SPROUT: P2P Routing with Social Networks ⋆ ⋆⋆

Sergio Marti, Prasanna Ganesan and Hector Garcia-Molina
{smarti, prasannag, hector}@cs.stanford.edu

Stanford University

Abstract. In this paper, we investigate how existing social networks can benefit P2P data networks by leveraging the inherent trust associated with social links. We present a trust model that lets us compare routing algorithms for P2P networks overlaying social networks. We propose SPROUT, a DHT routing algorithm that, by using social links, significantly increases the number of query results and reduces query delays. We discuss further optimization and design choices for both the model and the routing algorithm. Finally, we evaluate our model versus regular DHT routing and Gnutella-like flooding.

1 Introduction

Social networks are everywhere. Many people all over the world participate online in established social networks every day. AOL, Microsoft, and Yahoo! all provide instant messaging services to millions of users, alerting them when their friends log on. Many community websites, such as Friendster [1], specialize in creating and utilizing social networks. As another example, service agreements between ISPs induce a "social" network through which information is routed globally. Social networks are valuable because they capture trust relationships between entities. By building a P2P data-management system "on top of", or with knowledge of, an existing social network, we can leverage these trust relationships in order to support efficient, reliable query processing.

Several serious problems in peer-to-peer networks today are largely due to lack of trust between peers. Peer anonymity and the lack of a centralized enforcement agency make P2P systems vulnerable to a category of attacks we call misrouting attacks. We use the term *misrouting* to refer to any failure by a peer node to forward a message to the appropriate peer according to the correct routing algorithm. This includes dropping the message or forwarding the message to other colluding nodes instead of the correct peer, perhaps in an attempt to control the results of a query. For instance, in a distributed hash table (DHT) a malicious node may wish to masquerade as the index owner of the key being queried for in order to disseminate bad information and suppress content shared by other peers.

Using *a priori* relationship knowledge may be key to mitigating the effects of misrouting. To avoid routing messages through possibly malicious nodes, we would prefer

⋆ This research is supported in part by NSF Grant (IIS-9817799).

⋆⋆ A short position paper on this topic will be presented at IPTPS 2004. This paper elaborates on the design and presents new experiments and results.

forwarding our messages through nodes controlled by people we know personally, perhaps from a real life social context. We could assume our friends would not purposefully misroute our messages.[1] Likewise, our friends could try and forward our message through their friends' nodes. Social network services provide us the mechanism to identify who our social contacts are and locate them in the network when they are online.

Misrouting is far from the only application of social networks to peer-to-peer systems. Social networks representing explicit or implicit service agreements can also be used to optimize quality of service by, for example, minimizing latency. Peers may give queue priority to packets forwarded by friends or partners over those of strangers. Thus, the shortest path through a network is not necessarily the fastest.

In Section 2 we present a high-level model for evaluating the use of social networks for peer-to-peer routing, and apply it to the two problems we described above; yielding more query results and reducing query times.

Unstructured networks can be easily molded to conform to the social links of their participants. OpenNap, for example, allows supernodes to restrict themselves to linking only with reputable or "friendly" peer supernodes, who manage message propagation and indexing. However, structured networks, such as DHTs, are less flexible, since their connections are determined algorithmically, and thus it is more challenging to use social networks in such systems. In Section 3 we propose SPROUT, a routing algorithm which uses social link information to improve DHT routing performance with respect to both misrouting and latency. We then analyze and evaluate both our model and SPROUT in Section 4.

2 Trust Model

The basic intuition of this paper is that computers managed by friends are not likely to be selfish or malicious and deny us service or misroute our messages. Similarly, friends of friends are also unlikely to be malicious. Therefore, the likelihood of a node B purposefully misrouting a message from node A is proportional to (or some function of) the distance from A's owner to B's owner in the social network. Observe that in a real network with malicious nodes, the above intuition cannot hold simultaneously for all nodes; neighbors of malicious nodes, for example, will find malicious nodes close to them. Rather our objective is to model trust from the perspective of a random good node in the network. Likewise, we assume messages forwarded over social links would experience less latency on average because of prioritizing based on friendship or service agreements.

We now describe a flexible model for representing the behavior of peers relative to a node based on social connections. We will illustrate the model usage for two different specific issues: minimizing the risk of misrouting, and decreasing latency to improve Quality of Service.

