Off-line Karma: A Decentralized Currency for Peer-to-peer and Grid Applications

Flavio D. Garcia and Jaap-Henk Hoepman

Institute for Computing and Information Science,
Radboud University, Nijmegen, The Netherlands.
{flaviog,jhh}@cs.ru.nl

Abstract. Peer-to-peer (P2P) and grid systems allow their users to exchange information and share resources, with little centralised or hierarchical control, instead relying on the fairness of the users to make roughly as much resources available as they use. To enforce this balance, some kind of currency or barter (called *karma*) is needed that must be exchanged for resources thus limiting abuse. We present a completely decentralised, off-line karma implementation for P2P and grid systems, that detects double-spending and other types of fraud under varying adversarial scenarios. The system is based on tracing the spending pattern of coins, and distributing the normally central role of a bank over a predetermined, but random, selection of nodes. The system is designed to allow nodes to join and leave the system at arbitrary times.

Keywords Decentralised systems, micropayments, free-riding, security, grid, peer-to-peer.

1 Introduction

Peer-to-peer (aka. *P2P*) networks like BitTorrent [9], Gnutella [15] and Freenet [8], and grid systems like XGrid [1] are distributed systems without centralised control or hierarchical organisation. Given this flat structure, these systems scale very well when the number of nodes increases. Scalability is important, given the fact that the Internet is still growing exponentially and more people have permanent Internet connections.

Grid systems capitalise on the observation that computer resources are usually very badly distributed in both time and space, and that almost all of them are wasted most of the time. CPU cycles are maybe the best example of this. In an ideal grid system, the whole Internet constitutes a huge supercomputer with practically unlimited resources, that members can use as long as they contribute to it as well. Projects like seti@home, folding@home and distributed.net have shown that a large set of common desktop computers can provide a tremendous amount of

computing power. Even though they receive no direct benefit, users participate in such projects because they associate themselves with the goals of the project. If such large scale computations are for an uncompelling cause, it is not easy to find people willing to donate their CPU time.

Also, many P2P networks suffer form the 'free-riders' problem where users only occasionally connect to the network to use the resources offered by it, but do not donate any resources themselves. Adar and Huberman [3] performed a traffic study on the Gnutella network revealing that 70% of the users share *no* files at all. To counter such problems, 'currencies' of some sort have been proposed to reward users contributing to the system and that can be used as payment when the resources of the network are used.

This paper extends our earlier work in this area [12,11]. We refer to those papers for a more in depth discussion of the state of the art, and only briefly summarise it here. Several P2P systems use some kind of digital currency to enforce contribution and optimise resource distribution. All these systems use a central bank or broker to track each user's balance and transactions. Micropayment schemes [20,13,19] seem to be especially suitable for such a task. However, these schemes are centralised and the load of the central broker grows linearly with the number of transactions. It is clear that when scalability is of primary concern, a central bank or broker constitutes both a bottleneck as well as a single point of failure.

At the moment, the only distributed currency we are aware of that is fully decentralised is KARMA [23]. In that system, the bank for a user is distributed over a bank set of r users, that all need to be on-line, and that are all involved in all transactions between their "owners". This incurs a large overhead, especially in cases where the transaction rate is high.

Another interesting approach is PPay [24]. PPay is a lightweight micropayment scheme for P2P systems. In PPay the issuer of the coin is responsible for keeping track of it. With every transaction the issuer of the coin updates a pointer to the new owner, in a secure manner. The main drawback with PPay is that it uses a central server (called broker) when the issuer of a coin is off-line. Therefore, in certain situations PPay converges to a system with a centralised accounting bank.

1.1 Our Contribution

We present a completely decentralised, off-line karma implementation for *dynamic* P2P and grid systems, that detects double-spending and other types of fraud under varying adversarial scenarios. Previous work of us [12,11] focused on the static case. The system is based on the tracing

of the spending pattern of coins, and distributing the normally central role of a bank over a predetermined, but random, selection of nodes. Transactions between users do not require the cooperation of this distributed bank — this is more efficient, but as a result double spending cannot be prevented. Instead, karma coins need to be occasionally reminted to detect fraud. The system is designed to allow nodes to join and leave the system at arbitrary times.

