Running on Karma – P2P Reputation and Currency Systems

Sherman S.M. Chow

Department of Computer Science
Courant Institute of Mathematical Sciences
New York University, NY 10012, USA
schow@cs.nyu.edu

Abstract. Peer-to-peer (P2P) systems allow users to share resources with little centralized control. Malicious users can abuse the system by contributing polluted resources. Moreover, selfish users may just connect for their own benefits without donating any resources. The concepts of reputation and currency give possible approaches to address these problems. However, to implement these ideas is non-trivial, due to the non-existence of a single trusted party. Existing works circumvent this by placing trust assumption on certain nodes of an overlay network. This work presents a new reputation system and a new currency system. Our designs are simple thanks to the full use of the trust assumption.

Keywords: Decentralized systems, peer-to-peer, reputation, currency, micropayments, free riding, Byzantine agreement.

1 Introduction

In a peer-to-peer (P2P) network, every user is playing the roles of server and client simultaneously, i.e. there is a minimum central control. The last decade demonstrated many successful applications of the P2P computing model. The most popular one may be P2P file sharing, such as BitTorrent [5], Gnutella [12], etc. Thanks to the network's flat structure, P2P systems scale very well with the number of nodes. However, the lack of centralized control makes P2P systems suffer from many security problems.

Since every user can play the role of the "server" and share their resources (e.g. music files) to others, malicious users can abuse the system by contributing polluted resources. On the other hand, P2P systems suffer from the "free rider" problem, which means users only connect to the network for the resources they want, but not contribute any resources. A traffic study has shown that 70% of Gnutella network's users (at the time of the study) are not donating any file at all [1].

You will not ask for a file from someone who is notorious in polluting the network. This is where a reputation system comes to play. Generally speaking, a reputation system establishes trust among members of a community where each member does not have prior knowledge of each other, by integrating feedback from the peers to conjecture the trustworthiness of other peers.

F. Bao et al. (Eds.): CANS 2007, LNCS 4856, pp. 146–158, 2007.
© Springer-Verlag Berlin Heidelberg 2007

To fight against free riders, a natural solution is to offer users some "tokens" for their contributions, which enables them to retrieve their wanted resources at a later stage. Old approaches realize this concept in the software level. For example, KaZaA uses a measure of participation level [17], defined by upload/download ratio and the integrity rating, to prioritize among peers during periods of high demand. The participation level is stored locally. As one may expect, cracking tools are available. What we need is some kind of currency or barter that is transferable for exchanging resources, i.e. an electronic currency system over P2P networks.

1.1 Our Contribution

Putting a reputation system or an electronic currency system over P2P networks is non-trivial, due to the non-existence of a trusted party. Existing works circumvent this by placing a trust assumption on certain nodes of an overlay network. However, these solutions are rather inefficient, in terms of bandwidth or computational requirements.

Since we place trust on some nodes in the network anyway, why don't we fully utilize this trust assumption to avoid unnecessary operations? This work tries to make full use of these trusted nodes, to propose two simple and efficient systems.

1. Simple P2P Reputation System from Byzantine Agreement (SPRBA): Current approaches either require cryptographic schemes that cannot be realized in a P2P network (e.g. [9]), or require contacting with every online node to get responses about one's behavior history (e.g. [6]). The latter approach is expensive in terms of storage capacity and bandwidth. Moreover, it is difficult to enforce security in a large-scale dynamic system. In the face of a sybil attack that returns many negative comments, the security of the system depends on whether most of the responding nodes are trustworthy. On the other hand, SPRBA just places trust on a smaller set of nodes that are determined by a cryptographic hash function.
2. Simplified Karma (Karma$^+$): Off-line Karma [8] is a completely decentralized currency system for dynamic P2P networks. Extensive use of digital signatures is used to certify the transfer of electronic coins (e-coins), which means that the size of a e-coin will eventually get huge[1]. To resolve this issue, it is assumed that a set of nodes in the overlay network chosen by a cryptographic hash function will reduce the coin size by re-issuing signatures. Since the new signatures certify the current ownership of the coin, these re-issuing nodes are assumed trustworthy. If such an assumption is made, why don't we just assume these nodes to take the role of a "bank"? This gives the underlying idea of our proposed Karma$^+$.

[1] Even an aggregate signature scheme [3] is used; the messages being signed must be stored, which makes the coin bulky.

