DHT Routing Using Social Links

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

Abstract— **The equality and anonymity of peer-to-peer networks makes them vulnerable to routing denial of service attacks from misbehaving nodes. In this paper, we investigate how existing social networks can benefit P2P 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 significantly increases the probability of successful routing by using social links. Finally, we discuss further optimization and design choices for both the model and the routing algorithm.**

I. Introduction

Because of the anonymity of peers and the lack of a centralized enforcement agency, P2P systems are especially vulnerable to a category of attacks we call misrouting attacks. We use the term *misrouting* to refer to any failure by a 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. 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.

In addition, malicious users can acquire several valid network identifiers and thus control multiple distinct nodes in the network. This is referred to as the Sybil attack and has been studied by various groups (e.g. [4] [3] [7]). This implies that a small number of malicious users can control a large fraction of the network nodes, increasing the probability that they participate in any given message route.

To avoid routing messages through possibly malicious nodes, we would prefer forwarding our messages through nodes controlled by people we know personally, perhaps from a real life social context. We could most likely assume our friends would not purposefully misroute our messages.[1] Likewise, our friends could try and forward our message through their friends' nodes. This would require a mechanism to identify who our social contacts are and locate them in the network when they are online.

Fortunately, this mechanism already exists in the form of various social network services. AOL, Microsoft, and Yahoo! all provide instant messaging services to millions of users, alerting them when their friends log on. Many websites, like Friendster, specialize in creating and utilizing social networks. We propose building peer-to-peer networks which leverage these existing social network services to establish additional, highly trusted, links at little additional cost. To determine the value of such a system we need a new way of modelling social trust and how it translates to the chance of misrouting. We present such a trust model in Section II.

Adopting social links in an unstructured P2P network is relatively simple, since nodes are free to connect to any peers. However, using them in structured networks (e.g. DHTs) is more challenging because peer connections are typically determined algorithmically. In Section III we present SPROUT, a routing algorithm that takes advantage of social links in structured networks, and compare it to current standard routing techniques in Section IV. Social networks can be exploited by P2P systems for a variety of other reasons. In Section V we discuss application scenarios where our model is useful, as well as other related and future work. Finally, we conclude in Section VI.

II. Trust Model

The basic assumption 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. We assume the likelihood of a node B purposefully misrouting a message from node A is proportional to the distance from A's owner to B's owner in the social network.

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

A. *Trust Function*

We express the trust that a node A has in node 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 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). 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 node will successfully route a message, say $r = 0.6$. In large networks probability r represents the fraction of the network expected to be good nodes willing to correctly route messages. Thus, $r = 0.6$ indicates that we expect that 40% of the network nodes (or more accurately network node identifiers) will purposefully misroute messages. In smaller networks, r is the probability that a peer at a large social distance from us will route correctly. Here we have presented a linear trust function. We consider others in Section IV-C.

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 relation we believe exists between social structure and the probability of intentional routing misbehavior.

B. *Path Rating*

We need to compare the likelihood that a message will reach its destination given the path selected by a routing algorithm. For this reliability metric we calculate a *path trust rating*, denoted by P, by multiplying the separate node trust ratings for each node along the routing 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 path rating will be $P = T(S, A) * T(S, B) * T(S, C) * T(S, D)$. Given that $T(X, Y)$ is interpreted as the actual probability node Y correctly routes node X's message, then P 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.

Including the final destination's trust rating is optional and dependent on what we are measuring. If we wish to account for the fact that the destination may be malicious and ignore a message, we include it. Since we are using path rating to compare routing algorithms going to the same destination, both paths will include this factor, making the issue irrelevant.

III. Social Path Routing Algorithm

We wish to leverage the assumed correlation between routing reliability 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 [11]. Our technique is equally applicable to other DHT designs, such as CAN [9] or Pastry [10].

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 $O(\log n)$ *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, and is thus responsible for indexing the item, in $O(\log n)$ hops.

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 maintain links to their friends 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.

A. *Optimizations*

Here we present two techniques to improve the performance of our routing algorithm. We evaluate them in Section IV-B.

1) 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.

2) 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.

IV. Results

In this section we evaluate our friend-routing algorithm as well as present optimizations. We also discuss the trust model and compare the different trust functions.

A. Simulation Details

To try out our SPROUT algorithm for DHTs we decided to compare it to Chord. 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. 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 assigned each social network node a random id in Chord.

TABLE I
SPROUT VS. CHORD

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

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 [2]. Because of the size of this dataset, we have only used the data to verify results of our other experiments.

