SeAl: Managing Accesses and Data in Peer-to-Peer Sharing Networks

Nikos Ntarmos

R.A. Computer Technology Institute and
Computer Engineering and Informatics Dept.
University of Patras, Rio, Greece.
ntarmos@ceid.upatras.gr

Peter Triantafillou

R.A. Computer Technology Institute and
Computer Engineering and Informatics Dept.
University of Patras, Rio, Greece.
peter@ceid.upatras.gr

***Abstract*— We present SeAl[1], a novel data/resource and data-access management infrastructure designed for the purpose of addressing a key problem in P2P data sharing networks, namely the problem of wide-scale selfish peer behavior. Selfish behavior has been manifested and well documented and it is widely accepted that unless this is dealt with, the scalability, efficiency, and the usefulness of P2P sharing networks will be diminished. SeAl essentially consists of a monitoring/accounting subsystem, an auditing/verification subsystem, and incentive mechanisms. The monitoring subsystem facilitates the classification of peers into selfish/altruistic. The auditing/verification layer provides a shield against perjurer/slandering and colluding peers that may try to cheat the monitoring subsystem. The incentives mechanisms efectively utilize these layers so to increase the computational/networking and data resources that are available to the community. Our extensive performance results show that SeAl performs its tasks swiftly, while the overhead introduced by our accounting and auditing mechanisms in terms of response time, network, and storage overheads are very small.**

I. Introduction

The Peer-to-Peer (P2P) paradigm is becoming a new standard for architecting distributed applications, with file-sharing applications being by far the most popular among end-users. This popularity, however, raises new challenging data and resource management problems. In this realm there is no central authority for managing data and data accesses, unlike traditional (centralized or distributed) database systems, where database administrators were burdened with such tasks. The same lack of centralized control holds for the computational and networking resources as well. In P2P environments, users/peers manage their own data and their storage/computing/communication resources at will, which are a very small fraction of the total data and resources available in the system. In order for the whole system to be functioning efficiently, peers must be online and cooperating, contributing towards the achievement of the goals of system efficiency and high availability. The potentially huge numbers of peers, the resulting very large scale distribution of data and resources, and the peers' autonomous behavior make the task of managing resources, data and accesses a formidable challenge.

[1]This work is partly supported by the 6^{th} Framework Program of the EU through the Integrated Project DELIS (#001907) on Dynamically Evolving, Large-scale Information Systems.

On the other hand, so far, P2P systems mostly rely on an altruistic operational conceptualization, where peers are assumed to be willing to share content/resources with the rest of the community. In such an ideal setting, the aforementioned problems are greatly simplified. However, analysis of user behavior patterns has lead to the conclusion that, in the P2P world, selfishness is the norm [1], [2].

We consider the problem of tackling selfish behavior in a P2P sharing network, especially of the scale of modern file-sharing networks, to be (i) crucial for the performance, stability, and scalability of the system, and (ii) very challenging, due to the restrictions posed on such widely distributed systems.

We present SeAl: an infrastructure transparently weavable into (structured and unstructured) P2P sharing networks. SeAl components act in two ways; they provide the system with the necessary infrastructure to categorize peers and to allow them regulated access to resources, according to their contribution to the community; and they urge users to be altruistic, in order to build up a good reputation in the system.

II. A high-level view of SeAl

Conceptually, SeAl consists of two distinct layers: (i) the SeAl monitoring/accounting layer (*SAL*), monitoring behavior and maintaining all metadata pertinent to the peers' participation and contribution to the rest of the community, and (ii) the SeAl auditing/verification layer (*SVL*), utilizing cryptographic techniques in order to provide overwatch to the operations of the accounting layer in the presence of misbehaving users. These two layers form a substrate utilized by an incentives mechanism, which essentialy increases the shared pool of content and resources.

SeAl's counter-selfishness mechanism is based on the novel, but "natural" notion of "favors" (to be presented shortly). For simplicity reasons, we'll assume operation in the context of a file-sharing application (such as music file sharing). However, our algorithms and mechanisms are in general applicable to other classes of P2P applications, especially those dealing with large objects (e.g., digital libraries).

A. Favors

We say that a peer $n_1 \in \mathcal{N}$ ($\mathcal{N}$ being the set of all peers in SeAl), owes peer $n_2 \in \mathcal{N}$ a "favor" $f(n_1, n_2, r)$, when n_1 accesses a resource r shared by n_2 (e.g. by downloading files shared by n_2). Each node n_i, keeps a list $n_i.F_o$ of favors owed, and a list $n_i.F_d$ of favors rendered, to other peers.

Ideally, peers will contribute to the community the same amount of content/resources they take from it and the whole system will be in total equilibrium; thus, given similarly-sized resources, F_os and F_ds must be of equal size for every peer in the network (i.e. $|n_i.F_o| = |n_i.F_d|$, $\forall n_i \in \mathcal{N}$, where $|\mathcal{X}|$ denotes the cumulative size of the elements of the set $\mathcal{X}$). With this in mind, we define selfishness as a function of F_ds and F_os, using $\frac{|F_d|}{|F_o|}$ or $|F_d| - |F_o|$[2]. The higher (lower) the value, the more altruist (selfish) a peer is.