[1] We assume a slim, but nonzero, chance that a virus or trojan has infected their machine, causing it to act maliciously.

2.1 Trust Function

We express the trust that a node A has in peer B as $T(A, B)$. Based on our assumption, this value is dependent only on the distance (in hops) d from A to B in the social network. To quantify this measure of trust for the misrouting scenario, we use the expected probability that node B will correctly route a message from node A. The reason for this choice will become apparent shortly.

One simple trust function would be to assume our friends' nodes are very likely to correctly route our messages, say with probability f = 0.95. But their friends are less likely (0.90), and their friends even less so (0.85). Note, this is not the probability that the peer forwards each packet, but instead the probability that the peer is not misbehaving and dropping all packets. Averaged over all nodes, they are equivalent. A node's trustworthiness decreases linearly with respect to its distance from us in the social network. This would level off when we hit the probability that any random stranger node (far from us in the social network) will successfully route a message, say r = 0.6. For large networks with large diameters probability r represents the fraction of the network made up of good nodes willing to correctly route messages. Thus, r = 0.6 means that we expect that 40% of the network nodes (or more accurately network node identifiers) will purposefully misroute messages. Here we have presented a linear trust function. We consider others in the extended version of this paper [2].

When measuring QoS we would want to use a very different function. Let $T(A, B)$ be the expected additional latency incurred by a message forwarded through node B, which it received from node A. For simplicity, let us assume that $T(A, B) = \epsilon$ if a social link exists between A and B and $\Delta \cdot \epsilon$ otherwise. For example, assume ϵ = 1 and Δ = 3. If A has a service agreement, or is friends with, B, then B give any message it receives from A priority and forward it in about 1 (ms), otherwise it is placed in a queue and takes on average 3 (ms). We will use these same values for ϵ and Δ in our example below and in our analysis in Section 4.4.

We do not claim any of these functions with any specific parameter values is an accurate trust representation of any or all social networks, but they do serve to express the relationship we believe exists between social structure and the quality of routing.

2.2 Path Rating

We wish to use our node trust model to compare peer-to-peer routing algorithms. For this we need to calculate a *path trust rating* P to use as our performance metric. The method for calculating P will be application-dependent (and we will present two specific examples below), but a few typical decisions that must be made are:

1. *Source-routing or hop-by-hop?* Will the trust value of a node on the path be a function of its social distance from the message originator, or only from whom it received the message directly?
2. *How do you combine node trust?* Is the path rating the product, sum, maximum value, or average value of the node trust values along the path? Any appropriate function could be used.

We now give as example a metric for *reliability* in the presence of misrouting. We need to compare the likelihood that a message will reach its destination given the path selected by a routing algorithm. We calculate the reliability path rating by multiplying the separate node trust ratings for each node along the path from the source to destination. For example, assume source node S wishes to route a message to destination node D. In order to do so a routing algorithm calls for the message to hop from S to A, then B, then C, and finally D. Then the reliability path rating will be $P_R = T(S, A) * T(S, B) * T(S, C) * T(S, D)$. Given that $T(X, Y)$ is interpreted as the actual probability node Y correctly routes node X's message, then P_R is the probability that the message is received and properly handled by D. Note that $T(X, Y)$ is dependent only on the shortest path in the social network between X and Y and thus independent of whether Y was the first, second, or nth node along the path.

For the Quality of Service we would want our path rating to express the expected time a message would take to go from the source to the destination. Given that $T(A, B)$ is the latency incurred by each hop we would want to use an additive function. And if each node decides whether to prioritize forwarding based on who it received the message from *directly*, and not the originator, then the function would be hop-by-hop. Calculating the latency path rating for the path used above would be $P_L = T(S, A) + T(A, B) + T(B, C) + T(C, D)$.

Though we focus on linear paths in this paper, the rating function can generalize to arbitrary routing graphs, such as multicast trees.

3 Social Path Routing Algorithm

We wish to leverage the assumed correlation between routing reliability or efficiency and social distance by creating a peer-to-peer system that utilizes social information from a service such as a community website or instant messenger service. Though there are many ways to exploit social links, for this paper, we focus on building a distributed hash table (DHT) routing algorithm. Specifically, we build on the basic Chord routing algorithm [3]. Chord was chosen because it is a well-known scheme and studies have shown it to provide great static resilience, a useful property in a system with a high probability of misrouting that is difficult to detect and repair [4]. Our technique is equally applicable to other DHT designs, such as CAN [5] or Pastry [6].