2 System Model

2.1 Reputation System

Our system only cares about the retrieval, casting and maintenance of the votes. Issues like how the reputation is modeled (e.g. [10]) and assessed (e.g. [11]), and whether a bad vote is cast even the resource is good, are beyond the scope of this work.

Our general mechanism is as follows. Before initiating the download, *requestors* can assess the reliability of the sources by polling peers (*vote maintainers*, or VM). After the transfer, the requestors now take the role of voters, returning their opinions to the peers for later retrieval by others. We also discuss how these votes are maintained in face of offline VMs.

Apart from the obvious constraint of no centralized control, our system should satisfy the following properties.

1. *Scalability.* Transaction cost should be independent of the size of the network. For example, our system would not flood the whole network.
2. *Load balance.* The overhead incurred by our system should be evenly distributed over the nodes, on average.
3. *Robustness.* Our system should be robust against denial-of-service attack. For example, the adversary cannot manipulate the protocol messages to turn the internal state distributively held by different nodes into an inconsistent state.

2.2 Currency System

There are only three components in our currency system: minting, spending and double-spending detection. Users first mint their own e-coins. In spending, the ownership of a coin is transferred. When a coin is spent, the receiver should check whether the coin has been double-spent. It is possible to come up with a proof when some user double-spends his coin. In additional to the above properties for a P2P reputation system, the objectives of our e-coin system are:

1. *Efficient verification.* It is efficient to verify the validity of the coin (e.g. whether it has been double-spent). This also implies the e-coin size should be small.
2. *Oblivious transaction history.* The transaction history related to the coin should not be included in the coin.

Other issues, like coin stripping and the fair exchange of coins and resources, are outside our scope. We note that double-spending *prevention* is generally not possible for an offline currency system, without tamper-proof hardware.

2.3 Overlay Network

This work assumes the existence of an underlying overlay network that provides primitives for both user look-up and message routing. In such a network, every

node is assigned a uniform random identifier $u \in \mathcal{U}$ from an identifier space, and one can always efficiently and reliably compute the neighbor set $\aleph(u)$, which consists of all currently online nodes *close* to an identifier u. The exact definition of "close" depends on the actual overlay network one is using, but it is one of the essential and well-defined components of overlay network. Readers are suggested to refer routing overlays like CAN [13], Chord [15], Pastry [14] and Tapestry [16], as reviewed in [4].

We assume an ideal situation for the updating of routing information, where the overlay network instantaneously detects any change in the network topology. This depends on how well the underlying overlay network emulates this ideal functionality by a discrete approximation from constantly fingering to see whether a joined node has left. We also assume joining and leaving are atomic operations.

Finally, we assume that the overlay network has a blacklist mechanism. Whenever a fraud is detected, a user can somehow submit a proof of it to the overlay network and a blacklist will then be safely distributed.

In the rest of the paper, we use a cryptographic hash function $h : \{0,1\}^* \to \mathcal{U}$ to map an arbitrary bit-string to an element in the identifier space $\mathcal{U}$ of the overlay network. We use the notation of $\aleph^*(y) = \aleph(h(y))$ to denote the "neighbor set" corresponding to the bit-string y.

2.4 Certificate Authority

In Karma$^+$, we assume every user has his/her own public key and private key pair (PK, SK) along with a certificate certifying the binding between the public key and a node identifier. A certificate authority (CA) is only needed when a new user joins the system, and no other communication with the CA is needed. We require the CA to perform any task that substantiates the assumption required in the unforgeability of the underlying signature scheme, e.g. the knowledge of secret key assumption. No CA is required in SPRBA.

2.5 Threat Model

Our basic threat model assumes the network has a total of n users, and at most t of which are under the adversary's control.

For an overlay network, one can actually make a distinction between adding a user to the network and compromising an existing user. Let c $(0 \leq c \leq t)$ be an integer denoting the number of users corrupted by the adversary after they join the overlay network. An adversary who can compromise whoever he wishes means $c = t$, while $c = 0$ means all he can do is injecting random users. We also assume the adversary cannot make excessively many nodes to join the system. In practice, this can be done by requiring a node to compute a time-intensive operation (e.g. [7]) for each joining. Looking ahead, this is also the way an e-coin is minted in Karma$^+$.