B. Basic notation and infrastructure

Independently of the underlying (structured or unstructured) P2P network, our accounting layer (SAL) deploys a Distributed Hash Table (DHT) overlay of its own to store SeAl-specific data, to achieve low hop-counts and network overhead. The DHT used may well be any of the widely used ones [3]–[6]. If the underlying system already features a DHT overlay, SeAl can leverage this additional functionality provided by its substrate, instead of deploying a separate DHT. Note that there have been approaches [7] that attempt to use a structured P2P overlay to improve performance of traditionally unstructured P2P networks – a trend already followed by many currently available peer-to-peer systems (e.g. JXTA uses Chord for its name resolution layer, while Mnet (ex-Mojonation) and LimeWire also use Chord for routing).

Every node n in SeAl has a public/secret keypair $\{n.k_p, n.k_s\}$, generated at bootstrap-time. This keypair is used to both allow SAL to identify n during transactions and to allow SVL to guarantee integrity and confidentiality of resources and messages circulating in SeAl. We assume that the public keys of nodes in SeAl are directly accessible by all peers in the system. This functionality may be provided by a centralized public-key infrastructure (PKI), or in a distributed manner (e.g. a la SDSI/SPKI [8], or the PGP web of trust), or through the DHT deployed by the accounting layer, in a way similar to that described in [9]. In any case, we believe this kind of functionality to be orthogonal to the operation of our accounting layer.

We assume that resources, and (join/search/retrieval) transactions in SeAl are identified by unique IDs (e.g. 160-bit UUIDs [10]). Node IDs are a special case of unique identifiers; a node in SeAl is uniquely identified by the hash of its public key. This allows for pseudonymous operation of nodes in SeAl: (i) the identification of nodes is decoupled from their actual IP address or other link-level information, (ii) as long as a node uses the same keypair, its ID remains the same, and (iii) nodes may deny responsibility for certain actions by simply generating a new keypair and discarding the old one (although this would lose them their reputation). Furthermore, we want to prevent nodes (a la Freenet [11] and Achord [12]) from arbitrarily choosing their IDs (and thus their exact position in the DHT network overlay). Thus, in order for a node to be allocated a certain ID, it must prove knowledge of the corresponding secret key. For example, in a Chord-like system, the node's predecessor could ask the node to sign some piece of information using its secret key, considering that forging a signature is computationally infeasible.

[2]A more elaborate metric of altruism may well be used instead of the ratio or difference of the sizes of a peer's favor lists; however, we shall use these metrics for the rest of the paper, and leave more complex forumalae as a subject of future work.

III. The SeAl Monitoring / Accounting Layer

We shall first discuss the properties of SAL – the SeAl Monitoring/Accounting Layer. For now, we confine our discussion to tackling only selfish behaviors, where users may be either fully contributing their data and resources to the P2P community (altruistic) or keeping some or all of their files and resources off-line (selfish). We assume that peers in the system do not attempt to subvert the system protocols and mechanisms. We shall deal with such malicious behaviors later, introducing the SeAl Verification Layer.

A. Transaction receipts and favors

Every (retrieval) transaction in SeAl terminates with both of the participating parties having a digital "receipt" of the transaction, called a Transaction Receipt (or TR). TRs are created as the concatenation of the IDs of the sender (server) and receiver (client) nodes participating in a transaction, plus information about the resource exchanged (e.g. its ID, size, and a checksum), plus a timestamp. We denote by $TR(n_1.id, n_2.id,\ r.id, t)$ a Transaction Receipt concerning resource r being sent from n_2 to n_1 at time t. Favors in SeAl are implemented using TRs. Thus an entry in the F_o list of node n_1, about a transaction with a node n_2, concerning a resource r, is of the form: $\{n_2.id, r.id, t, TR(\ldots)\}$.

Since in the realm of file-sharing P2P applications, the main bottleneck and the main resource consumed by peers is bandwidth, TRs contain the size of the exchanged information (in the resource-specific information), as an indication of how-big-a-favor does the requesting node owe to the serving one. Thus, TRs are not mere tokens of esteem, but also have a quantitative nature – a TR is assumed to have a value of $TR.r.size \times \frac{TR.t}{current\ time}$, with time expressed relative to an Epoch a la SVr4 and *BSD). Timestamps are further used to allow for the implementation of an aging mechanism (left as a subject of future work) in order to (i) keep the memory/storage requirements low, and (ii) adapt to transitivity of node behavior. With these in mind, we can now see that the above mentioned scoring schemes (i.e. $\frac{|F_d|}{|F_o|}$ or $|F_d| - |F_o|$), although simplistic, manage to capture the degree at which a peer contributes or takes advantage of the community.

B. Favor Payback (Enforced!)

Assume that node n_1 accesses resource r shared by node n_2, and therefore n_1 owes n_2 a favor (i.e. $f(n_1, n_2, r) \in$

$\{n_2.F_d, n_1.F_o\}$). Peer n_2 can then use this favor as follows: the next time another peer n_3 will contact n_2 in order to download a file, n_2 can choose to redirect the request to one of the peers which owe it a favor – e.g. preferably n_1, if the resource wanted is r – and have n_3 report back its (dis)satisfaction.

In general, redirect targets are chosen among the peers appearing in a node's F_d, based on various heuristics, such as the history of transactions with that peer (e.g. what files the target peer has downloaded from the source peer), the processing/network capabilities of the target peer (e.g. as measured by PING messages), the reputation of the target peer, or some heuristics binding all of the above together (the actual selection algorithm remains a subject for future work). All three peers keep track of this transaction by adding or updating the appropriate entries in their favor lists; n_3 owes a favor to n_1 (i.e. $f(n_3, n_1, r) \in \{n_1.F_d, n_3.F_o\}$), while both n_1 and n_2 mark the corresponding favors as paid-back.

