AESOP: Altruism-Endowed Self-Organizing Peers*

Nikos Ntarmos and Peter Triantafillou

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

Abstract. We argue the case for a new paradigm for architecting structured P2P overlay networks, coined AESOP. AESOP consists of 3 layers: (i) an architecture, PLANES, that ensures significant performance speedups, assuming knowledge of altruistic peers; (ii) an accounting/auditing layer, AltSeAl, that identifies and validates altruistic peers; and (iii) SeAledPLANES, a layer that facilitates the coordination/collaboration of the previous two components. We briefly present these components along with experimental and analytical data of the promised significant performance gains and the related overhead. In light of these very encouraging results, we put this three-layer architecture paradigm forth as the way to structure the P2P overlay networks of the future.

1 Introduction

P2P networks have recently been receiving an everincreasing recognition and attention. Given the desirable characteristics and the inherent promises of the peer-to-peer computing paradigm, attention is likely to increase in the coming years. Within the related research a fundamental problem of "routing" (lookup) in P2P networks has received special attention; given a request arriving at a specific peer node, for a document id, route the request to the peer in the network, which stores the document.

Recent research in P2P networks has largely focused on structuring efforts for the network overlay so to ensure the fast routing of requests to the peers storing the requested objects. Most prominent in these efforts is a class of P2P network overlays based on Distributed Hash Tables (DHTs). DHT-based solutions [1–4] can provide routing in the steady-state case in $O(logN)$ hops, in a network of N peers, providing routing efficiency and scalability. These performance characteristics constituted an important step forward, compared to the original, pioneering attempts for "unstructured" P2P networks [5, 6]. In unstructured networks related overheads are much higher and no guarantees can be given for the efficiency or the locating of the requested documents.

Currently, there are efforts underway for bridging this structure chasm: unstructured networks are being enriched with DHTs (e.g., Limewire[7] and Mnet[8] (ex-Mojonation[9]) with Chord[2]) and structured networks aim to deal effectively with the operational characteristics found in applications built on top of unstructured networks (e.g., Structella [10]).

* 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.

1.1 Our Perspectives and Position

Real-world Perspective Our starting position is that our world consists of altruists, selfish people, and of others with behavior ranging in between, with a non-negligible percentage of the last category showing altruistic behavior if given the incentives to do so. Within the world of P2P networks this fact has been clearly manifested and documented: take as example the 'free riders'[11] phenomenon, first measured in the Gnutella network.

The bad news is that the great majority of Gnutella peers were proven to be free riders (more than 70%). And this is indeed very bad news for DHT-style overlays, since the great majority of peers may be joining the network and leaving very soon thereafter – [12] have show that the median session duration in Gnutella and Napster is approximately 60 minutes! There also exist other independent reports offering evidence that half of the peer population is changing almost every half an hour[13].

The good news are that a non-negligible percentage of the peers were proven to be altruistic[1]. In Mojonation[9], more than 1-2% of all users, stayed connected almost all the time. In Gnutella, 1% (10%) of peers served about 40% (90%) of the total requests [11], while [12] have shown that the longer a node has been up, the more likely it is to remain up. Note that the routing algorithm in Gnutella acts as a counter-incentive to acting altruistically, flooding peers with requests. Also, the majority of Mojo Nation users were dissatisfied with it and that is why they permanently disconnected. Thus, we conjecture that, by giving incentives (to avoid the so-called tragedy of the commons) and taking away such counter-incentives, more network nodes will be willing to act altruistically.

Research Perspective Looking at related research in DHT-structured P2P networks, one notices that given a highly-dynamic environment, routing performance degrades to $O(N)$ hops (that is, if the network remains connected). Fundamentally, this is due to the difficulty in keeping up with the required updates to routing state for special neighbors which ensure $O(logN)$ hops in the steady-state case. Much to their credit, the authors in [14] studied how to guarantee in highly-dynamic cases $O(logN)$ routing performance. To do this, $O(log^2N)$ so-called stabilization "rounds" need be ran by every node every half-life to update routing state (successors, predecessors, and fingers). However, this (i) transfers overhead from routing to the stabilization phases, (ii) this solution is expensive, yielding a total message overhead of $O(NlogN)$ per half life: e.g., each node in an one-million node network needs to run on the order of 400 stabilization rounds, say, every half hour!, and (iii) detecting the presence/absence of low-bandwidth nodes (which are the great majority) during stabilization is time- consuming and highly error prone (think of nodes behind 56Kbits lines). Hence, given the huge scales and the

[1] There is some disagreement whether this is true altruistic behaviour or a positive externality (i.e., a benefit to the community that results from peers acting in their own self- interest); Nonetheless, be it altruism or "altruism", the benefits to the community contributed by these peers are recognized by all!