3 Related Work

3.1 A Reputocracy System

A "reputocracy" system is presented in [9], which is a reputation system based on electronic voting from homomorphic encryption and storage enforcement protocols. The votes are for the resource requesters to make comments which affect the reputation of a resource contributor. The enforcement protocol is for showing a file of a certain complexity has been transmitted. We discuss this work to exemplify that some required cryptographic schemes cannot be easily deployed in a P2P setting.

Reputation is maintained by the votes that one receives. These votes are stored by the nodes responsible for tallying, which are called the tallying center. The tallying center of a node is determined by some globally-known hash function, i.e. different nodes will have different tallying centers, instead of a single global one. After a file transfer is done, the tallying center uses the storage enforcement protocol to get the cost of the communication $\$ > 0$. Then, the requester will cast an encrypted vote that is either $+\$$ or $-\$$, depending on whether he/she is satisfied with the file obtained. A zero-knowledge proof of the vote is either $+\$$ or $-\$$ is prepared. The tallying center has no idea whether a vote is good.

Problems: Note that only a single node is assigned as the vote maintainer, thus attack is relatively easy. Besides, it is not specified that who is responsible for the decryption of the tally. A natural choice is the tallying center, but it is meaningless to have the zero-knowledge proof in this case since he/she can decrypt anyway. Moreover, to the best of our knowledge, e-voting systems require the use of a bulletin board, even for those that are "self-tallying". Such a bulletin board is essentially a public-broadcast channel *with memory*, which is costly, if not unavailable, in a P2P network.

3.2 P2PRep

P2PRep [6] is a reputation system enabled by a peer review process. Each node keeps track of and shares with others the information about the reputation of their peers. Reputation sharing is based on a distributed polling protocol. After locating a list of servants who owns the wanted resources, the requester polls his/her peers about the reputation of servants in this list. Peers wishing to respond send back a reply, then the requestor selects a subset of them and contacts them directly. Their replies are integrated to make a decision. Additional mechanism can be added to poll for servant credibility, representing the trustworthiness of a servant in providing correct votes.

Problems: The requesters just ask around about one's reputation, and these broadcast messages occupy the network bandwidth significantly. Moreover, one's reputation depends on whether the peers in the previous transaction remain online. In the face of the free rider problem, this reliance is not desirable.

Regarding security, no mechanism is controlling who can respond. It is entirely possible that a whole bunch of malicious nodes respond with negative comments.

The suggested solution in [6] is to have "suspects identification", by computing cluster of voters whose characteristics suggest that they may have been created by a single malicious user. However, it depends on how good the clustering algorithm distinguishes between "consistent votes for bad behavior from many users" and "forged votes from a single malicious user". Even such an algorithm exists, it is implemented in the software level. Nothing prevents the adversary from re-engineering the algorithm and tailor-making bad votes accordingly.

3.3 Off-Line Karma

Off-line Karma [8] is a decentralized electronic currency system. A coin is minted by finding a collision of a hash function (of "small" output domain size) [2] with hash input including the owner of the coin, a serial number and the current time. At the very beginning, a user finds collisions and prepares a list of coins. Specifically, the user needs to find a y such that $H(U||sn||ts) = H'(y)$, where $H(\cdot)$ and $H'(\cdot)$ are two different hash functions, U is his node identifier, sn is a serial number and ts is a time stamp.

Transfer of a coin (i.e. spending) is done by a chain of signing. Suppose y is a coin corresponding to node A (i.e. $H'(y) = H(A||sn'||ts')$ for some serial number sn' and time stamp ts'), A gives it to another node B by signing on $(y, A \rightarrow B, z_A)$ where z_A is a random nonce. B spends this coin with U_C by signing on $(y, B \rightarrow C, z_B)$. Everyone can verify that C is the current owner of the coin since it is originally owned by A, A has certified the transfer $A \rightarrow B$ and B has certified $B \rightarrow C$.

As a result of a series of spending, the coin size will get huge eventually. To slim it, *re-minting* is done. Re-minting party is the neighbor set of the coin $\aleph(y)$ (i.e. treating the coin as a user) of the overlay network. *All* of them sign on y and the current time to certify the current ownership.

The timestamp associated with a coin also serves as an expiry time. Re-minting must be done before expiration. Double-spending is detected in re-minting. The nodes in the set $\aleph^*(y)$ will check whether there exists two different signatures signed by the same party. A random nonce is introduced to avoid the uncertainty about who is the traitor when a user spends the same coin twice with the same user.