Furthermore, given the selfishness/altruism definition presented earlier, peers in SeAl keep track of their "altruism score" – $n_i.\mathcal{A}$. The actual formula used is node-dependent; the administrator of each node autonomously sets her own desired value to $n_i.\mathcal{A} = \frac{|F_d|}{|F_o|}$ or $|F_d| - |F_o|$. Moreover, node administrators individually set an upper and a lower threshold for this score, denoted $n_i.\mathcal{A}_{max}$ and $n_i.\mathcal{A}_{min}$ respectively. Under this scheme, nodes choose to redirect or not an incoming request, based on their current score and their corresponding thresholds; always redirect when over the upper limit, never when under the lower limit, and probabilistically decide (with probability $\mathcal{P}(\mathcal{R})$) when in between.

Requests may be recursively redirected, up to a maximum (tunable) hop-count, thus creating chains of nodes (cycles are avoided using the ID of the request). In this case, only the last pair of nodes in the chain mark the corresponding favors as paid-back. The intuition is that the cost of redirecting an incoming request is negligible, compared to the cost of actually serving it; thus only the last node in the chain actually pays back the corresponding favor.

The favor payback mechanism outlined above has some rather interesting characteristics. First, it makes no assumptions on the distribution of popularities or on the degree of replication of the objects shared in the system; the probability of having two or more nodes sharing the same file grows with the popularity of this file. This means that the more popular a file is, the more it is replicated, as a sheer consequence of its popularity. Second, the probability that some of the nodes sharing a common subset of files owe favors to each other, grows with both the popularity of the shared files and the altruism score threshold values set by the nodes in the system.

C. Bad Reputation – The "black lists"

Any deviation from the normal behavior may trigger the "black-listing" of the corresponding peer. n_2 can then use its $f()$ entry (from its favor list) to "black-list" n_1 as not being a legitimate peer (a second-chance algorithm may also be used instead of the strict one described here). Note that being offline is viewed as selfish behavior, since the peer is not contributing to the community at the moment. Thus, when node n_2 wishes to blacklist node n_1, it creates a blacklisting request (BLR) and publishes it in the DHT overlay. A BLR is merely the TR of the transaction on which the blacklisting is based (with the ID of the black-listed (n_1) peer in the TR being hashed again), published on the DHT using an ID of $\mathcal{H}(\text{"}BLR\text{"}||\mathcal{H}(n_1.id))$ – $\mathcal{H}(\cdot)$ denoting a secure hash function. Thus, a node will be responsible of a BLR concerning itself with negligible probability, while the node storing a certain BLR can't know whom does it blacklist, due to the use of secure hashing for the black-listed peer's ID. The recipient of any such message can choose to either give n_1 a second chance to be good (with probability $\mathcal{P}(\mathcal{SC})$), or black-list n_1 (i.e. store the BLR) immediately (with probability $1-\mathcal{P}(\mathcal{SC})$).

Note that the use and publishing of BLRs poses no scalability restrictions on our system. We treat such objects much like we would treat any other object on a DHT (except, of course, from SVL's verification chores). Thus, storing and retrieving a (set of) BLR(s), needs as many hops and time as the underlying DHT (e.g. $O(\log N)$ hops for Chord). Fault tolerance and availability concerns are left on the DHT, although other approaches could be taken.

D. Request scoring – the "white lists"

Peers probe the DHT overlay periodically to discover any black-list records filed against them. Thus, they become aware of their status within the P2P community. When a node n_1 contacts a node n_2 for a retrieval request of resource r, n_2 asks n_1 to select a small fraction W_m of its F_d favor-list entries (called the "white records") whose sum of values is less than some threshold value localy defined by n_2, and to present this sum to n_2. Remember that, at this stage, we do not deal with malicious behaviors.

This leads to the following scoring scheme: the incoming request is assigned a preliminary positive score s_w, equal to the sum of the values of the selected white records. Subsequently, the sum of the values of the node's "black records" (s_b), up to the same threshold defined in the first step above, plus the size of the requested resource ($r.size$), are subtracted from the request's score, for a final score of $s_w - s_b - r.size$. Note that the serving node may well choose 0 as the threshold value for the "white records" presented by the requesting peer, thus practically ignoring the favor list entries altogether and scoring the incoming request on a per-size basis.

E. Request serving – the incentives

Incoming requests are not served directly but rather put in a waiting queue, sorted using the requests' scores (ties are broken using the requests' arrival time, in a FIFO order). Scheduling is hence based on the peers' reputation. Based on local decisions of the administrator of each node, requests from low-reputation peers may be either scheduled for processing for when the serving node will be idle, allocated limited resources (e.g. connection throttling, bandwidth throttling etc.), or even completely rejected, thus introducing a first slider –

the "scheduling" slider. Furthermore, since a request's position in the queue may change with the addition (or bailing-out) of higher-scored requests, served peers may probe the serving ones for their position in their queues (or for the amount of resources allocated to them).

This scheduling strategy (called the "feedback" mechanism) gives peers *incentives* to act altruistically, since highly-reputed peers will be served better or ahead of others. Moreover, we argue that, since reputation is of high importance for the users' satisfaction, users will have incentives to not mount Sybil[3] [13] attacks. Note that this scheme allows for the exploitation of such "positive externalities"[4] as underused resources, since even lowly reputed nodes will (eventually) be served.

F. Debt Payback

As already discussed, peers eventually become aware of their reputation in the system. This gives a black-listed peer the ability to make-up for its past selfish behavior; it can choose (some of) its black records and offer to pay back the corresponding favor, by contacting the node who blacklisted it. If everything works out well, the formerly black-listed node will end-up with a set of Transaction Receipts, denoting that it has paid-back its debt to the community, and can subsequently ask the node storing the relevant BLRs to remove them from the network. Naturally, every node can be blacklisted only once for every favor it owes and hasn't yet paid back.

