Limited Reputation Sharing in P2P Systems[*]

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

ABSTRACT

The increasing popularity of resource exchange through peer-to-peer networks has encouraged the development of ways to support more complex commercial transactions over these networks. Unfortunately, the prospect of higher volume and higher value transactions attracts agents seeking to exploit or weaken the network by propagating bad information and services. This paper presents advantages and disadvantages of resource selection techniques based on peer reputation. We evaluate the effect of limited reputation information sharing on the efficiency and load distribution of a peer-to-peer system. We show that limited reputation sharing can reduce the number of failed transactions by a factor of 20.

Categories and Subject Descriptors

H.3.4 [**Information Storage and Retrieval**]: Systems and Software

General Terms

Algorithms, Performance, Economics, Experimentation, Security

Keywords

peer-to-peer, trust, reputation

1. INTRODUCTION

The increasing availability of high bandwidth Internet connections and low-cost computers has stimulated the use of resource sharing and exchange using peer-to-peer (P2P) networks. These systems employ a simple scalable mechanism that allows anyone to offer content and services to other users, as well as search for and request resources from the network. Many research groups and organizations (e.g. [3]

[12] [26]) are working to develop better transactional mechanisms for P2P systems, making them viable platforms for monetary or barter e-commerce.

However, the open accessibility of these systems make them vulnerable to malicious users wishing to poison the system with corrupted data or harmful services for personal or commercial gain. Because of this danger, users must be wary of the quality or validity of the resources or services they purchase through the network.

Determining the validity of a resource is a costly operation. Verifying the authenticity of the content of a file or document requires downloading it from the provider. In the simplest case, a digest may be available for the file from a trusted authority who "owns" the file. Checking the file involves calculating a hash and comparing it to the digest. But often, locating the resource's authority (if one even exists) is more difficult than locating the resource itself.

This problem has led to the development of reputation systems as a means to detect misbehavior and circumvent (or punish) malicious nodes. Peer-to-peer reputation systems collect information on the trustworthiness of resource providers and propagate it to improve peers' chances of locating good providers.

Reputation systems have proven effective at promoting consumer confidence in online shopping. For example, eBay's [8] reputation system discourages fraud because buyers will usually only bid on high-priced items from sellers whose reputation is high from numerous successful transactions. And Amazon.com's [1] consumer product reviews allow people to read the opinions of others who have purchased the item, alleviating the trepidation many have of purchasing products online. These solutions rely on a central trusted organization to maintain the reputation system. However, a reputation system for P2P networks must be distributed across the autonomous peers with no *a priori* trust relationships.

In studying the use of distributed reputation systems we will focus on how peer-to-peer networks function and what threats they face today. There are two classes of misbehavior in P2P networks: selfishness and maliciousness. Selfish nodes wish to use the resources of the network without offering any themselves. Reputation-based incentive schemes aim to discourage freeriders and other selfish nodes by only offering services to nodes that reciprocate ([16] [22]).

Other systems combat malicious nodes who do not care about access to other peers' resources, but only want to propagate their invalid resources. Such systems often employ a shared history of node interactions to signal possibly malicious resource providers that users should avoid. Eigen-

Trust [15], for example, collects statistics on node behaviors and computes a global trust rating for each node. Yet global history schemes are complicated, requiring long periods of time to collect statistics and compute a global rating. They also suffer from the transience of nodes and continual anonymity afforded malicious nodes through zero-cost identities.

In this paper, we study the performance of a peer-to-peer resource-sharing network in the presence of malicious nodes, which, in contrast to global history schemes, uses only limited or no information sharing between nodes. We develop various techniques based on collecting reputation information. We present several interesting side-effects relating to load-balancing and message traffic resulting from some of the techniques. Due to space limitations, this paper presents a subset of the issues we have studied. More information on these and other experiments is available in a technical report [20].

In Section 2 we present our system model and its assumptions. Then, Section 3 discusses the reputation systems used in the experiments and their options. Section 4 describes the metrics used for evaluating our experiments. In Section 5 we specify the details of the simulation environment used for the experiments, and present the results in Section 6. Section 8 discusses related work. Finally, we conclude in Section 9.

2. SYSTEM MODEL

A peer-to-peer system is composed of n peer nodes arranged in an overlay network. In a resource-sharing network each node offers a set of resources to its peers, such as multimedia files, documents, or services. When a node desires a resource, it queries all or a subset of the peers in the network (depending on the system protocol), collects responses from available resource providers, and selects a provider from which to access or retrieve the resource.

Locating a willing resource provider does not guarantee the user will be satisfied with its service. Selfish peers may offer resources to maintain the impression of cooperation, but not put in the necessary effort to provide the service. Worse, certain nodes may join the network, not to use other peers' resources, but to propagate false files or information for their own benefits.