Problems: Before re-minting, the coin size is large. One needs to do a series of signature verifications to verify the current ownership of a coin. Yet, these computations do not help double-spending detection at all. A shorter time frame can make the re-minting happen more often, but keep in mind that one needs to contact every node in $\aleph^*(y)$ for re-minting. On one hand, the coin may expire before every such node can be reached. On the other hand, it gives a higher computational burden (to verify all signatures associated with the coin and issue a new one) to these nodes.

Double-spending can only be detected at the stage of re-minting, but not when the coin is spent. To make the double-spending detection "happens earlier", the suggested solution in [8] is to ask each user to spend a coin that is the nearest (in the context of neighbor set) to the one whom he wants to initiate a transaction.

However, a malicious user who wants to double-spend a coin can do anything deviated from the protocol. Only having good users following such a suggestion is clearly not sufficient to make double-spending detection any earlier.

Finally, the transaction history is included in every coin, which violates the requirement of oblivious transaction history and is undesirable.

4 SPRBA – Simple P2P Reputation System from Byzantine Agreement

The main components of our proposed SPRBA are as follows.

4.1 Retrieval

Suppose A is the requestor and B is a candidate who owns some resources A wants. A performs the following:

1. A computes $\aleph^*(B)$ to get the VM groups maintaining B's reputation,
2. A sends an enquiry to each node in $\aleph^*(B)$ for B's reputation.
3. A makes a decision according to the information received (e.g. taking the majority).

Suppose a node C is being asked for B's reputation, C first performs a (one-time) verification to confirm his/her membership in the set $\aleph^*(B)$. If C has no record about B, he/she just returns so. From this point, C knows the list of VMs for B, which will be used in the maintenance phase.

4.2 Casting

After the file transfer, A now wants to cast a vote about B. The voting information will include his/her identity A, the current time ts and other auxiliary information (e.g. the name of the file being transferred). What A needs to do is just broadcasting his/her vote to $\aleph^*(B)$. The consistency functionality we want from the broadcast is ensured by a Byzantine agreement protocol among the nodes in $\aleph^*(B)$.

4.3 Maintenance

VMs may go offline. When they go online next time, they should catch up with other VMs. The first retrieval request gives the list of the "partners" in maintaining the reputation of someone. Synchronization is done by identifying which vote is missing in one's own record but exists in a threshold portion of the partners.

4.4 Analysis

Our system is scalable in the sense that no flooding of the whole network is needed to retrieve one's reputation. Due to the uniformity of the hash function's

output, the incurred computational burden and the storage load are uniformly distributed across the network. Thus load balance is achieved.

There is a subtle difference between a resource requestor retrieving the reputation and VMs retrieving the reputation. For the former case, it just affects the one-time decision of whether a transaction should be carried. On the other hand, VMs do it for updating their own record to truly reflect the reputation of those nodes they are responsible for, which affects all the future reputation requests. Consider a malicious voter who sends different votes to different VMs, VMs who were offline before have no idea which vote is the "real" one during synchronization. Eventually, nothing useful can be inferred from the votes maintained distributively across the VMs. This is why Byzantine agreement is needed for each vote.

Byzantine agreement is a rather costly procedure, so we should keep the size of the VM group $\aleph^*(B)$ as small as possible. However, this parameter also governs the probability for the adversary to succeed in biasing the reputation.

Suppose r denotes the number of nodes output by $\aleph^*(B)$, f denotes the percentage of malicious nodes in this set of r nodes ($0 \leq f \leq 1$). Under the basic threat model that the adversary can only inject random nodes to the network; if 50% majority is the rule to make a decision, the probability for an adversary to succeed, i.e. having more than a half of nodes under his control, is given by $C^r_{r/2}(f(1-f))^{r/2}$. Depending on the actual scenario, says the security level we want, we should tune r accordingly.

5 Karma$^+$ – Simple Offline Electronic Currency System

Utilizing SPRBA, we can actually realize a simple electronic currency system. Instead of having the node-group chosen by the function $\aleph^*(\cdot)$ to manage the reputation of a node, we require the node-group to certify the current ownership of a coin. However, different from the reputation system that every node can cast a vote, there is only one node that can "change" the current ownership of a coin – the coin owner. We thus require cryptographic primitives providing authentication and non-repudiation to make it possible, i.e. digital signature schemes.