For this reason and for brevity in the remaining discussion we refer to altruistic nodes implying both altruistic and powerful nodes highly- dynamic nature of the vast majority of peers, current architectures fail to ensure $O(logN)$ routing in the highly- dynamic case.

Furthermore, even $O(logN)$ hops, achieved in steady-state assuming 'good node behavior', may not be good enough; after all, these are overlay hops with each being translated into multiple physical network hops. In addition, even $O(logN)$ hops over peers with low bandwidth will definitely create performance problems. Finally, within the DHT world there is a complete lack of attention on exploiting powerful peers in order to improve performance.

But even when considering the unstructured P2P research efforts, one also notices a lack of considerable attention on research exploiting the heterogeneities among peer nodes [12]. As an exception, [15] talk about exploiting powerful nodes, which are thought of consisting of a number of smaller, "virtual" nodes. This transforms several hops among weaker nodes, into internal "virtual hops" within a powerful peer. [16] presents distributed algorithms to force-flow increased loads towards more capable nodes.

But still, heterogeneity means more than a mere distinction between powerful and weak nodes; there is also heterogeneity with respect to their behavior, being altruistic or selfish. For example, there will be powerful nodes that will not be acting altruistically. It is reasonable to expect that altruistic nodes will tend to have greater (processing, memory, and bandwidth) capabilities, willing to share them (when not in use) with others (practically at very small extra costs, given the flat-rate resource pricing) . This expectation has been validated in [13].

Despite this, the aforementioned related work has made some good progress, showing the way in exploiting powerful nodes. Similar to our work, they are criticizing DHTs and structured overlays in failing to cope with highly-dynamic environments, such as the ones expected in sharing P2P networks [17]. However, this led them to avoid using structured overlays, which unfortunately led to their inability to deliver definite performance guarantees, with respect to routing hop counts and robustness. Conversely, we follow a different path; we add further structure to DHTs, leveraging altruistic peers. In this way, we can deliver definite performance guarantees for the steady-case and, perhaps more importantly, for the highly-dynamic cases. Over and above any hop-count improvements, we ensure a more stable infrastructure, especially during high churn[18].

Fundamental Peer Characteristics and Implications In general, in this work we define altruistic peers to be the peers having the following characteristics. They:

- stay connected for significantly longer periods of time, and
- are willing and possess the necessary capacity to accept greater loads.

With these altruists' characteristics in mind we revisit the "traditional" arguments about routing hot spots and about the overhead in dealing with the frequent topology changes inherent in P2P networks. Specifically:

- It is a good idea to concentrate most routing chores at altruistic peers; these peers are willing to carry extra load and have the required capabilities to do so. This results in more efficient routing than forcing weaker nodes to partake heavily in the routing tasks.

– The above decision will undoubtedly create greater routing tables at altruists. Traditionally, this causes greater reorganization overhead incurred when nodes enter and leave the network. However, the additional routing table entries of altruists will concern other altruistic peers. Because these stay connected for long periods of time, maintaining the freshness of this extra routing state does not result in prohibitively increased bandwidth overheads.

Our Position and Contributions This paper intends to show how to leverage the coexistence of altruists and selfish peers found in real-life networks and harness them to improve routing performance. We will focus on structured networks, and in particular we will base our proposal on the desirable characteristics of Chord[2], which include its simplicity, its acceptability within the community (as evidenced by the systems utilizing it, which include Limewire, MNet, Sun's JXTA, and others) and its flexibility [19]. However, the proposed architecture can also be applied on other DHTs as well. More specifically, our position is:

1. Weaving into the structured P2P network architectures the behavior and capability differences of peers, much-needed, quantifiable, and significant further routing speedups can be attained.
2. Routing speedups should refer to hop counts, routing state size and maintenance requirements, and robustness, and they should not be achieved by transferring overhead to other system operation phases (e.g., stabilization).
3. Routing speedups should pertain to the steady-state and highly-dynamic cases.
4. Altruistic and powerful nodes can be harnessed to offer these significant efficiency gains, while requiring that only a very small percentage of peers be altruistic, being burdened with only small overheads.
5. A software layer responsible for identifying and managing altruistic and powerful nodes can go long ways in offering these significant efficiency gains, while requiring that only a very small percentage of peers be altruistic, being burdened with only small overheads.
6. As a result, a paradigm that facilitates the cooperation of an altruist-based architecture and an auditing/accounting layer identifying altruist nodes is needed in order to take the next step in structured P2P network architectures.

2 PLANES: Altruists to the Rescue

In PLANES, node and document IDs consist of m bits, allowing $N \leq 2^m$ nodes and documents. A small percentage of nodes (i.e. $A << N$) is altruistic.

