifies the KARMA protocol by obviating costly agreement protocols, though it has a non-intuitive side-effect. A node's account balance at a particular bank-node may temporarily dip below zero, only to be restored when a later credit goes through. For instance, suppose a node with a zero account balance provides a file for y karma, and purchases a file for x karma, where $x < y$. A bank-set member that receives the debit for x before the credit for y informs the corresponding bank-set that the transaction should not continue. But if it is overruled by a majority of its own bank-set who perceived the credit before the debit, it goes through the protocol and locally adjusts the account balance to be $-x$. This accounting trick preserves commutativity when less than $k/2$ of the hosts perceive a different event ordering than the majority of the bank-set and allows the system to make forward progress without blocking.

Since KARMA does not require any distributed coordination between bank-set members, its performance is determined by two factors: (1) the DHT lookup latency for securely mapping a node to its bank-set, and (2) network transmission overhead between a k to k mapping of the provider's and consumer's bank-sets. We rely on the scheme suggested in [4] to perform the mapping securely, tolerating up to a fraction of malicious nodes in the peer-to-peer system. The time required for karma transfer is then reduced, since most messages are transmitted directly between the communicating parties, bypassing the overlay.

The preceding discussion outlined the basics of our approach for accounting for resource usage and contribution in a peer-to-peer system. By keeping track of a virtual currency that corresponds to how well-behaved users are, KARMA can force consumers to achieve parity between resources they consume and provide. However, the KARMA system itself requires the participating nodes to perform work on behalf of other nodes. In particular, each node is faced with the burden of following the protocol and keeping track of account information on behalf of other nodes for which it serves as a bank node. From a game-theoretic perspective, there is no strong incentive for a node to follow the protocol, and KARMA itself may suffer from freeloaders who keep accounts in the system without shouldering its load!

Luckily, the economic framework KARMA implements offers a ready solution: 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. However, 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. While KARMA has mechanisms that compensate for with inflation over large time frames, simply taxing the resource provider, or consumer, or both, might be a simpler solution that preserves the zero-sum property.

4.2 File Transfer

We use the Certified Mail Scheme [5] 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 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 enables KARMA to be incorporated into different peer-to-peer applications.

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.

Attacks against DHT routing: A faulty node that is part of the path used to deliver a message sent by a non-faulty node can attempt to disrupt message delivery by dropping the message entirely, or by modifying the contents of the message. 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.

Corrupt Bank-set: A malicious attacker that acquires a majority of a bank-set can manufacture any amount of karma for the nodes that map to that particular bank-set. The use of the secure entry algorithm, however, ensures that targeting a bank-set is not feasible. 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}. Therefore, the limiting factor to KARMA's tamper-resilience lies in the p2p routing substrate, and not in the higher level protocols.

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 identities [6]. 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

6 Related Work

Fair-sharing of Resources in P2P Systems: Ngan et al in [7] 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 [8] 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 of these schemes require a trusted centralized server. 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 [9]. 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 [10] advocate the use of mechanism design in p2p systems to make users behave in a globally beneficiary 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.

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, January 2002.

[2] E. Adar and B. Huberman. Free riding on Gnutella. *First Monday*, 5(10), October 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, November 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] B. Schneier *Applied Cryptography*, John Wiley and Sons, 2nd edition, 1995.

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

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

[8] P. Wayner. *Digital Cash: Commerce on the Net.*, Morgan Kaufmann, 2nd edition, April 1997. , 1996.

[9] 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.

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