When a user first joins the Chord network, it is randomly assigned a network identifier from 0 to 1. It then establishes links to its *sequential neighbors* in idspace, forming a ring of nodes. It also makes roughly $\log_2 n$ *long links* to nodes halfway around the ring, a quarter of the way, an eighth, etc. When a node inserts or looks up an item, it hashes the item's key to a value between 0 and 1. Using greedy clockwise routing, it can locate the peer whose id is closest to the key's hash (and is thus responsible for indexing the item) in $O(\log n)$ hops. For simplicity, we will use "key" to refer to a key's hash value in this paper.

Our Social Path ROUTing (SPROUT) algorithm adds to Chord additional links to any friends that are online. All popular instant messenger services keep a user aware of when their friends enter or leave the network. Using this existing mechanism a node can determine when their friends' nodes are up and form links to them in the DHT as

well. This provides them with several highly trusted links to use for routing messages. When a node needs to route to key k SPROUT works as follows:

1. Locate the friend node whose id is closest to, but not greater than, k.
2. If such a friend node exists, forward the message to it. That node repeats the procedure from step 1.
3. If no friend node is closer to the destination, then use the regular Chord algorithm to continue forwarding to the destination.

In order to improve the performance of SPROUT we apply two optimizations, which we evaluate in the extended paper [2]:

Lookahead With the above procedure, when we choose the friend node closest to the destination we do not know if it has a friend to take us closer to the destination. Thus, we may have to resort to regular Chord routing after the first hop. To improve our chances of finding social hops to the destination we can employ a *lookahead* cache of 1 or 2 levels. Each node may share with its friends a list of its friends and, in 2-level lookahead, its friends-of-friends. A node can then consider all nodes within 2 or 3 social hops away when looking for the node closest to the destination. We still require that the message be forwarded over the established social links.

Minimum Hop Distance Though SPROUT guarantees forward progress towards the destination with each hop, it may happen that at each hop SPROUT finds the sequential neighbor is the closest friend to the target. Thus, in the worst case, routing is $O(n)$.

To prevent this we use a *minimum hop distance* (*MHD*) to ensure that the following friend hop covers at least *MHD* fraction of the remaining distance (in idspace) to the destination. For example, if *MHD* = 0.25, then the next friend hop must be at least a quarter of the distance from the current node to the destination. If not then we resort to Chord routing, where each hop covers approximately half of the distance. This optimization guarantees us $O(\log n)$ hops to any destination but causes us to give up on using social links earlier in the routing process. When planning multiple hops at once, due to lookahead, we require the path to cover $\frac{MHD}{k}$ additional distance for each additional hop, for some appropriate k.

4 Results

To test our SPROUT algorithm for DHTs compare it to Chord in the following scenario. Assume the members of an existing social network wish to share files or information by creating a distributed hash table. Believing that some peers in the network are unreliable, each node would prefer to route messages through their friends' nodes if possible. We use two sources for social network data for our simulations. The first is data taken from the Club Nexus community website established at Stanford University [7]. This dataset consists of over 2200 users and their links to each other as determined by their Buddy Lists. The second source was a synthetic social network generator based on the Small World topology algorithm presented in [8]. Both the Club Nexus data and the

Small World data created social networks with an average of approximately 8 links per node. We randomly inserted each social network node into the Chord id space.

We also ran experiments using a trace of a social network based on 130,000 AOL Instant Messenger users and their Buddy Lists provided by BuddyZoo [9]. Because of the size of this dataset, we have only used the data to verify results of our other experiments.

For each experiment we randomly choose 500 query source nodes and, for each node, 500 random key hash values to search for (chosen uniformly from 0 to 1). We compute a path using each routing algorithm and gather statistics on path length and trust rating. Each data point presented below is the average of all 250,000 query paths.

4.1 Algorithm Evaluation

We first focus on the problem of misrouting. We use the linear trust function described in Section 2 with $f = 0.95$ and $r = 0.6$, which corresponds to 40% of the nodes misbehaving. We feel such a large fraction of bad nodes is reasonable because of the threat of Sybil attacks [10].