We focus on the payment for CPU cycles as an application of our techniques, and show how a special minting technique allows us to initialise the system and provide its users with coins in a quite autonomous fashion. Coins correspond to CPU cycles, and the bag of coins owned by a user corresponds, in a sense, to a battery charged with CPU cycles. The battery is initially empty, and can be charged by minting coins. Minting takes quite a few CPU cycles. Alternatively, a coin can be obtained by performing roughly the same amount of work, but for another user. Extensions of our protocols to trade coins for other resources are certainly possible, and only involves initialising the system with a set of coins in a different manner.

We design our system on top of an arbitrary overlay network which provides certain services as described in Section 2.

The remainder of this paper is organised as follows. Section 2 discusses the model and notation used throughout the paper. We describe the system objectives and the capabilities of the adversary in Section 3. Then we present our karma implementation in Section 4, first for a static network and then the dynamic case. Finally, Section 5 discusses methods for early double-spending detection, and Section 6 presents our conclusions and directions for further research.

2 System Model

In the context of this paper we want to stay abstracted from the underlying overlay network. We are going to model common characteristics that apply to routing overlays like CAN [18], Chord [22], Pastry [21] and Tapestry [25] as in [5], were the reader can find also a nice and brief description of each system. In this abstract model, every node that joins the system is assigned a uniform random identifier u from the identifier space Π. We assume that the overlay network provides primitives for both user look-up and message routing. Furthermore, for each possible identifier u (whether u is part of the network or not) the overlay network can efficiently and reliably compute the *neighbour set* $\aleph_r(u)$, which consist

of all on-line nodes *close* to u. The definition of *close* varies in each of the above mentioned systems, although it is always well-defined. We also assume that communication within this set is *efficient*, because nodes keep updated routing information of their neighbours. Given that the node identifiers are distributed randomly, any neighbour set represents a random sample of all participating nodes [5].

Off-line Karma requires every user to have his own public key pair (PK, SK) and a certificate that binds the public key with a node identifier. This may be provided by a trusted *Certification Authority* (aka. *CA*). We want to remark that the CA is only needed when a new user joins the system. After that communication with the CA is no longer needed.

Routing information in the overlay network is kept updated, in practise, by node join and node leave messages and periodic queries and fingers to detect when a node suddenly disconnects. This mechanism introduces propagation and update delays. This is, in fact, a discrete approximation of an ideal situation where any modification in the network topology is instantaneously detected by the overlay network. We assume such an ideal situation, and leave responsibility for emulating this ideal functionality in an efficient fashion to the overlay network. We also assume that node joins and leaves are atomic operations.

We also assume that the overlay network is capable of safely distributing a blacklist of banned users. Whenever a user detects fraud and has a proof of that, he can submit it to the overlay network which makes this information available to every user. How to implement the distribution of blacklist securely is beyond the scope of this paper.

2.1 Notation

We write $\{m\}_u$ for u's signature on message m, C_u for u's certificate, and $\texttt{validSig}(m, u, C_u)$ for the function that checks u's certificate C_u, and if valid uses the key in C_u to verify a signed message m.

We also use a multisignature scheme. A multisignature scheme [17,16] is a signature scheme where a set R of users sign a message. A multisignature $\{m\}_R$ for a message m has the same properties as if each user in R concatenates his own traditional public key signature to m, the only difference is that a multisignature is more efficient in size and in verification time (comparable to a single signer Schnorr's signature scheme). Unlike a threshold signature scheme however, it does not provide anonymity. We define $C_R = \{C_i : i \in R\}$ and $\texttt{validSig}(m, R, C_R)$ is the function that checks the certificates and verifies the multisignature.

Security of our system is parameterised by a security parameter s. All cryptographic primitives we use satisfy the requirement that the advantage of the adversary breaking them is less than 2^{-s}. We show that the advantage breaking our karma system is at most that large too.

For describing protocols we adopt the notation $a \rightarrow b : m \rightarrow m'$ denote that Alice sends a message m to Bob which he receives as m'. Also $a : f$ means Alice computes f. If f is a predicate, Alice verifies $f \equiv true$ and aborts if not.

3 System Objectives and Threat Model

3.1 Threat Model

We consider a set of n users U of which at most t are under control of the adversary. In the context of P2P networks, there is an important difference in the difficulty for an adversary between adding new corrupted users to the system and getting control over chosen users. Therefore, we also define $0 \leq c \leq t$ to be the number of corrupt users chosen by the adversary after they joined the overlay network. Then, when $c = t$ we give the adversary full control over which nodes in the overlay get corrupted, while for $c = 0$ the adversary is only able to get a randomly chosen set of corrupted users of size t.