IV. The SeAl Verification Layer

In real-world conditions, many "selfish" users will try to subvert the system operation in order to bypass the monitoring/accounting mechanism. Enter SVL – the SeAl Verification Layer. Due to space considerations, we'll simply outline the points in which SVL interferes with the operation of SAL.

A. Transaction receipts revisited

With malicious users in the scene, TRs must also provide both data origin checking, and uniqueness and timeliness guarantees – a type of "transaction authentication", as defined in [14]. Thus, we construct TRs as in the SAL case, only that now the TR information is also signed, first by the receiver node and then by the sender node. Any third party can verify the validity of a transaction receipt, by verifying the signatures of the serving and served peers and checking that both key hashes are present in the receipt. If the verifier wishes, she can further ask n_2 for the hash of (parts of) r, or even access, through a normal retrieval request (thus being charged with appropriate favors), (random blocks of) the resource for which this receipt was issued, and verify that the resource-specific information in the receipt is valid.

[3]A Sybil attack is a situation in which a peer connects to the system with a different ID, in order to get rid of possibly negative feedback.

[4]A positive externality is a benefi t to the community that results from peers acting in their own self-interest.

B. Blacklists revisited

With SVL in place nodes can't forge BLRs since such an operation is equivalent to forging digital signatures. Moreover, when a blacklist record is sent out on the overlay, a node receiving or retrieving a BLR, according to local decisions at this node, verifies its validity with a probability $\mathcal{P}(\mathcal{V})$. Furthermore, nodes periodically check for the validity of black records of random other nodes.

The verification of BLRs is done by asking the black-listed (and, optionally, the black-listing) peer. Peers can prove the (in)validity of such TRs by presenting the appropriate favor list entries, denoting the corresponding favors as paid-back. In this case, the legitimate peers may initiate the blacklisting of the "perjurers". Thus, both selfish and malicious peers are identified and, eventually, blacklisted. The usage of digital signatures and their verification guarantee that no malicious blacklisting will go by unnoticed or not penalized. Black records deemed invalid by both the serving and served nodes don't trigger any further black-listing; they are only marked as invalid and linger in the system – for some user-specified amount of time – as an indication of the nodes' past behavior. The above probabilistic scheme presents us with a "black-listing strictness" slider, creating a trade-off between system performance (bandwidth requirements, CPU usage, etc.), and responsiveness to selfish behavior.

C. Whitelists revisited

Assume again that a node n_1 contacts a node n_2 for a retrieval request of resource r. With SVL in place, the requesting node again chooses a subset of its white records (with sum of values less than a threshold defined by n_2), but now forwards them along with its request to n_2. Node n_2 can then choose to either serve n_1's request (optimistic scenario), or to check whether n_1 is a legitimate peer, by fetching and verifying one or more of the presented "favors" (following $\mathcal{P}(\mathcal{C})$). Assume that $W'_m \subset W_m$ favors are fetched and verified. Then the incoming request is assigned a preliminary positive score s_w, just like in the SAL case, while the remaining $W_m - W'_m$ non-verified favors are assumed to be of (time-normalized) size equal to the size of the requested resource.

Furthermore, n_2 can search the overlay for BLRs filed against n_1; n_2 may ask the node(s) responsible for holding n_1's black records for TRs whose time-normalized score s'_b amounts to less than or equal to the minimum of s_w and the white-list threshold defined in the first step above. The black-records returned are verified (following $\mathcal{P}(\mathcal{V})$), and a negative score s_b is computed. Finally, the request is assigned an overall score equal to $s_w - s_b - r.size$.

The above probabilistic scheme introduces a trade-off between the extra network accesses, and possibly fake list entries. We can imagine a "white-listing strictness" slider, where the administrator of each node locally decides on the level of confidence put on every favor list entry presented to it. This scheme has the very important property that it bounds the network overhead due to transfers of black records. Moreover, since we (probabilistically) verify white and black records,

there is now no need for a peer paying back some of its black records to publish the corresponding TRs to the DHT; the next time some node will try to verify the validity of any of the corresponding black records, the test will fail and the black record will be marked as invalid. Also remember that no further blacklisting is performed for black records marked invalid during delayed pay-back, since both peers engaged in the corresponding transaction will verify that the relevant favor has indeed been paid-back.

The above scheme provides strong disincentives, albeit not a complete solution, for misbehaving users. Such users may mount a Sybil attack (rejoin the network with a new ID) to rid themselves of their black records, or collude with others in order to create phony white records. A Sybil attack is made undesirable since it would result obviously in the loss of any white records gathered so far by the user. Further, a collusion attack is made undesirable since, regardless of how many (even phony) white records a selfish peer may have, it can only achieve a maximum request score equal to the one it would get if it were a newcomer, if there are enough black records filed against it. The worst-case scenario is when a node concurrently performs both types of attacks: it mounts a Sybil-attack (to get rid of black records) and colludes with other malicious peers (or multiple digital personas of the same node) to gather white-records as a newcomer. Note, however, that the amount of useable white-records is bounded by the threshold value defined by the serving node.

Ultimately, the problem of collusion in P2P networks remains an open issue, even for purely trust-related systems. We can prove that such "malicious" behavior can't be efficiently or realistically dealt with in a widely distributed environment, unless we use some centralized authority (e.g. to use a trust-based system, a la Advogato [15], where colluding peers can't gain high trust values outside the set of the peers engaged in the collusion), or we make outrageous assumptions on the degree of coordination of peers (a similar claim has been made by [13]). The modular, layered design of SeAl allows us to incorporate any such functionality, as soon as better methods and systems become stable and available.