We compare SPROUT, using a lookahead of 1 and *MHD* = 0.5, to Chord using the Club Nexus social network data. The first and third rows of Table 1 give the measured values for both the average path length and average reliability path rating of both regular Chord routing and SPROUT. With an average path length of 5.343 and average reliability of 0.3080, Chord performed much worse in both metrics than SPROUT, which attained values of 4.569 and 0.4661, respectively. In fact, a path is over 1.5 times as likely to succeed using standard SPROUT as with regular Chord.

Table 1. SPROUT vs. Chord

	Avg. Path Length	Avg. Reliabilty
Regular Chord	5.343	0.3080
Augmented Chord	4.532	0.3649
SPROUT(1,0.5)	4.569	0.4661

But this difference in performance may be simply due to having additional links available for routing, and the fact that they are friend links may have no effect on performance. To equalize the comparison we augmented Chord by giving nodes additional links to use for routing. Each node was given as many additional random links as that node has social links (which SPROUT uses). Thus, the total number of links useable at each node is equal for both SPROUT and augmented Chord. The performance of the augmented Chord (AC) is given in the second row of Table 1. As expected, with more links to choose from AC performs significantly better than regular Chord, especially in terms of path length. But SPROUT is still 1.3 times as likely to route successfully. In the following sections we compare SPROUT only to the augmented Chord algorithm.

From our analysis of the SPROUT optimizations (see [2]) we chose to use a 1-level lookahead and an *MHD* of 0.5 for our standard SPROUT procedure. Though 2-level lookahead produced slightly better reliability we did not feel it warranted the longer route paths and exponentially increased node state propagation and management. Our available social network data indicates that a user has on average between 8 and 9

friends. Thus, we would expect most nodes' level-1 lookahead cache to hold less than 100 entries.

We evaluate different trust functions and parameter values in [2] and find that SPROUT outperforms AC for $r < 0.95$. When 5% or less of unknown peers are likely to misroute ($r \geq 0.95$) both algorithms perform equally well, even with f also 0.95, indicating we trust our friends no more than any stranger. While SPROUT significantly improves path reliability in a peer-to-peer network with many malicious and selfish peers, we do not suffer a reliability penalty for using it in a network with very few bad peers.

4.2 Number of Friends

In a given network, a node with more friends is likely to perform better since it has more choices of social links to use. But how much better? How much improvement would a node expect to gain by establishing some trust relationship with another node? To quantify this, we generated 100 queries from each node in the Club Nexus network, calculated its path rating, and grouped and averaged the results based on the number of social links each node has.

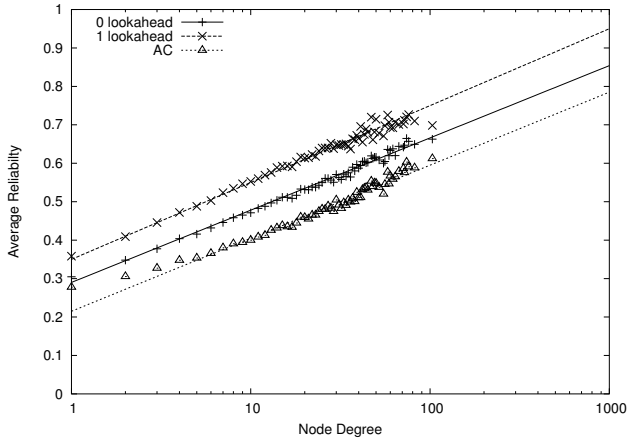

Fig. 1. Performance as a function of a node's degree. Club Nexus data.

Figure 1 shows the results for SPROUT using 0- and 1-level lookahead, as well as AC for the Club Nexus data. For example, 85 nodes in the network had exactly 10 social links. The average path rating for those 85 nodes when running SPROUT with 1 lookahead was 0.553. Note that the three curves are linear with respect to the log of the node degree, indicating an exponentially decreasing benefit return for each additional social link. For instance, nodes with only 1 social peer attained a reliability rating of 0.265 with SPROUT with no lookahead, while nodes with 10 social peers scored 0.471, a difference of 0.206. A node with 10 social peers would need to grow to over 100 social peers to increase their rating that same amount (the one node with 103 social links had a rating of 0.663).

From these curves we can estimate how many links a typical node would need to have in order to attain a specified level of reliability. For instance, considering the

SPROUT with 1-level lookahead curve, we see that a node would need about 100 social links to attain an average rating of 0.7, and about 600 social links to get a rating of 0.9.