2.1 Architecting Layered, Altruism-based P2P Networks

The fundamentals of our approach are:

[2] Due to this, the log()-notation in the examples to follow refers to base-2 logarithms.

1. The N nodes are partitioned into DHT-structured clusters. For each cluster, a DHT-structured overlay network is created. (Note that any similar structure of choice can be used instead of DHTs, such as [20, 21] yielding similar relative performance gains to that we show for DHTs). Given the desired cluster size, S, and the number of clusters, C, overall, $C = \frac{N}{S}$ clusters are formed.
2. Each node requires minimal overhead for associating node IDs with cluster IDs (e.g. by hashing node IDs to cluster IDs).
3. The vast majority of a cluster's peers are selfish. Within each cluster there exists at least one altruistic peer.
4. Altruistic peers maintain greater routing state, with an entry for all other altruists, creating a completely connected altruistic overlay network. Thus, communication between altruistic nodes (and thus between clusters) requires 1 hop[3]. Later, we will do away with the complete connectivity requirement.
5. Within every DHT cluster, all nodes maintain routing state for their neighbors, as required by the cluster's DHT. Thus, each node has $O(logS)$ neighbors. Also, all nodes keep routing state pointing to the altruistic node(s) in their cluster.

Routing is performed in two levels:

1. Across clusters, from any node to a node in a different cluster: given the completely-connected altruistic overlay network, routing to reach any cluster from outside the cluster requires two overlay hops: one hop from a node to its altruist and another from this altruist to the altruist of the target cluster.
2. Within clusters, from any node (including an altruistic node) to another node in the same cluster: routing is performed by sending the message over the cluster DHT network.

Fig. 1 visualizes an example layered P2P network.

Forming Clusters Using Consistent Hashing We present one possible alternative for cluster formation.

1. Clusters are assigned IDs as follows:
   $ID(c_i) = \frac{2^m}{C} \cdot (i-1),\ i = 1, \ldots, C$
2. Using consistent hashing[22][4] each node ID is mapped to one of the C cluster IDs: node n_j is assigned to cluster c_i if $ID(c_i) < ID(n_j) < ID(c_{i+1})$. Thus, step 1 partitions a Chord ring into C subrings, onto which node IDs are assigned (see fig. 2).
3. Using consistent hashing once more each node ID is placed within its cluster (sub-ring).

Figure 2 illustrates the clusters fomred for a network of eight clusters.

[3] In practice, we talk about a "highly-connected" altruistic network, and O(1) hops since complete connectivity is hard to achieve.

[4] Consistent hashing ensures that, with high probability, all clusters will have equal numbers of peer nodes (with negligible differences).

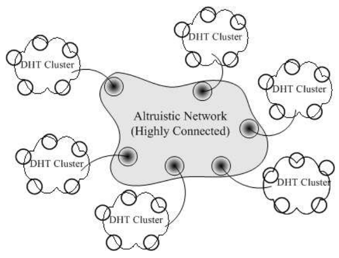

Fig. 1. An Example Network with N=30, C=6, S=5

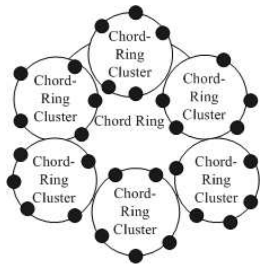

Fig. 2. Forming Clusters in an Altruism-based P2P Network – A Chord Ring of Chord Rings

2.2 PLANES Algorithms

Routing The routing algorithm is called at a node with srcNodeId, requesting docId, and produces the nodeId storing this document. In the algorithm, there are two functions used to send messages: send() and Route() (uses the cluster's DHT to route the message).

PLANESRouting (srcNodeId, docId): nodeId

```
1: srcNodeId.send (srcAltrNodeId, docId); /* send request to altruist within source cluster */
2: ClusterId := srcAltrNodeId.ConsistentHash(docId); /* id target cluster */
3: destAltrNodeId:=srcAltrNodeId.LookUp (ClusterId); /* get target cluster's altruistic node id
   */
4: srcAltrNodeId.send (destAltrNodeId, docId); /* send to target cluster's altruistic peer */
5: nodeId := destAltrNodeId.Route (ClusterId, docId); /* route to node in target cluster and
   return its id */
```

Addition and Deletion of Documents The id of the document being added is hashed to, first, the proper cluster and then, using the cluster's DHT document addition protocol, to one of the cluster's nodes where it is stored. Deleting a document requires no extra state maintenance.

Addition and Deletion of Peer Nodes A node (altruistic or not) n_i joins the network, as usual, by finding the address of any member node, say of cluster c_s. Communicating with it, finds the address of the altruistic node for c_s. Then the id of the cluster to which it will belong, c_d, is determined by hashing the id of n_i, as explained earlier in sec. 2.1 (discussing cluster formation using consistent hashing). The altruistic node of the target cluster c_d is then found using the altruist for c_s. Finally, n_i joins the DHT structure of the target cluster, using the DHT join protocol and including an entry in its routing table for the cluster's altruist.