In our model, each peer verifies the validity of any resource it uses. Accessing invalid or falsified resources can be expensive in terms of time and money. A system may implement a micropayment scheme requiring users to pay a provider before being able to verify the validity of the resource. In most cases the user must wait for a file to be downloaded or a remote computation to conclude and then verify the correctness of the result. Checking the validity of the file or service response may itself be a costly but necessary operation in the presence of malicious nodes. Because such an operation is highly domain-specific, we assume the existence of a global verification function, $V(R)$ which checks whether resource R provided by a peer is valid. Any node can perform this verification, but it is indeterminately expensive to compute and may require human interaction (such as listening to a song after downloading it from a music service to ensure it is the correct song and uncorrupted) or even a third-party. In addition, a resource must be downloaded or accessed before it can be verified, which costs time and bandwidth. We include this cost in the verification function, so that it represents the full price of accessing a bad resource. We do not explicitly consider the monetary cost of the resource, though it may be a component of the price.

To simplify the discussion we present our work in the context of a file-sharing system, where users query the network, fetch files from other peers, and verify the files' content is correct. Nodes hearing the query reply to the query originator if they have a copy of the file. The originator then fetches copies of the file from the responders until a valid, or *authentic* copy is located. Though we use the term "files" in the rest of the paper, most concepts apply to generic resources.

2.1 Threat Model

As stated above, the threat we are studying is that of a group of malicious nodes that wish to propagate inauthentic (or fake) copies of certain files. They do not care if they themselves are unable to query the system for files, thus incentive schemes fail to deter them. In addition, we assume they may pass false information to other nodes to encourage them to fetch bad files. We consider three behaviors for malicious nodes; abbreviated as N, L and C:

N: No misinformation is shared. All nodes give true opinions.

L: Malicious nodes lie independently for their own gain. They give a bad opinion of everyone else.

C: Malicious nodes collude. They give good opinions of each other and bad opinions of well-behaved nodes. For this model, we will briefly consider the situation where some malicious nodes act as "front" nodes by providing only authentic files (but never from the subversion set) in an attempt to gain the trust of other nodes and spread their malicious opinions.

Our threat model specifies a set of files, called the *subversion set*, that all malicious nodes wish to subvert by disseminating invalid copies. The percentage of nodes in the network that are malicious is given by the parameter π_B. This value includes not only the actual number of malicious users, but also all valid network identities the malicious users have attained in order to masquerade as multiple distinct nodes.

Each unique file has an equal probability of being in the subversion set, specified by the parameter p_B. We assume no correlation exists between a file's popularity and its likelihood to be targeted for subversion. Malicious nodes also share valid copies of files not in the subversion set. The effects on performance of varying both π_B and p_B are discussed in Section 6.2 and more information is available in [20].

We assume well-behaved nodes always verify the authenticity of any file they have before sharing it in the network. Though this assumption may be unrealistic for many peer-to-peer systems, experiments in which a small fraction of the files provided by good nodes were invalid demonstrated little effect on our experimental results.

We assume no other malicious activity, such as denial-of-service style attacks, are occurring in the network.

3. REPUTATION SYSTEMS

When a node queries the system for a file, it collects all replies (and their source IDs) in a *response set*. The node repeatedly selects responses from the set, fetches the copy of the file offered by the responder and verifies it (using the verification function) until an authentic copy is found.

As nodes interact with each other, they record the outcome, such as whether the file received was authentic of not. As a node collects statistics, it develops an opinion, or *reputation rating*, for each node. We make no assumptions of how this rating should be computed, but since it is used to compare and rank nodes, it should be scalar (see below for an example).

Each node records statistics and ratings in a *reputation vector* of length n, where n is the total number of nodes in the network. When a node first enters the system all entries are *undefined*. As the node receives and verifies files from peers, it updates the corresponding entry. Nodes may also share their opinions about other nodes with each other and incorporate them in their ratings. The reputation vectors can be viewed as an $n \times n$ *reputation matrix*, R, where the ith row is node i's reputation vector. Cell $R_{i,j}$ would contain node i's "opinion" of node j.

When a node has collected replies to a query, the reputation system calls a *selection procedure*, which takes as input the query response set and the node's reputation vector, and selects and fetches a file. The verification function is then calculated on the selected file. As stated earlier, this may be done programmatically if possible, but most likely requires presenting the file to the user. The system updates its statistics for the selected response provider based on the verification result. If verification failed, the selection procedure is called again with a decremented response set. This is repeated until a valid file is located, the response set is empty, or the selection procedure deems there are no responses worth selecting (such as if the remaining responders' ratings are too low).

For this paper we study variants on two reputation systems, one in which peers share their opinion and one in which only local statistics are used. They are compared against a random selection algorithm.