D. File Transfer

With SVL, the file transfers play a key role in establishing the validity of TRs circulating in the system. The file transfer protocol is outlined in alg. 1. Briefly, assume that n_1 accesses resource r shared by n_2. When the time has come for the request to be served, n_2 generates two random symmetric-cipher keys k_1 and k_2, and uses k_1 to encrypt r. It then encrypts k_1 using k_2 and sends the encrypted resource and encrypted k_1 to n_1. n_1 constructs a preliminary TR, signs it, and sends it to n_2. n_2 verifies the signature in the TR, and signs the resulting TR itself. It also encrypts k_2 using $n_1.k_p$ and sends it and the TR to n_1. n_1 recovers the symmetric-cipher key and decrypts r. Finally, both n_1 and n_2 add the final TR to their corresponding favor list. Note that (i) we use symmetric ciphers to encrypt the transfered file, so to achieve a high bit-rate, and (ii) the transaction-specific keys need not be kept after the transaction ends; the keypair of each node is enough to do the verification of black and white records. We consider the bandwidth to be the main bottleneck and the more scarce resource in current file-sharing P2P applications. One could argue that n_2 could bail out of the algorithm after step 6, thus ending up with a valid TR, while leaving n_1 with a pile of useless random bytes. The intuition behind the file-transfer protocol is that, since a node has spent some of its resources to send the actual byte stream, it has already given its share to the common good. Sticking to the protocol is then trivial and merely adds to the good reputation of the node itself (since a TR that can't be verified is useless).

Algorithm 1 File Transfer. Algorithm runs on $m.n_{server}$, unlessstated otherwise.

Require:
$send(msg, node\ ID)$: Send msg to node with given ID.
$\mathcal{E}_k(\alpha)$: Encrypt α using key k.
$\mathcal{S}_k(\alpha)$: Sign α using key k.

process(Msg m **)**

1: Generate k_1, k_2 = random symmetric-cipher keys;
2: $r_e = \mathcal{E}_{k_1}(r)$; $k_1' = \mathcal{E}_{k_2}(k_1)$;
3: $send(\{r_e, k_1'\}, m.n_{client}.id)$;
4: $m.n_{client}$:
  4.1: construct $TR' = \{m.n_{server}.id, m.n_{client}.id, r.id, t\}$;
  4.2: $TR_s' = \mathcal{S}_{m.n_{client}.k_s}(TR')$;
  4.3: $send(TR_s', m.n_{server}.id)$;
5: Verify the signature in TR_s';
6: $TR_s = \mathcal{S}_{m.n_{server}.k_s}(TR_s')$;
7: $send(\{TR_s, \mathcal{E}_{m.n_{client}.k_p}(k_2), m.n_{client}.id\})$;
8: $m.n_{client}$: recover k_2 and k_1 and decrypt r;
9: $F_d.add(TR_s)$; $m.n_{client}$: $F_d.add(TR_s)$;

V. Experimentation and Performance Results

Our experimental setup assumes that SeAl operates in a music-file sharing context, with file sizes (in Mbytes) uniformly distributed in the range 3-10 (for an average size of 6.5 Mbytes). The simulated network consists of 2048 nodes, sharing 50,000 distinct documents, replicated across peers following a Zipf access distribution, with α = 0.7 and 1.2 [16] (for a total of approximately 50,200 and 51,350 documents respectively). We have also tested our system with larger node populations (with similar results), but with not as many queries, due to CPU and memory constraints, and thus report only on the 2048-node case here. The simulation runs for 1,000,000 requests. Requests arrive at the system following a Poisson distribution, such that every peer will make approximately 5 requests per day of simulated time. The documents requested follow a Zipf distribution too, with similar results for both α values (0.7 and 1.2). Due to space considerations, we report only on the α = 1.2 cases.

The peer population consists of 90% (70%) free-riders and 10% (30%) altruists, with network connections ranging (uniformly) from 33.6kbps (modem) to 256kbps (cable) lines for selfish peers, and from 256kbps to 2Mbps (T1) lines for altruists. Furthermore, all peers may fail or deny service with a probability of 0.2, and delete/unshare files with a probability of 0.1. Finding a peer sharing a file and downloading the file are both 1-hop operations. All SeAl operations are run

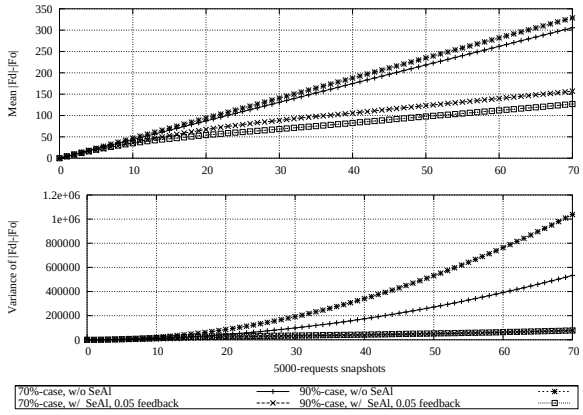


Fig. 1. Convergence

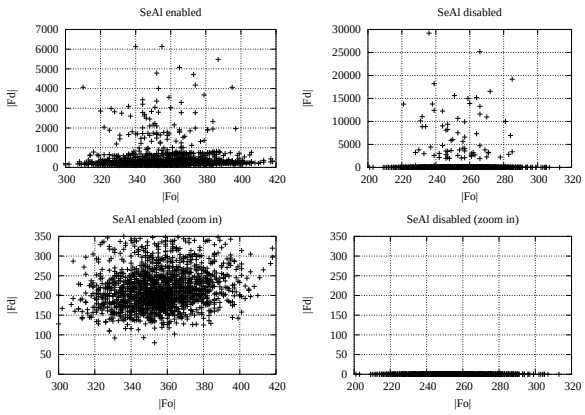


Fig. 2. |Fd| vs. |Fo| (90% selfi sh users)