For each experiment we chose 1000 random nodes and, for each node, 1000 random keys to search for (uniformly from 0 to 1). We computed a path using each routing algorithm and gathered statistics on path length and trust rating. Each data point presented below is the average of all 1,000,000 paths.

In Section IV-B we use the linear trust function described in Section II 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. We evaluate different trust functions and parameter values in Section IV-C.

B. Algorithm Evaluation

We first evaluate 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 I give the measured values for both the average path length and average path rating (or path *reliability*) of both regular Chord routing and SPROUT. With an average path length of 5.343 and average rating per path 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.

But this difference in performance may be simply due to SPROUT having additional links available for routing, and the fact that they are friend links may have no effect on performance. To even the comparison we augmented Chord by giving each node an equal number of random links for Chord to use as it has friend links. The performance of the augmented Chord (AC) is given in the second row of Table I. 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

TABLE II
EVALUATING LOOKAHEAD AND *MHD*

MHD	Lookahead					
	None		1-level		2-level	
	Length	Rating	Length	Rating	Length	Rating
0	4.875	0.4068	5.101	0.4420	5.378	0.4421
0.125	4.805	0.4070	5.003	0.4464	5.258	0.4478
0.25	4.765	0.4068	4.872	0.4525	5.114	0.4551
0.5	4.656	0.4033	4.569	0.4661	4.757	0.4730

successfully. In the following sections we compare SPROUT only to the augmented Chord algorithm.

How were the lookahead and *MHD* values used above chosen? Table II shows the results of our experiments in varying both parameters in the same scenario. As we see, the largest increase in reliability comes from using a 2-level lookahead. But this comes at a slight cost in average path length, due to the fact that more lookahead allows us to route along friend links for more of the path. For example, for *MHD* = 0.5, no lookahead averaged 0.977 social links per path, while 1-level lookahead averaged 2.533 and 2-level averaged 3.491. Friend links tend to not be as efficient as Chord links, so forward progress may require 2 or 3 hops, depending on the lookahead depth. But friend links are more likely to reach nodes closer to the sending node on the social network.

Increasing *MHD* limits the choices in forward progressing friend hops, causing the algorithm to switch to Chord earlier than otherwise, but mitigates inefficient progress. A large *MHD* seems to be most effective at both shortening path lengths and increasing path rating. This is not very surprising. Since our reliability function is multiplicative each additional link appreciably drops the path reliability.

From these results 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 path ratings 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 the average node's level-1 lookahead cache to hold less than 100 entries.

The path ratings presented above were relatively small, indicating a low, but perhaps acceptable, probability of successfully routing to a destination in the DHT. If the number of friends a user has remains relatively constant but the total number of network nodes increases we would expect performance to drop. But by how much? To study this we ran our experiment using our synthetic Small World model for networks of different sizes with a maximum peer social degree proportional to $O(\log n)$. We present these results in Figure 1.

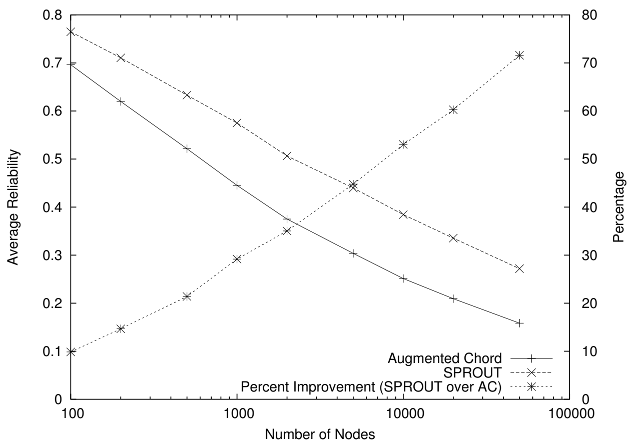

Fig. 1. Performance of SPROUT and AC in different size Small World networks. The third curve shows the relative performance of SPROUT with respect to AC, plotted on the right-hand y-axis. Note that the x-axis is logscale.

We see that the drop in reliability for both routing algorithms is linear with respect to log of the number of nodes. If the average path length were $\Theta(\log n)$ as in Chord, we would expect the reliability to drop exponentially with respect to $\log n$. But the additional use of social links resulted in average path length increasing more slowly than $\Theta(\log n)$. Notice that the relative performance gain of SPROUT over AC increases as the network grows. At 10,000 nodes SPROUT performs over 50% better than AC. As the network grows, the average number of social links increases slightly. The benefit SPROUT derives from additional friend links is greater than the benefit AC derives from additional random links.