Random Selection: Our base case for comparison is an algorithm which randomly chooses from the query responses until an authentic file is located. Since no knowledge or state about previous interactions is stored, shared or used, this algorithm models the performance of a system with no reputation system.

Local Reputation System: With this reputation system each node maintains statistics on how many files it has verified from each peer and how many of those were authentic. Each peer's reputation rating is calculated as the fraction of verified files which were authentic. This results in a rating ranging from 0 to 1, with 0 meaning no authenticity check passed and 1 meaning all authenticity checks passed. When processing a query, these ratings are used in the selection procedure to select the peer from which to fetch the file. We consider two procedures in our experiments:

- The *Select-Best* selection procedure selects the response from the response node with the highest rating. If the selected response is invalid, the procedure chooses the next highest-rated node.

- Select-Best will prefer to choose good nodes it has previously encountered and thus may overload a small subset of reputable peers. To spread out file requests we propose the *Weighted* selection procedure, which probabilistically selects the file to fetch weighted by the provider's rating. For example, if nodes i and j both provide replies to node q and $R(q,i) = 0.1$ and

$R(q,j) = 0.9$, then j is nine times as likely to be chosen as i. We study load distribution in Section 6.3.

The Select-Best method requires a node maintain an ordered list of the most reputable nodes it knows. We call this list a *Friend-Cache* of maximum size $\overline{FC}$. There are additional benefits to maintaining a Friend-Cache in the local reputation system. By sending queries directly to nodes in the Friend-Cache before propagating the query normally, the message traffic of query floods in flat unstructured networks can be greatly reduced. We call this the *Friends-First* technique and evaluate it in Section 6.4.

Voting Reputation System: This system collects statistics and determines local peer ratings just as the local system does. It extends the previous system by considering the opinions of other peers in the selection stage. When a node, q, has received a set of responses to a query, it contacts a set of nodes, Q, for their own local opinion of the responders. Each polled node, or *voter* $v \in Q$, replies with its rating (from 0 to 1) for any responder it has interacted with and thus has gathered statistics. The final rating for each responder is calculated by the formula

$$\rho_r = (1 - w_Q)R(q,r) + w_Q \frac{\sum_{v \in Q} R(q,v)R(v,r)}{\sum_{v \in Q} R(q,v)} \qquad (1)$$

For each responder r, the querying node q sums each voter's (v) rating of r weighed by q's rating for v. This result is the *quorum rating*. If node q has no prior knowledge of r, it uses the quorum rating as r's rating in the selection procedure. If q already has statistics from prior interaction with node r, the rating for node r is the combination of the local statistics and the quorum rating, by some given weight called the *quorumweight*, w_Q. Note that when $w_Q = 0$ the voting system works exactly like the local system.

Until now we have not discussed how the nodes in the quorum Q are selected to give their opinion. We consider two methods of selecting voters. The first method is to ask one's neighbors in the overlay topology. These are typically the first peers a node is introduced to in the network and, though neighbors may come and go, the number of voters will remain relatively constant. The other method is to ask peers from whom one has fetched files and who have proven to be reputable. This group would consist of the peers with the f highest local ratings at node q. The former quorum selection we call *Neighbor-voting* while the latter is referred to as *Friend-voting*. In the Friend-voting scheme we reuse the Friend-Cache described above to maintain our list of voters. The cache has a maximum size of $\overline{FC}$. We study the effects of varying $\overline{FC}$ in Section 6.1.

Above, we describe the source node as contacting each voter for their opinion for each and every query once it has collected the responses. Realistically, nodes may instead periodically exchange reputation vectors with each other. If the rate at which reputation vectors are exchanged is as frequent as once per query, then the two methods are equivalent. For simplicity, we assume this equivalence in our simulator and model the system as acquiring voter opinions at the time of the query.

Both reputation systems have two additional parameters. An initial reputation rating ρ_0 is used for responding peers for which no statistics are available, or "new" nodes. In some domains it may be easy for malicious nodes to automatically generate fake responses to queries. Therefore, in

situations where a node is querying for a rare file, it may receive many replies, all of which are bad. To prevent the node from fetching every fake file and calculating $V(R)$, we introduce a *selection threshold* value (ρ_T). Any response from a node whose reputation rating is below this threshold is automatically discarded and never considered for selection.[1] This technique is used in both the local and voting-based reputation systems. We present an analysis of the effects of parameters ρ_0 and ρ_T in [19].

3.1 Identity

To collect statistics on peers, nodes must be able to identify and distinguish their peers over a period of time. Therefore, building reputation requires persistent node identities. Enforcing persistent identities conflicts with the goal of anonymity expounded by many P2P networks.