Adding an altruist also involves communication within the altruist network to store the altruist network's routing table and update the other altruists' tables to include it. Further, all DHT-cluster nodes need add an entry for it.

A node leaving the network affects only the cluster's DHT structure, calling on the leave protocol of the DHT. Altruist deletion requires informing its cluster nodes and may require finding a replacement for it, e.g., if it is the last altruist for its cluster. This can be done examining its local routing table and selecting another altruist (typically one that is associated with a small number of clusters).

2.3 Performance of Routing in PLANES

To illustrate the improved efficiency, we first present a simple "winning configuration" for PLANES. We set $C = \frac{N}{logN}$, yielding $S = logN$. Furthermore, we structure each cluster network using a DHT, as explained in sec. 2.1.

Routing Hop-Count Efficiency This configuration will ensure routing in $O(logS) = O(log(logN))$ hops, in the steady-state case, within each cluster. This follows straightforwardly from the adoption of a DHT-cluster organization and the $O(1)$ hop-count routing between the altruistic nodes.

The "hard" case occurs when peers join and leave the network concurrently with such high rates that the DHT cannot maintain fresh routing states (i.e., information about neighbors of nodes). Depending on the adopted DHT organization, the probability of facing this "hard" case varies. However, in highly dynamic networks, as argued earlier, the performance of routing will degrade to $O(N)$. Since DHTs guarantee routing hop counts in this case in the order of the size of the DHT network, it follows that with our architecture, in this highly dynamic case, routing is ensured in $O(S) = O(logN)$ hops, given the $O(1)$ hops between the altruists.

The existence of $\frac{N}{logN}$ clusters implies requiring a number of at least $\frac{N}{logN}$ altruistic peers (one per cluster). However, note that as N increases, an increasingly smaller percentage of N is required: e.g., for 1000 nodes, 10% of the nodes need be altruistic; for one million nodes, only 5% of the nodes need be altruistic.

The following examples, quantify the improvements. For simplicity, in most discussions we drop the O-notation, since the constant factors should be the same for our architecture and for DHT-style architectures.

Routing Performance at Larger Scales: Example Consider a ($N =$) one million node network. Then, routing in a traditional DHT overlay requires ($logN =$) 20 hops, in the steady-state. In the highly-dynamic case, DHT routing requires $O(N)$ hops – a gloomy prospect!

For altruism-based routing we assume that 5% (50,000) of peers are altruistic. Having one altruistic node per cluster, each cluster will have about $logN$ ≈20 peer nodes and the routing within each cluster will require $log(logN)$ ≈4 hops and 6 in total, after adding the two hops needed to get to the source and destination clusters' altruistic nodes, in the steady-state case. Therefore, in similarly-sized networks, routing hop-count efficiency is improved by a factor of better than $3\times$.

In the highly-dynamic case, when large numbers of peers join and leave the network concurrently and frequently and the special neighbors structure of the DHT cluster breaks, altruism-based routing requires $O(S) = O(logN)$ ≈20 DHT hops, plus 2 for reaching the target cluster. This yields an impressive improvement of several orders of magnitude!

Routing Performance at Smaller Scales: Example For smaller size networks, with one thousand peer nodes, traditional solutions yield a routing hop count about ($logN =$) 10 and 1000 in the steady-state and highly-dynamic cases, respectively. With altruism-based routing, the steady-state routing hop count is about 3 (i.e. $log(log1000)$ and the highly-dynamic case is about $S = log1000$ = 10, yielding improvement factors of about $2\times$ (after accounting for altruistic network hops) and about two orders of magnitude, respectively.

2.4 PLANES Architecture Extensions

In dynamic systems, the main concern is the overhead for maintaining freshness of routing state about altruists, which may become expensive or even impossible to maintain, breaking the complete-connectivity of the altruistic network. We first work in two dimensions: (i) reduce the required number of altruists and (ii) reduce their routing state. Then, we discuss an architecture without complete connectivity for the altruists network.

PLANES with Fewer Altruists: Riding the $Log()$ Factor... We present an extended architecture, showing how to offer small sacrifices in routing speedups, to secure dramatic reductions in the overhead for maintaining altruist network routing state.

We architect the network to consist of $C = \frac{N}{z \times logN}$ clusters, yielding a cluster size of $S = z \times logN$ for an appropriately-valued integer z. The key is to note that: (i) the required number of altruists (which is equal to C) decreases by a factor equal to z; and (ii) the routing hop count within each DHT cluster increases by $logz$, since $logS = log(z \times logN) = logz + log(logN)$.