Furthermore, we assume that the adversary cannot make excessively many nodes join and leave the system, or let some nodes join and leave in a very high frequency (in attempts to mount sybil attacks, to use them as strawmen, or to overcome the random assignment of node identifiers). In fact, we do not allow the adversary any control over when nodes join the system. In practise, this could be achieved by requiring nodes to pay each time they register, or making node joins a time-intensive procedure (e.g., by requiring them to compute a moderately hard, memory bounded function [2,10]).

3.2 System Objectives

We note that for any system offering off-line currency, double-spending *prevention* is generally speaking not possible, unless extra assumptions (e.g., special tamper proof hardware) are made. As we are designing an off-line system, we only require double spending detection. We do not consider issues like fair exchange or coin stripping. We focus on the payment itself and not on the exchange of coins for goods.

Then, the requirements on a usable, off-line and decentralised, karma system for P2P and grid applications are the following.

Scalability Transaction cost should be independent of the size of the network.

No centralised control The system should not rely on one or several central, special, nodes (e.g., banks or brokers) and should not require any predetermined hierarchy. We do allow a centralised registration procedure.

Load Balance The overhead of the protocol is, on average, evenly distributed over the peers.

Availability Transactions among users can be processed uninterrupted even when users join or leave the system.

Double-spending detection The system must detect double-spending, and for every double spent coin, a fraudulent user should be blacklisted.

4 The Off-Line Karma Protocol

4.1 Informal Description

To implement the CPU cycles battery metaphor presented in the introduction, a user can mint coins by finding collisions on a hash function (a la hashcash [4]). This rather expensive minting process is preferred over giving an initial amount of free coins to new users, as in that case the system becomes vulnerable to users changing their identities after spending those coins. A minted coin contains the name of the minting user as well as a sequence number (limiting the number of coins a single user can mint). User identity and sequence number together constitute the unique coin identity. Coins also contain a time stamp recording the time they were minted.

The coins are transferable [6]. A user can pay for resources by transferring a coin to another user. The sender signs the coin, and the receiver verifies this signature and stores the coin (with signature) for further use. With every transfer, a coin is extended with another signature. Thus, the sequence of signatures on a coin record the payment history of that coin. Double-spending is detected by comparing the history of two coins with the same coin identity, and the culprit (or his accomplice) will be found at the node where both histories fork. This check is performed whenever a coin is reminted. Fraudulent nodes are blacklisted, together with a proof of their misbehaviour (namely two signatures of the fraudulent node over the same coin). This prevents unfair blacklisting.

Every once in a while (but at least before the coin expires), coins must be reminted. Reminting is used to detect double-spending, and at

the same time to reduce the size of the coin by removing its history. In classical systems, reminting is done by a central bank. Here the function of the bank is distributed over a set of users on the network called the *reminters* for the coin. The set of reminters is constructed in such a way that

- at least one of the reminters is a non-corrupted node, and
- all honest reminters possess the history of previously reminted coins with the same identity.

We first describe the static case where we assume to have a set of n users which are always on-line and later, in Section 4.3 we describe the modifications needed for handling dynamic networks, where users join and leave at arbitrary times.

4.2 Off-Line Karma for Static Networks

Minting Let $h_1 : A \to C$ and $h_2 : B \to C$ be hash functions, and suppose every user is allowed to mint 2^q karma coins. A user u has to spend some CPU time finding a collision y satisfying: $h_1(x) = h_2(y)$ and $x \neq y$, with

$$x = \underbrace{u||sn}_{coinId}||ts$$

where sn is the serial number $|sn| \leq q$ and ts is a time stamp. This is an expensive but feasible operation, for suitable functions h_1 and h_2. In analogy with the monetary system, imagine that the cost of the metal needed for minting a coin is greater than its nominal value. We define the new karma coin as

$$k_0 = \langle x, y \rangle$$

Spending To spend a coin, a user u transfers ownership of it to the merchant m, by putting a signature over the coin together with the merchant identity m and a random challenge z it receives from the merchant. The random challenge is included to avoid uncertainty about who is the traitor in the case where a user spends the same coin twice at the same user. Otherwise, a fair user might look like the double-spender (unless he keeps a history of received coins forever). Concretely, suppose that the user s owns the coin k_i and wants to spend it at the user m. Then, the last one sends a random challenge z to the first one who computes:

$$k_{i+1} = \{k_i, z, m, C_u\}_u$$

and sends it to m.