Though a single node increasing its number of friends does not greatly influence its performance, what performance can nodes expect if we *a priori* set the number of friend connections each node must have? To analyze this we create a random regular social network graph of 2500 nodes where each node has an equal degree and vary this degree for each simulation run. The results are shown below in Figure 2.

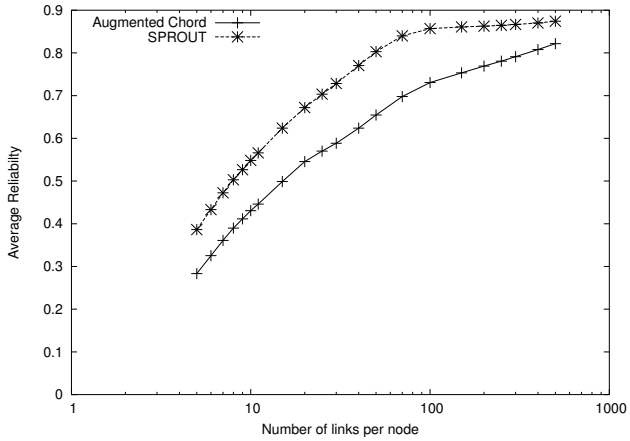

Fig. 2. Performance of SPROUT and AC for different uniform networks with varying degrees.

The curves correspond to SPROUT with 1-level lookahead and augmented Chord. As expected, we see that both curves rise more steeply than in the previous graph. If all nodes add an extra social link the probability of successful routing will rise more than if only one node adds a link (as seen in Fig. 1). But the curves level off just below 0.9. In fact, similar simulations for larger networks showed the same results, with reliability leveling off under 0.9 at around 100 social links per node. This confirms that even at high social degree, each path is expected to take multiple hops through nodes that are, to some small amount, unreliable. Even if all nodes were exactly two social hops away from each other, this would yield a reliability of 0.95*0.9=0.855. Therefore, we would not expect a node in the Club Nexus dataset, as seen in Figure 1, reach 0.9 reliability, even with 600 links.

Though SPROUT provides greater reliability than Chord, neither algorithm performs particularly well. Our results from Table 1 showed ratings of less than 0.50, indicating less than 50% of messages would be expected to reach their destination. Perhaps DHT routing is incapable of providing acceptable performance when members of the network seek to harm it. In the next section, we evaluate the brute force method of query flooding.

4.3 Comparison to Gnutella-like Networks

So far we have limited our analysis of SPROUT to Chord-like DHT routing. We were also interested in comparing the effects of misrouting on structured P2P networks to

unstructured, flooding-based networks, such as Gnutella. To balance the comparison we assume the unstructured network's topology is determined by the social network, using only its social links, and apply the same linear trust function used before to calculate the probability that a node forwards a query flood message.

Because querying the network is flooding-based, we cannot use the probability of reaching a certain destination as our metric. Instead, we would like to find the expected number of good responses a querying node would receive. For a DHT we assume a node would receive all or no responses, depending on whether the query message reached the correct well-behaved index node (we do not consider the problem of inserting item keys into the DHT caused by misrouting). In an unstructured network the number of good responses located is equal to the number of responses at well-behaved nodes reached by the query flood. Because the flood is usually limited in size by a *time-to-live* (TTL), even if there are no malicious nodes in the network, not all query answers will be located.

We modelled a Gnutella-like network with a topology based on the Club Nexus data and used a TTL of 5, allowing us to reach the vast majority of the nodes in the network (over 2000 on average). We seeded the network with files based on empirically collected data from actual networks [11] and ran 10000 queries for different files from varying nodes, dropping a query message at a peer with a probability based on the trust function and the shortest path to the querying node. We averaged across 10 runs (for different file distributions) and present the results, as a function of r (the expected reliability of a node distant in the social network), in Figure 3.