If signatures are used, Byzantine agreement is not necessary in our case since the recipient of the coin will actively check whether he/she is the new owner of the coin, and signatures signing on different messages can be used as a proof of misbehavior.

With these ideas in mind, our system turns out to be a simplified version of Off-line karma. We assume all nodes in the P2P network get the same set of system parameters, e.g. the maximum number of coins one can mint, the description of the hash function, and the signature scheme to be used, etc. The main components of our proposed SPRBA are as follows.

5.1 Minting

Minting of a coin involves finding a hash function collision [2] similar as that in Off-line Karma [8].

The user needs to find a $p+q$-long bit-string $y \in \{0,1\}^{(p+q)}$ such that[2] $H(x) = H'(y)$, for $x = (U||sn)$, where U is his node identifier of length p, sn is a bit string denoting a coin's serial number of length $|sn| = q$. This length restriction, together with the logic governing how the nodes managing this coin is determined, limits every user to mint up to 2^q karma coins.

The node identifier and the serial number uniquely determine the coin. The coin is defined as $\langle x, y \rangle$. For brevity, we call it "coin y". In contrast to Off-line Karma, a coin does not include the timestamp denoting the minting time.

5.2 Spending

We start by the case that a newly-minted coin is spent, followed by the case that a coin is spent by the one who did not mint it.

Suppose node A is spending a coin $\langle x, y \rangle$ minted by him/her with another node B. A sends y to each node in $\aleph^*(y)$ and notifies them B is now the coin owner of y by signing y together with a current timestamp ts.

The current owner of coin y is determined by the record held by the node-group $\aleph^*(y)$. We can view the node-group $\aleph^*(y)$ is performing the bank function of the coin y. They will keep track of the current coin ownership. We call these nodes the *"bank-nodes"*.

Even though a bank-node obtains the collision pair from someone else and is the authority to say who is the current owner of the coin. It is not true that a bank-node of a coin cannot be the initial owner of a coin, (when he/she is lucky enough to mint a coin y having the node-group $\aleph^*(y)$ including him/herself). The reason is that the minting node is specified in the x component of the coin. No other node would mint a coin on other's behalf. Even a malicious bank-node claims the ownership of a coin after obtained a collision pair $\langle x, y \rangle$ from some other node is not convincing since the node identifier in x does not match.

Each bank-node needs to check whether x is in the correct form (i.e. including A's identifier and a bit-string sn of length q), $\langle x, y \rangle$ really gives a collision, and A gives a signature that signs on the coin, the recipient's node identifier, and a recent enough timestamp, i.e. $(y||B||ts)$. If all the verifications go through, the coin $\langle x, y \rangle$ is then sent to B. B is convinced he/she is the new owner if responses from a threshold number of bank-nodes are obtained.

In the second case, the spender is not the original owner. Suppose B is the spender and C is the "merchant" that B is dealing with. B signs on the coin with the recipient C's identifier and a new timestamp, and gives the coin to C. C contacts each bank-node in $\aleph^*(y)$ to see whether B is the current owner of the coin. Specifically, this is done by taking the latest ownership status purported by $\aleph^*(y)$ as the current ownership status of the coin.

[2] Instead of using two different hash functions, one can actually use a single hash function by appending x with a bit 0 and y with a bit 1. Otherwise, an attack exploiting the symmetry is possible so that a single coin can be interpreted in two ways corresponding to different minting-node.

5.3 Maintaining the Current Ownership

In a dynamic P2P network, the bank-nodes of a coin may not be online at the same time. Each of them should keep the signatures certifying the transfer of ownership, until a threshold number of the bank-nodes got the same set of signatures and update their own records. In this way, the storage requirement on the bank-node is minimized. The latest ownership can be easily identified since the signature is binding with a timestamp. It can be considered as a hybrid approach that combines a pure-updating of the coin ownership and the signature chain approach used in Off-line Karma.

5.4 Double-Spending Detection

Double spending means the current owner of a coin spends it at two different users. To do so, the double-spender must have signed on two different messages specifying different recipients, which give a cryptographic evidence of the misbehavior.