Example: with one million nodes and $z = 6$, $C \approx 8335$ clusters, cutting the state and freshness overhead requirements by a factor of $6\times$ (going from a 50,000 altruistic-nodes network down to 8335 nodes). In return, instead of routing in <5 hops within the DHT cluster, we now require about <7 hops ($log120 \approx 7$). Taking this point further, using $z = 50$, results in 1000 clusters, each with 1000 nodes in it. Routing within each cluster now needs 10 hops; and the required number of altruists are now reduced by a factor of $50\times$.

Thus, we can still ensure significant speedups in routing (by a factor of about $2\times$ for the steady-state case and by orders of magnitude in the highly-dynamic case). At the same time, the overhead for maintaining high connectivity within the altruistic network, is dramatically reduced, since the number of altruists has been drastically reduced. Finally, we stress that requiring such a small percentage (0.1%) of altruistic nodes is not unrealistic [11–13].

PLANES with Less Altruism, and/or Less Power: Keep on Clustering... Alternatively (or complementarily) to the previous architecture, one can adopt the same layering principle recursively with respect now to the altruistic network. The motivation for this is twofold: (i) reduce further the overhead for maintaining complete/high connectivity between altruists, and (ii) exploit possible heterogeneities among altruists.

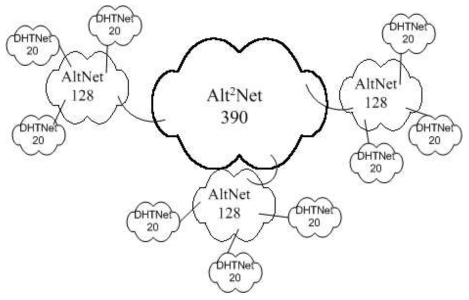

Fig. 3. A Multilevel Altruism-based P2P Network.

Fig. 3 outlines our approach for a three-layer P2P network. This network now includes a clustered altruistic P2P network with $A = 50,000$ altruistic nodes, partitioned in $C = 390$ clusters (AltNets), each of $S = 128$ altruistic nodes. This configuration is obtained by employing the $C = \frac{A}{z \times logA}$ configuration presented earlier and applied now for the altruistic network of A nodes, with $z = 8$.

Altruists belong in AltNet, or in Alt2Net. In the network in Figure 3 the network has N =1,000,000 nodes, with 5% (50,000) altruistic nodes being organized into two levels (of 390 altruistic clusters - AltNets - with 128 altruists each and an Alt2Net network of 390 altruists, one for each AltNet cluster). Each DHT cluster (DHTNet) consists of 20 nodes. Each DHTNet is served by an AltNet, with each AltNet serving 128 DHTNets. All AltNets and the Alt2Net are as before (almost) completely connected. The requirements in order to maintain this high connectivity are further drastically reduced: altruists within an AltNet (Alt2Net) need have 128 (390) neighbors. This represents a reduction by a factor of about 400 to 128, respectively.

At the same time, the significant routing speedups are maintained. For the example one-million-node network, instead of requiring two hops to reach the target DHT cluster, now two or four hops are required for this purpose (depending on whether the source and destination nodes are served by the same AltNet overlay, or not, respectively). So a total of about 6-8 hops are required: 2-4 to reach the target DHT cluster and about 4 to reach the target node within it. This gives a speedup of about 2.5 to 3$\times$, compared to the 20 hops required in the steady-state case for the plain DHT architecture. Again, in the highly-dynamic case, the speedup is several orders of magnitude.

Note that, the more layers we deploy, the less the routing state powerful peers have to maintain. The latter, however, comes at the expense of routing hops. Whether more layers are suitable or not is highly dependent on the characteristics of the network and the actual application.

PLANES without Altruistic Network Complete Connectivity Altruistic networks can also be structured using a DHT, doing away with the completely-connected network. The layered routing algorithm presented earlier, needs to be updated to combine steps 3 and 4 and call on the DHT's routing operation to send the request to the target cluster.

Routing without Complete Connectivity: The altruists form a Chord unit ring onto which altruistic node ids are placed. In addition, cluster ids are mapped to the same ring (like document ids are mapped to Chord rings). Thus, altruists are responsible for clusters, as nodes are responsible for documents in Chord networks. Routing then at the top layer (i.e., from a source-cluster altruist to the destination cluster altruist) is done by routing through the altruists' Chord ring for the destination cluster id. The other steps remain the same.

Consider a network with N = 1,000,000, nodes with $S = C = \sqrt{N}$ =1,000. Employing a DHT for the C =1,000 altruists does not improve hop counts in the steady-state case: about $logC$ (=10) hops would be required to route through the altruists' DHT and $logS$ (=10) hops to route within the target cluster's DHT.