C. Calculating Trust

All of our previous results used a linear trust function with f = 0.95. Of course other trust functions or parameter values may be more appropriate for different scenarios. $T(A, B)$, using the linear trust function LT we previously described, is defined in Equation 1 as a function of d, the distance from A to B in the social network.

$$LT(d) = \max(1 - (1 - f)d, r) \quad (1)$$

Instead of a linear drop in trust, we may want to model an exponential drop at each additional hop. For this we use an exponential trust function ET, shown in Equation 2.

$$ET(d) = \max(f^d, r) \quad (2)$$

Another simple function we call the step trust function $ST(d)$ assigns an equal high trustworthiness of f to all

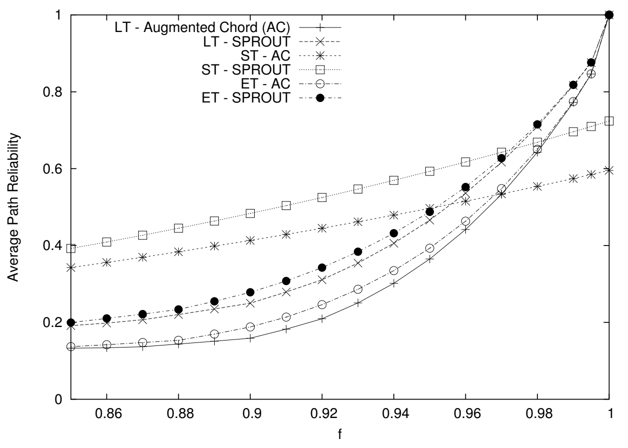

Fig. 2. Performance of SPROUT and AC for different trust functions and varying f. Higher value is better.

nodes within h hops of us and the standard rating of r to the rest. Equation 3 defines the step trust function.

$$ST(d) = \textbf{if } (d < h) \textbf{ then } f \textbf{ else } r \quad (3)$$

In our experiments we set h, the *social horizon*, to 5.

All three functions are expressed so that f is the rating assigned to nodes one hop away in the social network, the direct friends. In Figure 2 we graph both routing algorithms under all three trust functions as a function of the parameter f.

We see here that both the linear (LT) and exponential (ET) trust functions perform equivalently while the step trust function (ST) gives less performance difference for varying f. For all trust functions SPROUT demonstrates a clear improvement over augmented Chord for $f > 0.85$. For example, at $f = 0.96$ using the exponential function SPROUT succeeds in routing 55% of the time, while AC only 46%.

We also varied r, the expected fraction of good nodes in the network. We found that for values of $r < 0.75$ performance remained almost constant. Above 0.75 both algorithms steadily increased.

D. Message Load

One problem SPROUT faces is uneven load distribution due to the widely varying social connectivity of the nodes. Peers with more social links are expected to forward messages for friends at a higher rate than weakly socially connected peers. To study this issue we measure the number of messages forwarded by each node over all 1,000,000 paths for both SPROUT(1,0.5) and augmented Chord. The resulting load on each node, in decreasing order, is given by the first two curves in Figure 3. The load is calculated as the fraction of all messages a node participated in routing.

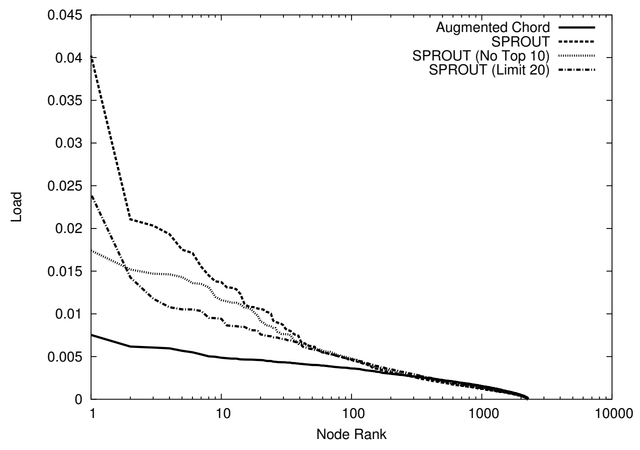

Fig. 3. Distribution of load (in fraction of routes) for augmented Chord and SPROUT. Lower is better. Social links were removed for the top 10 highest connected nodes for the *No Top 10* curve. All nodes were limited to at most 20 social links for the *Limit 20* curve. Note the logscale x-axis.

The highest loaded node in the SPROUT experiment was very heavily loaded in comparison to AC (4% vs 0.75%). As expected, a peer's social degree is proportional to its load, with the most connected peers forwarding the most messages. Though the top 200 nodes suffer substantially more load with SPROUT than AC, the remaining nodes report equal or less load. Because the average path length for SPROUT is slightly higher than for AC, the total load is greater in the SPROUT scenario. Yet the median load is slightly lower for SPROUT, further indicating an imbalanced load distribution.