Reminting To prevent the coins to grow unreasonably large and to bound the amount of history that needs to be kept, coins must be reminted regularly, at least within the time to live T. This bank functionality is performed by a random but predefined set of users R_k. The selection of this set must be done in such a way that

- each user is responsible for reminting roughly the same amount of coins (load balance) and
- at least one honest user is a member of the remint set.

Whenever a user u has to remint a coin k, he sends it to each user in the remint set $R_k = \aleph_r(h(id(k)))$. Here the hash function is used as a consistent mapping from the coin identifier space to Π. Each user in R_k must verify the authenticity of k and store it in his local history database. If the verification succeeds, the reminters will create a multisignature

$$k_{new} = \{\texttt{XY}(k), ts, R_k, C_{R_k}, u\}_{R_k}$$

for the new coin with the same coin identifier and owner, but with a new time stamp ($\texttt{XY}()$ extracts the collision out of k). If the verification fails, either because the coin is invalid or because a coin with the same identifier and time stamp was already reminted, the reminters will audit the coin and trace back the cheater in the signature chain.

Protocol Description.

Minting For a user u:
Initially: $K_u := \emptyset; sn_u := 0$
$sn_u := sn_u + 1$
$ts := \texttt{now}()$
$x := u||sn_u||ts$
Find y satisfying: $h_1(x) = h_2(y)$
$k := \langle x, y \rangle$
$K_u := K_u \cup \{k\}$

Spending User u spends a coin at merchant m:
m : pick nonce z
$m \rightarrow u : z$
u : select $k \in K_u$
$u \rightarrow m : \{k, z, m, C_u\}_u \rightarrow k'$
$m : \texttt{check}(m, k', z)$
$u : K_u := K_u \backslash \{k\}$
$m : K_m := K_m \cup \{k\}$

where $\texttt{now}()$ returns the current time and K_u is the bag of coins of user u

Reminting User u remints a coin $k = \{\tilde{k}, z, u, C_s\}_s$:
$u : R = \aleph_r(h(id(k)))$
$u \rightarrow r_i : k, \texttt{now}(), R \rightarrow k', t', R' \quad \forall_i : r_i \in R$
$r_i : t' \approx \texttt{now}()$
. $\quad R' = \aleph_r(h(id(k')))$

. $\texttt{check}(u, k', \bot)$
. $\texttt{verifyHistory}(k')$
. $k_{new} = \{\texttt{XY}(k'), t', R', C'_R, u\}$
$R \leftrightarrow u : \{k_{new}\}_R \rightarrow k_R$ (this is a three-round protocol)
$u : \texttt{checkBase}(u, k_R)$

$\texttt{check}(u, k, z)$:
$\texttt{if isBase}(k)$ $\texttt{then checkBase}(u, k)$
$\texttt{else}$ $\{k', z', u', C_s\}_s := k$
. $\quad$ $\texttt{return}$ $(z' = z \vee z = \bot) \wedge u' = u$
. $\quad\quad \wedge \texttt{validSig}(k, s, C_s) \wedge \texttt{check}(s, k', \bot)$

$\texttt{checkBase}(u, k)$:
$\texttt{if isReminted}(k)$ $\texttt{then}$
. $\quad \{k', newts, R', C_R, u'\}_R := k$
. $\quad$ $\texttt{return}$ $u' = u \wedge R' = R = \aleph_r(h(id(k')))$
. $\quad\quad \wedge newts \in [\texttt{now}() - T, \texttt{now}()]$
. $\quad\quad \wedge \texttt{validSig}(k, R, C_R)$
$\texttt{else}$
. $\quad \langle x, y \rangle := k$
. $\quad u'||sn||ts := x$
. $\quad$ $\texttt{return}$ $h_1(x) = h_2(y) \wedge u' = u$
. $\quad\quad \wedge ts \in [\texttt{now}() - T, \texttt{now}()]$