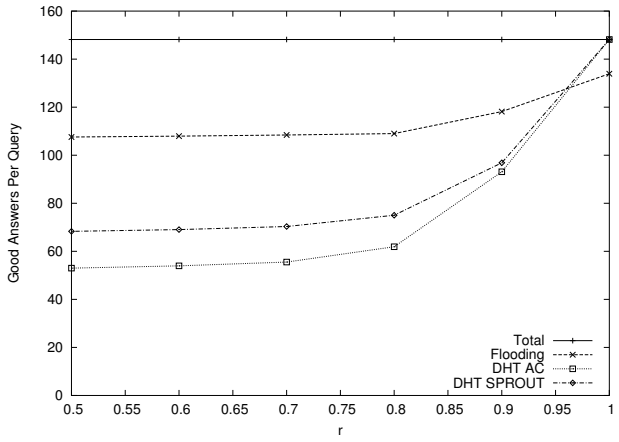

Fig. 3. Performance of SPROUT and AC versus unstructured flooding.

The top curve, labelled *Total*, indicates the total number of files in the entire network matching each query (on average), independent of the routing algorithm used. This value is approximately 150. The expected number of good answers received for the DHT curves was calculated as this total number times the expected probability of reaching the index node storing the queried for items. Flooding results in significantly more responses on average, a factor of almost 2 for small r. More importantly, this means we would expect to locate at least some good answers flooding when the DHT

completely fails. For values of r less than 0.5 all the curves level off. If r = 1 then we assume no nodes in the network are malicious. Thus DHT outperforms flooding since it will always locate the index node and retrieve all the available answers.

Note, these results are meant to be a rough comparison of these two P2P styles. The flooding model does not take into account messages dropped due to congestion. This is a much larger problem for flooding protocols than DHTs. In our simulations on the 2200 node Club Nexus network each query reached, on average, over 2000 nodes. This indicates the number of messages produced by the flood was even greater (due to duplicate messages). The DHT algorithm, on the other hand, averaged around 5 messages to reach the index node. Thus, flooding schemes will not scale to very large networks as well as DHTs.

On the other hand, in the DHT model, we are only considering the probability of a query message being misrouted. We assume all good answers are inserted at the correct index node, not taking into account that index insertions may fail just as well as index queries. If we factor in index insertion failures, the DHT curves would shift down, further increasing the relative performance difference with flooding.

Though flooding is more costly in terms of processor and network bandwidth utilization, it is clearly a more reliable method of querying in a network suffering from some amount of misrouting. A better solution may be to use a hybrid scheme where one uses DHT routing until they detect misrouting or malicious nodes, then switch to query flooding. In fact, a such a scheme is proposed in [12] and discussed in Section 5.

4.4 Latency Comparisons

As we stated before, both SPROUT and our social trust model are not limited to studying misrouting. With few modifications our model can be used to evaluate other issues, such as Quality of Service. If peers prioritized their message queues based on service agreements and/or social connections we may want to use latency as the metric for comparing routing algorithms. Using the latency trust function (with ϵ = 1 and Δ = 3) and latency path rater we described in Section 2, we route messages using both SPROUT and augmented Chord and see which provides the least latency. We would expect SPROUT to perform even better with respect to Chord in such systems.

We performed an analysis to determine the optimal *MHD* for latency-based routing. As in the misrouting scenario an *MHD* of 0.5 performed the best. This is surprising since the latency path rater is additive, not multiplicative. The difference with other values for *MHD* was almost negligible, indicating that for small Δ where the cost of social links and regular links are similar, shortening the overall path outweighs choosing social links. In fact, with a larger Δ of 10, smaller *MHD* values perform significantly better than 0.5.

Figure 4 shows the average path latency for both SPROUT and augmented Chord as a function of the network size (using a Small World topology). The third curve shows the percent decrease in latency attained by switching from AC to SPROUT. We see that SPROUT results in roughly half (40-60%) the latency of AC. We would expect SPROUT to deliver messages twice as fast as AC by preferring to take advantage of service agreements, rather than simply minimizing hop count.

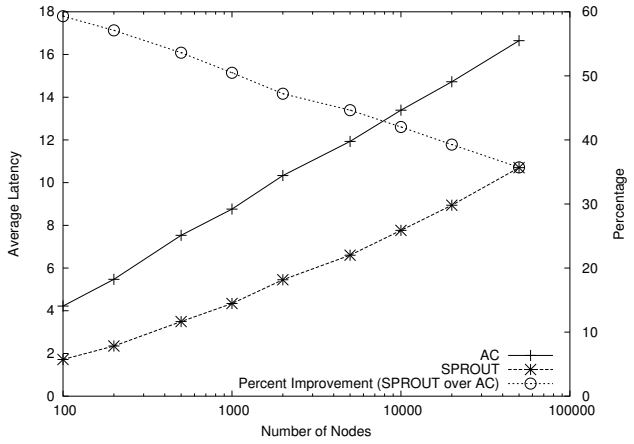

Fig. 4. Latency measurements for SPROUT vs AC w.r.t. network size. Lower is better.