on top of a DHT, thus every SeAl transfer is assumed to take $O(log(N)) = 11$ hops on average (for the 2048-node network). As we'll show shortly, in spite of this handicap, SeAl incurs negligible network overhead. Moreover, should SeAl be operating on top of store-and-forward networks, such as FreeNet or AChord, the observed network and storage overhead would be orders of magnitudes smaller. Peers compute their scores using $|F_d| - |F_o|$. Altruistic (selfish) peers redirect incoming requests with probabilities 0, 1, and 0.5, when their score is below their lower threshold, above their upper threshold, or within these values respectively. Moreover, the aging mechanism for the black- and white- records is off in the simulation.

Since the incentives given by our mechanism are rather based on the user experience, we tried to also model the user behavior. Thus, every peer is characterized by the probabilities:

- $\mathcal{P}(\mathcal{R}_{a,s})$: the probability for a node to remain altruist/selfish (initially set to 0.8 and 1.0 respectively). Used on a per-request basis to model transient user behavior. In the case of altruistic peers, we can also think of $\mathcal{P}(\mathcal{R}_a)$ as the probability of node failures.
- $\mathcal{P}(\mathcal{E}_f)$: the probability for a node to erase a file (set to 0.1). Used on a per-download basis.
- $\mathcal{P}(\mathcal{C}_a)$: the probability of a node to abort a transfer (set to 0.1). Used on a per-request basis.

Every time a request is to be enqueued, the serving node informs the requesting node of the expected waiting time, while probes are periodically sent to the serving nodes for the current (expected) waiting time every 2 minutes, and every user/peer has an upper threshold of 20 minutes of waiting time. When the expected waiting time of a request is above this value, the request is dropped and the user decides (with probability $\mathcal{P}(\mathcal{S}_d)$) to become a "better" user; users get better by (probabilistically) increasing their $\mathcal{P}(\mathcal{R}_a)$ and decreasing their $\mathcal{P}(\mathcal{R}_s)$, by an amount of SD. We have run experiments with the following values for $\mathcal{P}(\mathcal{S}_d)$ and SD respectively: $\{0.0, N/A\}$ (aka no feedback), $\{0.5, 0.05\}$ (aka small feedback), and $\{0.5, 0.2\}$ (aka medium feedback).

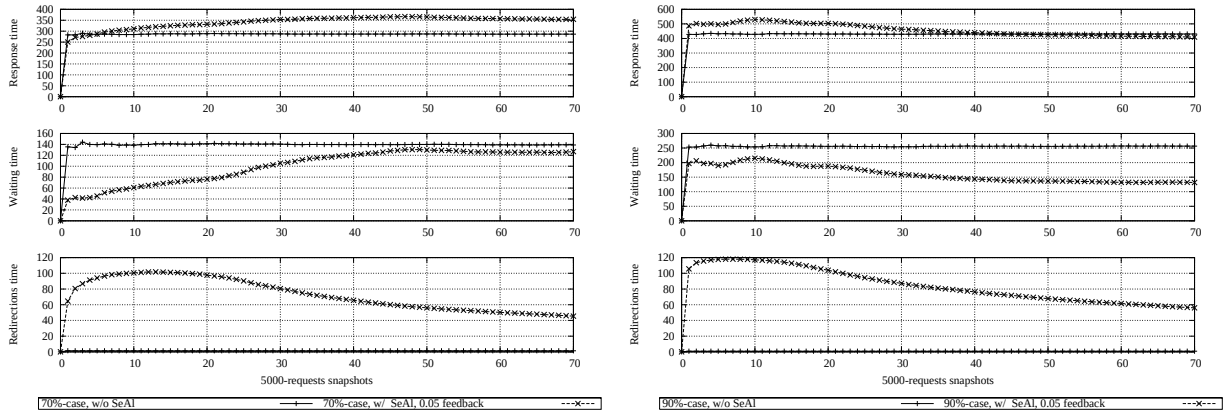
Discussion

We have measured the average altruism score in the system (using $|F_d| - |F_o|$). In fig. 1 we have plotted the mean value and the variance for $|F_d| - |F_o|$. As can be easily seen, the SeAl-enabled simulations scale much better as time passes by.

In fig. 2 we have plotted the $|F_d|$s versus the $|F_o|$s of all 2048 peers in the system, after all 1,000,000 requests, for the 90%-selfish case (SeAl using 0.05 feedback). This figure can help us understand the real advantages of SeAl. If we observe carefully the left subfigures, we'll see that peers are divided in two main clusters; the altruists, having higher $|F_d|$s than $|F_o|$s and thus occupying the upper part of the figures, and the selfish peers, occupying the lower part of the figures. The effects of SeAl are twofold: (i) the altruistic part of the figures is more dense around its average coordinates – yet another evidence of the improved stability and convergence of the SeAl-enabled system versus the classic, non-SeAl system; and (ii) if we zoom in to the lower (selfish-occupied) part of the figures, we'll see that, while in the non-SeAl system all selfish peers have 0 $|F_d|$'s, in the SeAl-enabled case a good fraction of these peers has moved closer to the equilibrium point ($|F_d| = |F_o|$).