On the other hand, someone who wants to "spend" the coin once he/she owned will not be treated as a double-spending. It will be treated as just an invalid request instead since it is essentially the same thing as having a random node claiming for the ownership of someone else's coin. Depending on the level of service one desires, the signature can also be submitted to the blacklist mechanism of the network to impose a certain kind of penalty on the one making the invalid request.

On the other hand, a bank-node may be malicious in falsely-accusing someone has double-spent. An accusation thus requires a signature by a bank-node on a message stating the current owner as well. If any abnormality is observed, one can simply forward the signature to all other bank-nodes for further investigation of whether a bank-node is being malicious or just not-up-to-date.

5.5 Efficiency Analysis

Load balancing follows from the uniform distribution of the hash output. Scalability can be seen from the simple design of our system. We require no flooding of the network and no complex operations other than minting of the coin. The complexity involved in minting a coin is actually a good thing to hinder the extent of sybil attack.

No heavy cryptographic operations are involved other than signature generation and verification. The current state-of-art signatures offer a short signature size that makes the protocol bandwidth-efficient. In contrast with Off-line Karma [8], each node only needs to verify a single signature instead of a series of them.

The storage and complexity requirements of the bank-nodes are minimal. Each node needs to sign every time the current owner of the coin is about to change, but it can always be pre-computed. Besides, the message to be signed is short, in contrast with the signature on the huge coin in Off-line Karma. It is true that the bank-node needs to perform signature verification every time the coin ownership is changed. The total number of signature verification is the same as

that in the re-minting phase in Off-line Karma. However, the computational cost is amortized in our case.

5.6 Security Analysis

It is easy to see that as long as there exists one honest node with up-to-date information among the bank-nodes, security is guaranteed. Here we assume the extended threat model that t nodes are controlled by the adversary and c of them are corrupted after they joined the overlay network (i.e. after they became the bank-nodes of a certain coin). Our analysis is similar to that of Off-line Karma [8].

Theorem 1. *Let r be the size of the bank-node set $\aleph^*(y)$ of a coin y. If $r > \gamma s + c$ for some constant γ, the probability that $\aleph^*(y)$ contains no honest nodes is less than 2^{-s}.*

Proof. By assumption, the adversary can compromise c nodes in the set $\aleph^*(y)$. All we need to show the probability that the remaining $r-c$ nodes happen to be taken from the remaining $t-c$ corrupted nodes to be included in the set $\aleph^*(y)$. Let X be the random variable of the number of honest nodes in the set $\aleph^*(y)$, we have

$$\begin{aligned}\Pr[X=0] &= \frac{C_{r-c}^{t-c}}{C_{r-c}^{n-c}} \\ &= \frac{(t-c)!/((r-c)!(t-r)!)}{(n-c)!/((r-c)!(n-r)!)} \\ &< (\frac{t-c}{n-c})^{(r-c)}\end{aligned}$$

Since we want to upper-bound the success probability of the adversary by 2^{-s}, note that $\frac{t-c}{n-c} < 1$, we have

$$\begin{aligned}(\frac{t-c}{n-c})^{(r-c)} &< 2^{-s} \\ r-c &\geq \log_{(t-c)/(n-c)} 2^{-s} \\ r &\geq -s(\log_{(t-c)/(n-c)} 2) + c\end{aligned}$$

Setting $\gamma = -(\log_{(t-c)/(n-c)} 2)$ completes the proof.

5.7 Improvements over Off-Line Karma

Apart from the efficiency gain like smaller coin size as revealed in the previous analysis sections, Karma$^+$ enjoys the following features over Off-line Karma.

No Coin Expiration. Timestamp is used in Off-line Karma for expiring the coin, and it forces the re-minting and let the re-minting party to have a chance to do double-spending detection. In Karma$^+$, a coin would not expire so the situation that a coin cannot be re-minted before its expiration is avoided. Besides, Karma$^+$ detects double-spending in every transaction, but not well after the coin is double-spent as in Off-line Karma.

Limit on the Maximum Number of Minting. Off-line Karma aims to limiting the number of coin one can mint by imposing a maximum length on the serial numbers. It is claimed that (U, sn) uniquely determines a coin. However, in addition to the owner identifier U and the serial number sn , time stamp ts is another varying factor. Note that the bank-node group $\aleph^*(U||sn||ts)$ is unlikely to contain a node that also appears in $\aleph^*(U||sn||ts')$ for a timestamp $ts' \neq ts$. It is difficult to discover two coins in the world are actually sharing the same U and sn. The number of coin one can mint is thus limited within a time period, and depends on the granularity of the time periods. However, the time-complexity of finding a collision already imposes an inherent limit on the number of coins one can mint at a given time.