$\texttt{audit}(k, k')$:
$\{\ldots\{k_0, z_1, u_1, C_0\}_{u_0} \ldots, z_m, u_m, C_{m-1}\}u_{m-1} := k$
$\{\ldots\{k'_0, z'_1, u'_1, C'_0\}_{u'_0} \ldots, z'_{m'}, u'_{m'}, C'_{m'-1}\}u_{m'-1} := k'$
$\texttt{for}(i = 1$ $\texttt{to}$ $\min(m, m'))$ $\texttt{do}$
. $\quad \texttt{if}(z_i \neq z'_i \vee u_i \neq u'_i)$ $\texttt{then return}$ u_{i-1}

$\texttt{verifyHistory}(k)$:
$\texttt{Hcoin} := \{k' \in H |$ $\texttt{id}(k) = \texttt{id}(k') \wedge \texttt{ts}(k) = \texttt{ts}(k')\}$
$\texttt{foreach}$ $k' \in \texttt{Hcoin}$ $\texttt{do}$
. $\quad B := B \cup \{\texttt{audit}(k, k')\}$
$H := H \cup \{k\}$
$\texttt{return Hcoin} = \emptyset$

where B is the set containing all the blacklisted users and H is the set of all reminted coins.

Security Analysis. We will show that our protocol is secure by showing that for every double-spent coin, a corrupted node is blacklisted.

Lemma 1. *Let r be the size of the remint set R. If $r > \gamma s + c$, for some constant γ, then the probability that R contains no honest nodes is less than 2^{-s}.*

Proof. Since c nodes can be corrupted by the adversary at will, $r > c$. So we need to see how large the probability is that the remaining $r - c$ nodes happen to be taken form from the remaining $t - c$ corrupted nodes when constructing the set R. We define a random variable X equal to the number of honest nodes in R, given that c nodes in R are already corrupted. We want

$$P(X = 0) < 2^{-s} \tag{1}$$

were s is our security parameter. As $t < n$ we have

$$P(X = 0) = \frac{\binom{t-c}{r-c}}{\binom{n-c}{r-c}} < \left(\frac{t-c}{n-c}\right)^{r-c}$$

and we want

$$\begin{aligned} \left(\frac{t-c}{n-c}\right)^{r-c} &< 2^{-s} \\ \{\tfrac{t-c}{n-c} < 1\} & \\ r - c &\geq \log_{\frac{t-c}{n-c}} 2^{-s} \\ r &\geq -s\left(\log_{\frac{t-c}{n-c}} 2\right) + c\,. \end{aligned}$$

This completes the proof. □

Lemma 2. *Given a coin k, $t = \mathtt{ts}(k)$, there is no relevant information in k after $t + T$.*

Proof. The proof is split in two cases.

- If k is never double-spent in the period [t,t+T] then there is no relevant information at all.
- If k is double-spent first at time t' with $t < t' < t + T$ then:
  - If k is reminted before t' then the new coin $\hat{k}$ with $\mathtt{ts}(\hat{k}) > t$ contains the proof of double-spending and therefore there is no relevant information in k.
  - If k was not reminted before t' then both double-spent coins k_1 and k_2 must be reminted at least once before $t + T$(they would expire otherwise). Then any double-spending attested by k is detected before $t + T$. □

Theorem 1. *Whenever a coin is double-spent, that coin expires or one corrupted node is identified.*

Proof. Whenever a coin is double-spent, both coins have the same identifier and time stamp. It is not possible for an adversary to change any of them: in case of a just minted coin they are protected by being part of the collision of the hash functions; and in the case of a re-minted coin it is not possible for an adversary to forge the multisignature, given that Lemma 1 ensures that there is always a fair user in every remint set. Then, coins with the same identifier must be sent to the same remint set before their expiration time, otherwise they expire and the condition of the theorem holds. Therefore, at least one fair user $\dot{u}$ must receive both coins before its expiration time. Let k_{i_1} and k_{i_2} be the first remint request of each version of the double-spent coin k_i, received by $\dot{u}$ after the double-spending. Then, $\dot{u}$ detects fraud and calls `audit`(k_{i_1}, k_{i_2}). `audit` first checks whether the signatures are valid. It is clear that a user endorsing a coin with an invalid signature or that is improperly minted is faulty (he should have checked it). If that is not the case, then the coin is fairly minted and $id(k_{i_1}) = id(k_{i_2})$, at least the first user endorsing the coin is the same in k_{i_1} and k_{i_2}. Therefore, and given the fact that both coins are different, there must be one user in the signature chain that transferred the coin to two different users (or to the same user twice). In this case the userIds inside of the signature are different (or the nonces are different), which constitutes a proof of double-spending. □

4.3 Handling Dynamic Networks

In a static network, the remint set R_k for a coin k never changes. That makes easy to verify that a given coin was fairly reminted at some point in the past, as verifying a remint set is as trivial as checking $R_k = \aleph_r(h(id(k)))$.