However, this still leverages the characteristics of altruists to yield high routing performance in the highly-dynamic case. Since we assume that the altruist routing state about their neighbors can be maintained even in the highly- dynamic case, routing within it requires $O(logC) = O(log\sqrt{N})$. But even in the unlikely case where this cannot be achieved, routing within the altruists' DHT will require $O(C) = O(\sqrt{N})$. Routing within the cluster DHTs requires in the highly-dynamic case $O(S) = O(\sqrt{N})$. Thus, overall, $O(\sqrt{N})$ hops are required, a drastic improvement compared to DHTs, making this architecture highly desirable.

Finally, in the architecture of sec. 2.4, altruist DHTs can be used at one level only. This still enjoys smaller hop counts in the steady-state and highly-dynamic cases, and requires a percentage of completely-connected nodes.

3 AltSeAl: Altruism Sealed

SeAl[23] is a software layer we have developed for identifying selfish peers and giving them incentives to behave fairly so to improve load balancing and overall data-access performance. SeAl can be transparently incorporated in structured and unstructured networks. Here we outline AltSeAl, a modified version of SeAl, addresssing the needs of AESOP.

3.1 The Monitoring/Accounting Layer

The basic idea is that all transactions between peers result in the creation of tokens (called "Transaction Receipts" or TRs) that can be used much like "favors" in real life; the peers rendering favors (i.e. sharing resources) gain the right to ask peers receiving favors to somehow pay them back in the future or get "punished" otherwise.

All of these operations are performed transparently to the user. Nodes keep track of the favors they render or receive (i.e. store the corresponding TRs) in two "favor lists": the "Favors-Done" (F_d) and "Favors-Owed" (F_o) lists. Moreover, nodes in AltSeAl are characterized by their "altruism score" (denoted by $n_i.\mathcal{A}$). This is simply a function of $|F_d|$ and $|F_o|$, where $|X|$ denotes the size of the set X. For example, we can consider $|F_d| - |F_o|$ or $\frac{|F_d|}{|F_o|}$ as possible altruism score functions.

If node n_1 shares a resource r_1 and node n_2 accesses it, the favor-lists mechanism enables n_1 to selectively redirect a subsequent incoming request for r_1 to n_2. AltSeAl nodes autonomously and independently set an upper ($n_i.\mathcal{A}_{max}$) and a lower ($n_i.\mathcal{A}_{min}$) threshold value for their score. When they rate higher than $n_i.\mathcal{A}_{max}$ they always redirect incoming requests (if possible), while never redirecting when rating lower than $n_i.\mathcal{A}_{min}$. In all other cases, nodes with a tunable probability decide whether to serve or redirect.

3.2 Favors and Complaints

In the previous scenario, if n_2 serves the redirected request, then the corresponding favor is marked as paid-back. Otherwise, n_1 may choose to use the corresponding TR – i.e. $TR_{n_1}^{n_2}(r_1)$ – as a means of accusing n_2 of acting selfishly. This is accomplished in the following manner. AltSeAl uses a DHT overlay of its own to store "complaints". n_1 sends $TR_{n_1}^{n_2}(r_1)$ to the appropriate node – say n_3 – on this DHT (found by hashing the TR itself). n_3 then acts as an arbitrator between n_1 and n_2; it can ask (both) nodes to verify $TR_{n_1}^{n_2}(r_1)$ and have n_2 pay back the corresponding favor. If the verification succeeds but n_2 still refuses to play fair, n_3 stores $TR_{n_1}^{n_2}(r_1)$ for other nodes to know. If the verification fails, n_3 may choose to similarly "complain" about the perjurer peer.

What's more interesting is that, if n_2 chooses to go altruist at some time, it can go out on the DHT, collect all filed complaints, and selectively pay them back, thus improving its status with respect to the community. Note that, while complaints are sent out to the DHT, TRs concerning favors done or paid-back are kept locally at the altruistic peer. Moreover, to keep storage requirements constant as the system evolves, we use an aging scheme for stored TRs.

TRs are constructed using strong (e.g. public-key) cryptographic primitives, while nodes in AltSeAl are equipped with a public/private key-pair and identified using a digest of their public key, also used to verify TRs. Thus, nodes can't fake TRs or refuse the validity of a TR, unless they change their ID (key-pair). Furthermore, AltSeAl deploys a feedback mechanism rather than a penalizing one – requests are queued and served in a prioritized manner, while the actual resources allocated for serving these requests (e.g. bandwidth, storage, etc.) vary, based on the overall "score" of the served peers. Moreover, peers commence their lifecycle in the system with the worst possible score, thus having no incentive to change their ID or mount a Sybil [24]-class attack.