Since the expiration mechanism is not necessary in our system, we can remove the inclusion of the timestamp in finding a collision pair, which means (U, sn) really serves as a unique identifier of a coin. In this way, we pose a limit on the maximum number of coins one can mint.

Higher Security in a Dynamic Network. Karma$^+$ works better in the case of dynamic network when compared with Off-line Karma [8]. The security of Off-line Karma depends on the nodes that are responsible for re-minting at the re-minting time, since the re-minting of the coin is done by asking all those nodes to give signatures to certify the new ownership of the coin. One has no information about the state of the network at that time, so an adversary may have unfairly constructed a re-mint set with only nodes that are under his control, giving signatures certifying himself, and claiming all other nodes were offline at that time. Karma$^+$ does not have this problem, since it is the verifier who computes this set and contacts the nodes according to the protocol for his own good.

6 Conclusion

We have presented two completely decentralized systems to address the pollution problem and the free-rider problem in peer-to-peer resources sharing applications. One is a reputation system and the other is an electronic currency system. Our systems outperform existing systems of similar functionalities under similar trust assumptions. Our simple design and efficiency gain are obtained from making full use of the trusted nodes.

Acknowledgements

Thanks to Lakshminarayanan Subramanian for his comments and support. Also thanks to Joël Alwen, Saurabh Kumar and Ning Ma for the discussions.

References

1. Adar, E., Huberman, B.A.: Free Riding on Gnutella. First Monday 5(10) (October 2000), http://firstmonday.org/issues/issue5_10/adar
2. Back, A.: Hashcash - a Denial of Service Counter Measure, http://www.hashcash.org/papers/hashcash.pdf

3. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In: Biham, E. (ed.) EUROCRPYT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003)
4. Castro, M., Druschel, P., Ganesh, A., Rowstron, A., Wallach, D.S.: Secure Routing for Structured Peer-to-Peer Overlay Networks. SIGOPS Operating Systems Review 36/SI, 299–314 (2002)
5. Cohen, B.: Incentives Build Robustness in BitTorrent. In: Workshop on Economics of Peer-to-peer Systems (2003)
6. Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P., Violante, F.: A Reputation-based Approach for Choosing Reliable Resources in Peer-to-Peer Networks. Computer and Communications Security 2002 , 207–216 (2002)
7. Dwork, C., Goldberg, A., Naor, M.: On Memory-Bound Functions for Fighting Spam. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 426–444. Springer, Heidelberg (2003)
8. Garcia, F.D., Hoepman, J.-H.: Off-line Karma: A Decentralized Currency for Peer-to-peer and Grid Applications. In: Ioannidis, J., Keromytis, A.D., Yung, M. (eds.) ACNS 2005. LNCS, vol. 3531, pp. 364–377. Springer, Heidelberg (2005)
9. Garcia-Martinez, A., Chuang, J.: A Cryptographic Reputation Scheme for Peer-to-peer Networks, `http://citeseer.ist.psu.edu/550626.html`
10. Gupta, M., Judge, P., Ammar, M.: A Reputation System for Peer-to-Peer Networks. In: NOSSDAV 2003. Network and Operating Systems Support for Digital Audio and Video, pp. 144–152 (2003)
11. Kamvar, S.D., Schlosser, M.T., Garcia-Molina, H.: The Eigentrust Algorithm for Reputation Management in P2P Networks. In: World Wide Web Conference 2003, pp. 640–651 (2003)
12. Kirk, P.: Gnutella, `http://rfc-gnutella.sourceforge.net`
13. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A Scalable Content-Addressable Network. Computer Communication Review 31(4), 161–172
14. Rowstron, A., Druschel, P.: Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-peer Systems. In: Guerraoui, R. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
15. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In: SIGCOMM 2001, pp. 149–160
16. Zhao, B.Y., Huang, L., Rhea, S.C., Stribling, J., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A Global-Scale Overlay for Rapid Service Deployment. IEEE J-SAC 22(1), 41–53 (2004)
17. KaZaA.com. The Guide - The Glossary: Participation Level, Available at `http://www.kazaa.com/us/help/glossary/participation_ratio.htm`