In a dynamic network, it is not possible to be so restrictive while defining a valid remint set. Otherwise every time a user $u_o \in R_k$ is off-line, the coin k cannot be reminted, and therefore may expire. On the other hand, the selection of the users in R should somehow be predefined in order to limit the influence of the adversary, and at least allow the validity of the remint set to be reliably determined at a later time (unless we require $r > t$, which trivially implies that at least one fair node is in the remint set).

As a solution we define a valid remint set for a coin k, as the closest r users to $h(id(k))$ in the identifier space, that are on-line at remint time.

Then the verification of the fairness of a remint set is difficult, given that the verifier has no information about the state of the network at remint time. An adversary could try to unfairly construct a remint set with only nodes that are under his control, by claiming that all other (fair) users were off-line at remint time. We are going to prevent this kind of attack by taking the density of the set R as an indicator for the authenticity of the coin. We define the density as

$$d(R_k) = \max_{i \in R_k} |i - h(id(k))| \ .$$

Let us assume that the density of the overlay network does not change very fast. Meaning that it is very unlikely, in a worldwide network with a large amount of users, to have big fluctuations in the amount of users connected to it, in short periods of time. Let α be the maximal rate of change for the density of the overlay network, i.e. if T is the maximal time to live for a coin, and $d(t)$ is the density at time t, then for all t' between t and $t+T$, we have $\frac{1}{\alpha} d(t) \leq d(t') \leq \alpha d(t)$.

We call a remint set *acceptable* (for a coin) if it satisfies our constraints on the remint set, and does not contain members beyond the boundaries specified by the density. In such a scenario, an adversary does not have much freedom while selecting the users in R without drastically increasing $d(r)$.

Another issue that needs to be addressed in a dynamic network is the history transfer between users. The neighbourhood R_k for a coin k should keep as an invariant the history of any reminted coin within the period $[\mathtt{ts}(k), \mathtt{ts}(k)+T]$. Given that the history consists of signed coins, it is not possible for an adversary to forge it. Therefore, a joining user can just renew its history by querying its neighbours for it.

```
checkBase(u, k):
if isReminted(k) then
.     {k', newts, R', C_R, u'}_R := k
.     return u' = u ∧ R' = R
.          ∧ newts ∈ [now() − T, now()]
.          ∧ d(R) ≤ α d(now())
.          ∧ validSig(k, R, C_R)
else
.     ⟨x, y⟩ := k
.     u'||sn||ts := x
.     return h_1(x) = h_2(y)
.          ∧ u' = u ∧ ts ∈ [now() − T, now()]
```

The only modification that remains, with respect to the static version, is the function `checkBase`, which now verifies the density of the remint set, instead of the equality with the neighbourhood.

Security Analysis. We analyse security of the dynamic protocol similar to the static case.

Proposition 1 (Hoeffding bound). *For a hyper-geometrically distributed random variable X, representing the number of successes among n draws, with probability p of success we have [14,7]*

$$P(X \geq np + g) \leq e^{-2g^2/n}$$

Lemma 3. *Let $p = \frac{t-c}{n-c}$, fix β such that $c \leq \beta r$, and suppose $\beta + \alpha^2 p < 1$. If $r \in O\left(\frac{\alpha^2 s}{(1-\beta-p\alpha^2)^2}\right)$, then any acceptable remint set contains at least one honest node with probability $1 - 2^{-s}$.*

Proof. The remint set is fixed at remint time t. The adversary needs to pick r nodes for the remint set such that it does not violate the acceptability condition $d(R) \leq \alpha d(t')$, which is checked the next time the coin is reminted at time $t' \leq t + T$. At t', the density $d(t') \leq \alpha d(t)$ This means that at time t it can, at best, select r nodes from the first $\alpha^2 r$ nodes from the root of the coin and then take control over c of them. It is successful if among these $\alpha^2 r$ nodes there are $r - c$ faulty ones.