To analyze the importance of the highly connected nodes we removed the social links from the top 10 most connected nodes, but kept their regular Chord links and reran the experiment. As the third curve in Figure 3 shows, the load has lowered for the most heavily weighted nodes, yet remains well above AC. Surprisingly the reliability was barely affected, dropping by 2% to 0.4569. If highly connected nodes were to stop forwarding for friends due to too much traffic, the load would shift to other nodes and the overall system performance would not be greatly affected.

Instead of reacting to high load, nodes may wish to only provide a limited number of social links for routing from the start. We limited all nodes to using only at most 20 social links for SPROUT. As we can see from the *Limit 20* curve in Figure 3, the load on the highly-loaded peers (excluding the most loaded peer) has fallen further, but not significantly from the *No Top 10* scenario. The average path reliability has dropped only an additional 1.5% to 0.4500.

In the end, it is the system architect who must decide whether the load skew is acceptable. For weakly connected homogeneous systems, fair load distribution may be critical. For other systems, improved reliability may be more important. In fact, one could take advantage of this skew. Adding one highly-connected large-capacity node to the network would increase reliability while significantly decreasing all other nodes' load.

V. Related and Future Work

In [1], 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 open question is whether node ids can be assigned more intelligently to improve trustworthiness. That is, if identifiers were assigned to nodes based on the current ids of their connected friends, what algorithm or distribution for id assignment would optimize our ability to route over social links?

We would like to evaluate SPROUT in a system using replication. To illustrate, assume we use k replicas, and node A attempts to insert an item in the DHT and B searches for that item's key. If a message has an expected probability p of reaching its destination, then the probability of B discovering A's item is $1-(1-p^2)^k$. Using the values in Table I for p and $k = 3$, SPROUT would succeed 52% of the time compared to only 35% for AC.

Users may be willing to declare more friends if it would improve their performance. How many social links would each user need to maintain to reach a target average path rating?

With few modifications our model can be used to evaluate other issues, such as Quality of Service. If peers prioritized message forwarding based on service agreements and/or social connections we may want to use latency to compare routing algorithms. Using functions that give expected delay at each node in place of trust functions, and using an additive, instead of multiplicative, path rating function, we could express this appropriately. In [6] we explore this and other issues and demonstrate that SPROUT performs even better with respect to Chord in such systems.

VI. Conclusion

We have presented a method for leveraging the trust relationships gained by marrying a peer-to-peer system with a social network, and applied it to the problem of mitigating misrouting attacks. We described a model for evaluating routing algorithms is such a system and proposed SPROUT, demonstrating how it can improve successful routing in a system where a large fraction of the nodes are malicious.

VII. Acknowledgements

The authors would like to thank Orkut Buyukkokten for providing us with the Club Nexus data and Adam D'Angelo for providing us with the BuddyZoo data.

References

[1] Castro, M., Drushel, P., Ganesh, A., Rowstron, A., and Wallach, D. Secure routing for structured peer-to-peer overlay networks. In *OSDI '02* (2002).

[2] D'Angelo, A. BuddyZoo. http://www.buddyzoo.com.

[3] Douceur, J. R. The Sybil Attack. In *Proceedings of the International Workshop on Peer-to-Peer Systems* (2002).

[4] Friedman, E., and Resnick, P. The social cost of cheap pseudonyms. *Journal of Economics and Management Strategy 10*, 2 (1998), 173–199.

[5] Gummadi, K., Gummadi, R., Gribble, S., Ratnasamy, S., Shenker, S., and Stoica, I. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM* (2003).

[6] Marti, S., Ganesan, P., and Garcia-Molina, H. SPROUT: P2P Routing with Social Networks. Tech. rep., 2004. dbpubs.stanford.edu/pub/2004-5.

[7] Marti, S., and Garcia-Molina, H. Identity crisis: Anonymity vs. reputation in p2p systems. In *IEEE 3rd International Conference on Peer-to-Peer Computing (P2P 2003).*

[8] Puniyani, A. R., Lukose, R. M., and Huberman, B. A. Intentional Walks on Scale Free Small Worlds. *ArXiv Condensed Matter e-prints* (July 2001). http://aps.arxiv.org/abs/cond-mat/0107212.

[9] Ratnasamy, S., Francis, P., Handley, M., Karp, R., and Shenker, S. A scalable content addressable network. Tech. Rep. TR-00-010, Berkeley, CA, 2000.

[10] Rowstron, A., and 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.

[11] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D. R., Kaashoek, M. F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw. 11*, 1 (2003), 17–32.