3.3 SeAled PLANES: AltSeAl and AltNets

AESOP requires the capability to find the IDs of a number of altruistic peers, asynchronously with respect to when they assumed such a status. In the context of AltSeAl, altruism can be expressed using the "altruism score". For example, only peers with $\frac{|F_d|}{|F_o|} \geq 2$ may be deemed altruists. Proofs of altruism are also needed. In AltSeAl, TRs of favors rendered or paid-back can serve for this purpose[5]. Using these TRs, further auditing is possible, validating a peer's claims for altruism.

Finally, AESOP needs a structure to manage altruists. We use a second DHT-based (e.g. Chord[2]) overlay for the altruists – the *AltDHT*. As soon as a node n is proved to be an altruist, the *AddToAltDHT(n)* routine is called. This routine:

1. Is directed to the node n', responsible for maintaining the "complaints" for n.
2. n' audits n, retrieving its white records and checking the locally-stored black records for it.
3. If the audit is successful (i.e. n's altruism score is higher than the system's altruism threshold), n' computes an altruist ID (e.g. by hashing the concatenation of the string "Altruist" and n's ID) and, using the DHT node addition protocol, it adds n to the AltDHT. If the audit fails, n' returns an error to n.
4. Whenever a node is promoted to the AltDHT it assumes special responsibilities (e.g. routing).
5. When a peer loses its altruist status (e.g. its altruism score drops below the corresponding threshold), it is removed from the AltDHT using the DHT's node deletion protocol.

Note that peers have a natural incentive not to cheat staying in AltDHT when they wish not to be altruists, since they receive extra load. In addition, peers in AltDHT can perform random audits: periodically, they choose a random ID n'' from those in AltDHT and calculate its altruism score. Peers that are discovered to cheat can be ejected! A more elaborate (but also more resource-hungry) approach to verifiable monitoring and auditing is described in [23].

With this infrastructure in place, discovering altruists is straightforward. To discover k altruists we can, for example, compute a random ID, use AltDHT to locate the node responsible for it and then follow k successor pointers (if AltDHT is implemented using Chord).

4 The Performance of AltSeAl

We have simulated AltSeAl. Our performance results show that AltSeAl performs very well in terms of network/storage overhead and the number of transactions required to audit all nodes.

4.1 Experimental Setup

Our experimental setup assumes that AltSeAl 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 2,500 nodes, sharing 50,000 distinct documents, replicated across peers following a Zipf access distribution, with α = 0.7 and 1.2 [25] (for a total of approximately 50.200 and 51,350 documents respectively).

[5] We'll call such "positive" TRs the *white records*, as opposed to "negative" TRs called the *black records*.

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 2500-node case here. The simulation runs for 200,000 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 AltSeAl operations are run on top of a DHT, thus every AltSeAl transfer is assumed to take $O(log(N)) = 11$ hops on average (for the 2500-node network). As we'll show shortly, in spite of this handicap, AltSeAl incurs negligible network overhead. Moreover, should AltSeAl 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.

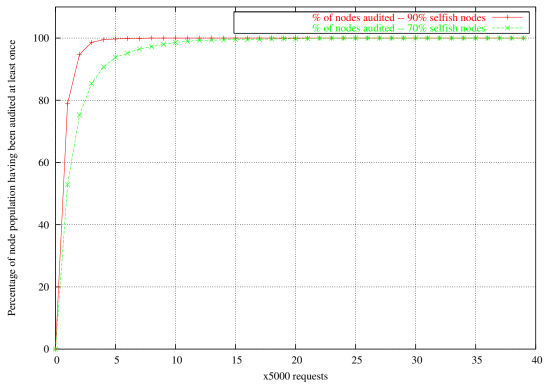

Fig. 4. Performance of the Auditing Mechanism

The simulation results are depicted in fig. 4 and 5. In the simulation the TR aging mechanism was off, hence the linear growth of the storage overhead (fig. 5). Even with this handicap, AltSeAl inflicts on average a mere 0.16% storage overhead. Moreover, the network overhead stabilizes to at most approximately 0.7%, while more than 90% of the total node population has been audited, after the first 5k requests (fig. 4).

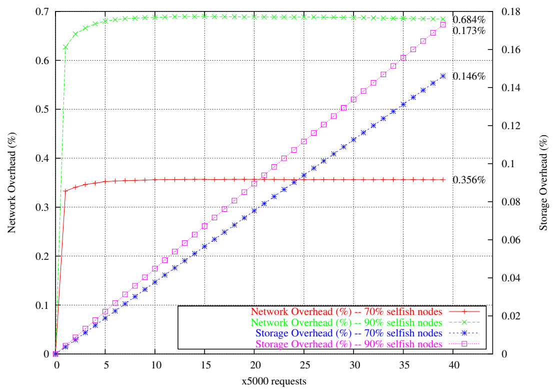

Fig. 5. Network & Storage Overhead of AltSeAl

5 Conclusions

The coexistence of altruistic and selfish peers in P2P networks has been well documented. With this paper we PLANES; an architectural paradigm harnessing these characteristics, weaving them into the structured network architecture. We argued for the need and have shown how to achieve significantly greater routing efficiency in such networks for both the steady-state and highly-dynamic cases, without transferring routing overheads to other system functionalities, and while introducing significant efficiency gains in terms of hop counts, routing state size and maintenance requirements, and robustness.