Let X be a random variable representing the number of faulty nodes in such a sample of $\alpha^2 r$ nodes from all n nodes ($t - c$ of which are faulty). Then the adversary is successful if $X \geq r - c$. X is distributed according to the hyper-geometric distribution, with $p = \frac{t-c}{n-c}$, and we are interested in bounding

$$\begin{aligned} P(X \geq r - c) &\leq P(X \geq r - \beta r) \qquad \{\beta + \alpha^2 p < 1\} \\ &= P(X \geq p\alpha^2 r + (r - \beta r - p\alpha^2 r)) \qquad \{\text{Hoeffding bound}\} \\ &\leq e^{-2(r-\beta r-p\alpha^2 r)^2/\alpha^2 r} = e^{-2r(1-\beta-p\alpha^2)^2/\alpha^2} \end{aligned}$$

which we want to be less than 2^{-s}. Then, by taking logarithms

$$\log_2 e(-2r(1 - \beta - p\alpha^2)^2/\alpha^2) < -s$$

and hence

$$r \geq \frac{\alpha^2 s}{2(1 - \beta - p\alpha^2)^2 \log_2 e}$$

which completes the proof. □

Lemma 4. *In every remint set, fair nodes can always transmit their remint history to another fair node before leaving.*

Proof. As a corollary of Lemma 3 and given the assumption that node joins and leaves are atomic operations, at least two fair nodes must be in a valid remint set, whenever a fair node is going to leave it. This fact, together with the secure routing assumption over the overlay network, implies that fair users can always transmit their remint history to another fair node before leaving.

Theorem 2. *Whenever a coin is double-spent, that coin expires or one corrupted node is identified. (the proof in Theorem 1 also applies here)*

5 Early Double-spending Detection

In some scenarios double-spending detection might not be good enough. This is the case when an adversary is able to add new corrupted nodes easily. It is possible for a corrupted user who owns a karma coin, to spend it many times and very quickly, especially when the coin is just minted (or reminted). Although those actions are eventually going to be detected, this is not going to happen until the first two remint-request of this coin are submitted. This user of course is going to be punished, but then the adversary might get another Id and repeat this operation. To counteract this kind of attacks, besides making it harder for an adversary to get new ids, it is possible to detect double-spending early. As a first line of defence, when a user receives a new coin, he performs a search over the coins he possess looking for duplicated identifiers. In case he succeeds, the double-spender is immediately blacklisted. The probability of finding a duplicated coin just like that is small, especially when the number of copies is not too big. To improve this, we introduce coin attractors to the system. An attractor is a user, whose hashed id is the closest to the hashed id of the coin. Then, when a user s wants to spend a coin at the merchant m, s searches over his coins for the one which has the minimum distance with the merchant's hashed id,

$$k_d = \min_{k \in K_s} |h(m) - h(id(k))| \ ,$$

and pays with it. Even thought faulty nodes may avoid sending coin to attractors, eventually a good node will do so. At that point the attractor will detect the double spending.

6 Conclusions

We have presented a completely decentralised, off-line karma implementation for P2P and grid systems, that detects double-spending and other types of fraud under varying adversarial scenarios. This is, so far, the first system for truly off-line karma coins, which can be used in highly dynamic peer-to-peer networks and grid systems. Our system outperforms previously proposed system of similar characteristics, under certain scenarios. In particular, we are able to completely replace a central bank by a distributed remint set whose size is roughly proportional to the security parameter s.

Several interesting research questions remain. For instance, the length of a coin increases with every transaction, and involves several public-key cryptographic operations. This is quite heavyweight, in contrast with micropayment schemes that are usually associated with the kinds of value transfers we consider here. One open area of research is to investigate the use of micropayment techniques in off-line scenarios like karma. Another question is whether the use of trusted computing enabled nodes allows for more efficient implementations of karma.