The network overhead incurred by SeAl is negligible, averaging only 0.4% of the total traffic. As far as the storage overhead is concerned, even with the aging mechanism turned off in the experiments, SeAl was found to incur on average a mere 0.53% overhead in storage space requirements (approximately 780 kbytes per peer) after 1,000,000 requests, or an average 0.8-bytes per peer per request! Moreover, should SeAl be operating on top of store-and-forward networks, such as FreeNet or AChord, the observed network and storage overhead would be orders of magnitudes smaller.

A rather interesting result can be seen in the time-overhead figure for the 90%-case (fig. 3(b)), where the lower subfigure shows the time taken until a peer is found to serve a request, the middle subfigure shows the time lost waiting in queues, and the higher subfigure shows the total response time. Remember that the non-SeAl system pays no overhead for finding serving nodes (depicted by the near-zero curves in the lower subfigure), as opposed to the 50" for SeAl, caused by (failed)



(a) 70% selfish user population

(b) 90% selfish user population

Fig. 3. Redirection, Waiting, and Response Times (in secs)

redirections. Note, however, that SeAl makes up for this initial handicap in the total response time subfigure, averaging 20" lower than the non-SeAl system.

This can be explained as follows; there are four main factors that influence the response time: (i) the number of (possibly failed) redirections (e.g. due to redirection to selfish/offline nodes), (ii) the scheduling at the waiting queue of every peer (resulting in an SJF-like schedule in SeAl, since higher-reputed nodes are more likely to have faster connections than lower-reputed nodes), (iii) the average length of these waiting queues, and (iv) the feedback mechanism (which results in more peers serving requests, thus lowering the average queue length).

Obviously, SeAl benefits from (ii), (iii) and (iv) but not from (i). In the 90%-selfish case (fig. 3(b)), (i) is overshadowed by (ii) and (iv), while in the non-SeAl case only 10% of the peers actually serve requests, thus having huge waiting queues. In the 70%-selfish case (fig. 3(a)), however, the effect of (iii) and (iv) is lighter, due to the increased number (by a factor of 3x; 30% vs. 10%) of altruists, hence the smaller improvement. Note that selfish, low-bandwidth nodes starting to behave better (i.e. sharing a file once in a while) may also cause an increase in the overall response time; while in the non-SeAl system all requests are served by the altruistic, high-bandwidth nodes, when SeAl and its feedback mechanism kick in, a fraction of the downloads are served by these low-bandwidth nodes, thus further raising the overall service time.

What we face here is a fundamental trade-off between fairness in the load distribution and efficient request serving. Our algorithm adds an average extra 20% to the overall response time in the 70%-selfish case (fig. 3(a)), but leads to a fairer load distribution (as shown in fig. 1 and 2). On the other hand, when altruists are more scarce in the system (as is the case in the 90%-selfish population runs – fig. 3(b)), the sheer fact of the more balanced load suffices to make SeAl the winning configuration.

Note that a large part of the time-domain overhead is due to the auditing/verification chores of SVL; in a system with a peer population consisting of more altruists and less selfish peers, we may only need to use the monitoring/accounting capabilities of SAL and leave aside the expensive cryptographic mechanisms of SVL. Alternatively, we may tune our black- and white-listing "sliders" so that the probability with which we engage in the verification transactions is low enough to achieve a better/lower response time overhead. In any case, we believe that even the 20% overhead of the 70%-selfish case is acceptable, given the achieved fairness of the load distribution.

VI. Related Work

The problems of "trust", "reputation", and "accountability" in distributed systems have been a hot-spot for quite some time. However, widely deployed, web-scale, data sharing systems that maintain metadata about participating nodes [17], [18] don't address the problems posed by selfish and malicious peers. Payment schemes have been suggested by several researchers for enforcing fair-sharing. However, to the authors knowledge, all existing payment schemes [19]–[22] make the assumption of the existence of a globally trusted (centralized) entity at their core. Cornelli et al. [23], present a system to select the most reputable peer to download content from, a limited functionality compared to SeAl. GNUnet [24] like SeAl, allows for the exploitation of "excess" resources. However, SeAl uses a much stronger fair-play enforcement scheme, with TRs and black-lists, while also being completely decentralized. FreeHaven [25] is a client-server-based system where only servers perform trust computation, not applicable to a true peer-to-peer sharing network. [26] have peers keep "usage files", similar to our favor lists, and other peers "audit" both their neighbors and random nodes at random intervals. Their problem is trade-oriented: nodes want to store their data on other nodes. Furthermore, they use a quota-based reasoning; "under quota" peers are excluded from the network. This leaves no space for the exploitation of excessive network/processing resources.

A work very close to ours is that of Aberer et al. [27]. [27] also use a DHT-type overlay (i.e. P-Grid) to store transaction information. Compared to their work, SeAl: (i) uses both black- and white-lists, as opposed to only black-lists ("complaints"), thus being less prone to Sybil attacks; (ii) features queueing and scheduling algorithms (built on top of our monitoring layer) unique to our setting, providing incentives for altruistism; (iii) uses random checks and different levels of verification checks for efficiency, while the verification chores can be left aside or tuned at will to achieve better response-time efficiency; (iv) allows for inconsistent malicious/selfish behavior, different forms of punishment, and for the exploitation of excess resources ([27] use binary logic as to the behavior exhibited by peers – a peer is either completely selfish or completely altruist – and, based on this, are allowed either full or no access to resources); (v)features an aging mechanism further addressing temporal inconsistencies in user behavior.