We presented several architectures and algorithms offering trade-offs between routing speedups vs the required number of altruists and their routing state and between routing path lengths in the steady-state case vs altruist-network connectivity requirements. The end result is that extremely small percentages of altruistic nodes are required, being burdened with small overheads, and introducing steady-state routing speedups by factors of up to 2-4$\times$, and by several orders of magnitude in the highly-dynamic case. At the same time, total routing state size is reduced by a factor of about 2$\times$, which leads to improved robustness.

Furthermore, routing robustness is improved due to the smaller total routing state and the isolation of the ill-effects of selfish behavior within small clusters of peers. Because of the above and its simplicity, we believe the proposed paradigm is viable and realizable and we offer it as the way to structure the P2P networks of the future.

Finally, we presented a number of open problems whose solution can ensure significant further performance gains. In a related thread we are developing a software monitoring/auditing layer that can seamlessly and efficiently discover altruistic nodes within the network [23]. In addition, we are developing a protocol suite integrating this layer with the PLANES architecture. The two components lead to a self-organizing, altruist-inspired, dynamic P2P network architecture, maintaining the highly desirable performance characteristics of PLANES, as presented here.

References

1. Druschel, P., Rowstron, A.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. (In: Proc. Middleware '01)
2. Stoica, I., et al.: Chord: A scalable Peer-To-Peer lookup service for internet applications. (In: Proc. ACM SIGCOMM '01)
3. Ratnasamy, S., et al.: A scalable content-addressable network. (In: Proc. ACM SIGCOMM '01)
4. Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. (Technical Report UCB/CSD-01-1141)
5. Gnutella: (http://rfc-gnutella.sourceforge.net/)
6. Clarke, I., Sandberg, O., Wiley, B., Hong, T.: Freenet: A distributed anonymous information storage and retrieval system. (In: Proc. ICSI Workshop on Design Issues in Anonymity and Unobservability '00)
7. LimeWire: (http://www.limewire.org/)
8. Mnet: (http://mnetproject.org/)
9. Mojonation: (http://www.mojonation.com/)
10. Castro, M., Costa, M., Rowstron, A.: Should we build gnutella on a structured overlay? (In: Proc. HotNets II '03)
11. Adar, E., Huberman, B.: Free riding on Gnutella. Technical report, Xerox PARC (2000)
12. Saroiu, S., Gummadi, K., Gribble, S.: A measurement study of peer-to-peer file sharing systems. (In: Proc. MMCN '02)
13. Wilcox-O'Hearn, B.: Experiences deploying a large-scale emergent network. (In: Proc. IPTPS '02)
14. Liben-Nowell, D., Balakrsihnan, H., Karger, D.: Observations on the dynamic evolution of peer-to-peer networks. (In: Proc. IPTPS '02)
15. Ratnasamy, S., Shenker, S., Stoica, I.: Routing algorithms for DHTs: Some open questions. (In: Proc. IPTPS '02)
16. Lv, Q., Ratnasamy, S., Shenker, S.: Can heterogeneity make Gnutella scalable? (In: Proc. IPTPS '02)
17. Chawathe, Y., et al.: Making Gnutella-like P2P systems scalable. (In: Proc. SIGCOMM '03)
18. Rhea, S., Geels, D.: Handling churn in a DHT. (In: Proc. USENIX Technical Conference '04)
19. Gummadi, K., et al.: The impact of DHT routing geometry on routing resilience and proximity. (In: Proc. SIGCOMM '03)
20. Malkhi, D., Naor, M., Ratajczak, D.: Viceroy: A scalable and dynamic emulation of the butterfly. (In: Proc. ACM PODC '02)
21. Aberer, K., Hauswirth, M., Punceva, M., Schmidt, R.: Improving data access in P2P systems. IEEE Internet Computing **6** (2002)
22. Karger, D., et al.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. (In: Proc. ACM STOC '97)
23. Ntarmos, N., Triantafillou, P.: SeAl: Managing accesses and data in peer-to-peer sharing networks. (In: Proc. IEEE P2P '04)
24. Douceur, J.: The Sybil attack. (In: Proc. IPTPS '02)
25. Sripanidkulchai, K.: (The popularity of gnutella queries and its implications on scalability) White paper, Feb. 2001.