References

1. Xgrid website. http://www.apple.com/acg/xgrid/.
2. M. Abadi, M. Burrows, M. Manasse, and T. Wobber. Moderately hard, memory-bound functions. In *Proceedings of the 10th NDSS*, pages 25–39, San Diego, CA, Feb. 2003. Internet Society.
3. E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), Oct 2000. http://firstmonday.org/issues/issue5_10/adar/index.html.
4. A. Back. Hashcash - a denial of service counter-measure. http://www.cypherspace.org/hashcash, Mar. 1997.
5. M. Castro, P. Druschel, A. J. Ganesh, A. I. T. Rowstron, and D. S. Wallach. Secure routing for structured Peer-to-Peer overlay networks. In *Proceedings of the 5th OSDI*, Operating Systems Review, pages 299–314, New York, Dec. 9–11 2002. ACM Press.
6. D. Chaum and T. P. Pedersen. Transferred cash grows in size. In R. A. Rueppel, editor, *Advances in Cryptology—EUROCRYPT 92*, volume 658 of *LNCS*, pages 390–407. Springer-Verlag, 1992.
7. V. Chvtal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.
8. I. Clarke, O. Sandberg, B. Wiley, and H. Hong. Freenet: a distributed anonymous information storage and retrieval system. In *International Workshop on Design Issues in Anonymity and Unobservability*, pages 311–320, 2000.
9. B. Cohen. Incentives build robustness in bittorrent. In *Proceedings of the Workshop on Economics of Peer-to-Peer Systems*, Berkeley, CA, USA, 2003.

10. C. Dwork, A. Goldberg, and M. Naor. On memory-bound functions for fighting spam. In D. Boneh, editor, *Advances in Cryptology – CRYPTO ' 2003*, volume 2729 of *LNCS*, pages 426–444. International Association for Cryptologic Research, Springer-Verlag, 2002.
11. F. D. Garcia and J.-H. Hoepman. Off-line karma: Towards a decentralized currency for peer-to-peer and grid applications (brief abstract). In *Workshop on Secure Multiparty Computations (SMP)*, Amsterdam, The Netherlands, Oct. 7–8 2004.
12. F. D. Garcia and J.-H. Hoepman. Off-line karma: A decentralized currency for static peer-to-peer and grid networks. In *5th Int. Networking Conf. (INC)*, 2005. (to appear).
13. S. Glassman, M. Manasse, M. Abadi, P. Gauthier, and P. Sobalvarro. The MilliCent protocol for inexpensive electronic commerce. In *Fourth International Conference on the World-Wide-Web*, pages 603–618, MIT, Boston, Dec. 1995. O'Reilly.
14. W. Hoeffding. Probability inequalities for sums of bounded random variables. *J. Amer. Statist. Assoc.*, 58:13–30, 1963.
15. P. Kirk. Gnutella. http://rfc-gnutella.sourceforge.net.
16. S. Micali, K. Ohta, and L. Reyzin. Accountable-subgroup multisignatures: extended abstract. In P. Samarati, editor, *Proceedings of the 8th CCS*, pages 245–254, Philadelphia, PA, USA, Nov. 2001. ACM Press.
17. K. Ohta and T. Okamoto. Multi-signature scheme secure against active insider attacks. In *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, pages E82–A(1): 21–31, jan 1999.
18. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable Content-Addressable network. In R. Guerin, editor, *Proceedings of the ACM SIGCOMM 2001 Conference (SIGCOMM-01)*, volume 31, 4 of *Computer Communication Review*, pages 161–172, New York, Aug. 27–31 2001. ACM Press.
19. R. L. Rivest. Peppercoin micropayments. In A. Juels, editor, *Proceedings Financial Cryptography '04*, volume 3110 of *LNCS*, pages 2–8. Springer, Feb 2004.
20. R. L. Rivest and A. Shamir. PayWord and MicroMint: Two simple micropayment schemes. In M. Lomas, editor, *Proceedings 1996 International Workshop on Security Protocols*, volume 1189 of *LNCS*, pages 69–87, Cambridge, United Kingdom, Apr 1997. Springer-Verlag.
21. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.
22. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
23. V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: a secure economic framework for peer-to-peer resource sharing. In *Proceedings of the Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, California, 2003. Papers published on Website: http://www.sims.berkeley.edu/research/conferences/p2pecon/index.html.
24. B. Yang and H. Garcia-Molina. PPay: micropayments for peer-to-peer systems. In V. Atluri and P. Liu, editors, *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS-03)*, pages 300–310, New York, Oct. 27–30 2003. ACM Press.
25. B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE J-SAC*, 22(1):41–53, January 2004.