Apart from the individual differences between this and related work, the main overall differences are in comprehensiveness and philosophy. Specifically, SeAl:

1) contributes a comprehensive system design, identifying selfish/altruistic peers, with the capability of dealing with slandering and colluding peers, and increasing the total number of shared data and resources.
2) is based on a comprehensive performance analysis of the (i) (storage, network bandwidth, and response time) overheads, and (ii) the positive effects of SeAl.
3) is weavable in both structured and unstructured P2P overlays, not bound to any underlying topology. This fact bears major implications on the system's usability and applicability.
4) allows for the exploitation of excess resources and may employ various levels of punishment (from simple blacklisting, to ousting nodes from the network) exploiting its very capability to categorize nodes into various types.

VII. Conclusions

We have presented SeAl, a novel infrastructure that addresses a key problem in P2P data sharing networks, namely the problem of wide-scale selfish behavior. Toward this goal SeAl offers (i) definitions/metrics of selfishness/altruism, (ii) subsystems performing monitoring/accounting and verification/auditing functionalities that enable the efficient, reliable, auditable identification of selfish peers, and (iii) accompanying incentive-offering mechanisms, while (i) respecting the autonomy of each peer to define his own selfishness/altruism levels and (ii) allowing for the exploitation of positive externalities (in the form of excess resources), which abound in P2P networks. Furthermore, depending on the environment in which SeAl is to be deployed, the modular architecture of SeAl permits the use of just SAL's monitoring/accounting mechanisms, if we do not want to counter or do not expect to face malicious behavior, or to also use the extra security encompassed by SVL's cryptographic verification mechanisms.

SeAl forms a complete infrastructure software layer that is weavable in both structured and unstructured P2P networks, making it thus usable in any existing or future P2P infrastructure. Our implementation and extensive performance testing of SeAl shows that SeAl achieves its identification goals swiftly. At the same time, the network, storage, and response time overheads imposed by SeAl have been measured to be very small, if any. SeAl can be the basis for the development of a wide variety of services in P2P data networks. For example, the ability to discover altruistic/powerful peers can play a key role in the derivation of better (more reliable and faster) network architectures and in the optimization of queries in p2p data networks. This is a major focus point for our future work.

References

[1] E. Adar and B. Huberman, "Free riding on Gnutella," Xerox PARC, Tech. Rep., 2000.
[2] S. Saroiu, P. Gummadi, and S. Gribble, "A measurement study of peer-to-peer file sharing systems," in *Proc. MMCN '02*.
[3] P. Druschel and A. Rowstron, "Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM Middleware '01*.
[4] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan, "Chord: A scalable Peer-To-Peer lookup service for internet applications," in *Proc. ACM SIGCOMM '01*.
[5] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. ACM SIGCOMM '01*.
[6] P. Maymouknov and D. Mazières, "Kademlia: A peer-to-peer information system based on the XOR metric." in *Proc. IPTPS '02*.
[7] M. Castro, M. Costa, and A. Rowstron, "Should we build gnutella on a structured overlay?" in *Proc. HotNets II '03*.
[8] IETF, "SPKI Working Group," http://www.ietf.org/ html.charters/spki-charter.html.
[9] K. Aberer, A. Datta, and M. Hauswirth, "A decentralized public key infrastructure for customer-to-customer e-commerce," *International Journal of Business Process Integration and Management*, to be published.
[10] M. Mealling, P. Leach, and R. Salz, *A UUID URN Namespace*, Network Working Group – IETF, October 2002.
[11] Clarke, I., et al., "Freenet: A distributed anonymous information storage and retrieval system," in *Proc. PET '00*.
[12] S. Hazel and B. Wiley, "Achord: a variant of the Chord lookup service for use in censorship resistant Peer-to-Peer publishing systems." in *Proc. IPTPS '02*.
[13] J. Douceur, "The Sybil attack," in *Proc. IPTPS '02*.
[14] A. Menejes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1996.
[15] Advogato, http://www.advogato.org/.
[16] K. Sripanidkulchai, "The popularity of gnutella queries and its implications on scalability," white paper, Feb. 2001.
[17] Gnutella, http://gnutella.wego.com/.
[18] FastTrack, http://www.fasttrack.nu/.
[19] R. Rivest and A. Shamir, "PayWord and MicroMint: Two simple micropayment schemes," in *Proc. Security Protocols Workshop '96*.
[20] D. Chaum, A. Fiat, and M. Naor, "Untraceable electronic cash," in *Proc. CRYPTO '88*.
[21] P. Golle, K. Leyton-Brown, and I. Mironov, "Incentives for sharing in peer-to-peer networks," in *Proc. ACM EC '01*.
[22] Mojonation, http://www.mojonation.com/.
[23] F. Cornelli et al., "Implementing a reputation-aware Gnutella servent," in *Proc. Networking Workshops '02*.
[24] C. Ghrothoff, "An excess-based economic model for resource allocation in peer-to-peer networks," *Wirtschafts Informatik*, March 2003.
[25] R. Dingledine, M. Freedman, and D. Molnar, "The Free Haven Project: Distributed anonymous storage service," in *Proc. PET '00*.
[26] T. Ngan, D. Wallach, and P. Druschel, "Enforcing fair sharing of peer-to-peer resources," in *Proc. IPTPS '03*.
[27] K. Aberer and Z. Despotovic, "Managing trust in a peer-to-peer information system," in *Proc. CIKM '01*.