Clearly, Quality of Service issues greatly benefit from routing algorithms which account for service agreements between peers, as SPROUT does. In fact, real-world systems which deal with QoS, such as ISPs and phone carriers, base their routing decisions on service agreements among their peers, though their networks are not as dynamic as peer-to-peer networks.

5 Related and Future Work

In [12], Castro et al propose using stricter network identifier assignment and density checks to detect misrouting attacks in DHTs. They suggest using constrained routing tables and redundant routing to circumvent malicious nodes and provide more secure routing. SPROUT is complementary to their approach, simply increasing the probability that the message will be routed correctly the first time. One technique of theirs that would be especially useful in our system was their route failure test based on measuring the density of network ids around oneself and the purported destination. Not only can this technique be used to determine when a route has failed, but it can be used to evaluate the trustworthiness of a node's sequential neighbors by comparing local density to that at random locations in idspace or around friends.

One method to provide greater fault tolerance and/or security in a DHT, is to replicate the index to multiple nodes. If we do k-replication then when we insert, update or search for an entry in the DHT, we must contact k nodes determined by using k hash functions. If a good node A wishes to insert an item into the DHT, it attempts to contact all k replicas. Each message has an expected probability p of having traversed only well-behaved nodes to the destination. Likewise, if node B wishes to look up the item A inserted it can try to contact all k replicas, each time with an expected probability of success of p. Assuming neither A nor B can determine whether they contacted a good node or are being lied to, the probability of B locating A's item is $1-(1-p^2)^k$. Using the values in Table 1 for p and a typical replication factor of $k = 3$, SPROUT would succeed 41% of the time compared to only 26% for AC.

We are currently analyzing the effects of SPROUT on load distribution. Since it prefers to use social links, we would expect SPROUT to unfairly load popular peers with many social links.

6 Conclusion

Today's peer-to-peer systems are very vulnerable to malicious attacks. The anonymity and transience of the members make it difficult to determine who to trust. Integrating social networks with P2P networks will provide this much-needed trust information.

We have presented a method for leveraging the trust relationships gained by marrying a peer-to-peer system with a social network, and showed how to improve the expected number of query results and the how to reduce the expected delays. We described a model for evaluating routing algorithms in such a system and proposed SPROUT, a routing algorithm designed to leverage trust relationships given by social links. Our results demonstrate how SPROUT can significantly improve the likelihood of getting query results in a timely fashion, when a large fraction of nodes are malicious. Though flooding-based search schemes are far more robust when threatened by a large number of malicious users, with the right techniques structured networks can obtain acceptable performance at far less bandwidth costs.

References

1. Friendster Inc.: Friendster Beta (2003) http://www.friendster.com.
2. Marti, S., Ganesan, P., Garcia-Molina, H.: SPROUT: P2P Routing with Social Networks. Technical report (2004. dbpubs.stanford.edu/pub/2004-5)
3. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, M.F., Dabek, F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup protocol for internet applications. IEEE/ACM Trans. Netw. **11** (2003) 17–32
4. Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., Stoica, I.: The impact of DHT routing geometry on resilience and proximity. In: Proc. ACM SIGCOMM. (2003)
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA (2000)
6. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (2001) 329–350
7. Buyukkokten, O.: Club Nexus (2001)
8. Puniyani, A.R., Lukose, R.M., Huberman, B.A.: Intentional Walks on Scale Free Small Worlds. ArXiv Condensed Matter e-prints (2001) http://aps.arxiv.org/abs/cond-mat/0107212.
9. D'Angelo, A.: (BuddyZoo) http://www.buddyzoo.com.
10. Douceur, J.R.: The Sybil Attack. In: Proceedings of the International Workshop on Peer-to-Peer Systems. (2002)
11. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of Multimedia Computing and Networking 2002 (MMCN '02), San Jose, CA, USA (2002)
12. Castro, M., Drushel, P., Ganesh, A., Rowstron, A., Wallach, D.: Secure routing for structured peer-to-peer overlay networks. In: OSDI '02. (2002)