We discuss in detail certain viable identity models in [19] and study their impact on P2P reputation systems. Here we will consider two identity models derived from our previous work. The stricter model is equivalent to a trusted login server requiring one's real-world identity be tied to one unique system identity. This scheme ensures users cannot (easily) change identities to hide their misbehavior. In this system the trusted server need not know which system ID refers to which real-world ID [10].

The second model relies on users generating their own certificates and public/private key pairs as forms of identification. Though robust to spoofing, any user can easily discard an identity and generate a new one. Using self-managed identities makes the system vulnerable to *whitewashing*, where malicious nodes periodically change their identities to hide their misbehavior [16]. We conduct experiments using both identity models and distinguish the two as the whitewashing and static scenarios. In our results, the default identity model is the static model, unless whitewashing (WW) is specified.

4. METRICS

Here we present the metrics we use to evaluate our experimental results. We ran simulations of our system model for a period of time and gathered statistics at the end. These statistics are used to compute the metrics. They are summarized in Table 1.

From among all the queries generated during execution (q_{tot}) we are specifically interested in the number of successful queries (q_{succ}). A *successful query* is a query that results in an authentic copy of the requested file being selected and verified by the selection procedure.

4.1 Efficiency

When designing reputation systems our primary concern is to reduce the number of files which must be fetched and verified before locating a valid query response. During execution we record the number of file verifications supplied by each node i, which we refer to as V_i. From this data we compute the total number of verification function evaluations, V, as

$$V = \sum_{i=1}^{n} V_i \tag{2}$$

[1]New nodes are automatically exempt from being discarded, even if $\rho_0 < \rho_T$.

Table 1: Simulation statistics and metrics

Metric	Description
q_{tot}	# of queries generated
q_{succ}	# of successful queries where the selection procedure located an authentic file
V_i	# of verification function evaluations performed on files fetched from node i
n_G	Number of good nodes in the network
V	# of verification function evaluations
V_G	Total number of verification function evaluations of files fetched from good nodes
r_V	Verification ratio
ℓ_i	Load on node i
ℓ_G	Average load on good nodes
MT_{rel}	Relative message traffic of Friends First w.r.t. flooding

But V alone is insufficient. A system could ignore every response, report failure on every query, and have $V = 0$. To account for the fact that some systems may incur more verification checks, but locate valid files to more queries, we divide V by the number of successful queries (q_{succ}). We call this metric the *verification ratio* (r_V).

$$r_V = \frac{V}{q_{succ}} \tag{3}$$

The lower the value of r_V, the more efficient the system is. The best possible performance would be a prescient algorithm which always chose a valid file if one was available in the response set, and ignored all responses if not. This would give an r_V of 1. The verification ratio measures the *efficiency* of a reputation system and is our principal metric of system performance.

4.2 Load

We are also interested in measuring the load on the network under the various reputation systems and threat models. We are primarily concerned with the load on the well-behaved nodes in the network from file fetches. If each file is transferred only when it is selected to be verified, then the number of files a node has uploaded is equal to the number of verification function evaluations of files from that node. We define the load on node i (ℓ_i) as the number of verification checks on files it supplies normalized by the total number of queries, or

$$\ell_i = \frac{V_i}{q_{tot}} \tag{4}$$

We measure the average load on the network as the average load across well-behaved nodes, ℓ_G. Let G be the set of all good nodes in the network and let n_G be the total number of good nodes. Therefore, the average load is

$$\ell_G = \frac{\sum_{i \in G} \ell_i}{n_G} \tag{5}$$

Network load is analyzed in Section 6.3.

4.3 Message Traffic

To measure the message efficiency of the Friends-First method we compare the network query message traffic generated by this method to the default practice in unstructured

networks of flooding the network for each query. We calculate the relative message traffic (MT_{rel}) using the formula

$$MT_{rel} = \frac{\text{Number of Friends-First Messages}}{\text{Number of Flooding Messages}} \quad (6)$$

and compute it using the system parameters and the statistics gathered from the Select-Best experiments. Note that the number of Friends-First messages includes messages sent directly to friends and messages from query floods, resulting when the query goes unanswered by friends.

5. SIMULATION DETAILS

We evaluate the reputation systems using our own P2P Simulator based on our system model. The simulations were run on a Dual 2.4Ghz Xeon processor machine with 2GB of RAM. Each data point presented in the results section represents the average of at least 5 simulation runs with different seeds.

Though most of our findings apply to any peer-to-peer network, for our experiments we construct a Gnutella-like flat unstructured network. Specifying the overlay topology is necessary for studying certain issues, such as Neighbor-voting and message traffic reduction. Studies of unstructured peer-to-peer networks have shown their topologies are power-law networks [9]. We use randomly generated, fully connected power-law networks with $n = 1000$ nodes, a maximum node degree of $d_{max} = 50$ and an average node degree of $d_{avg} \approx 3.1$. We have experimented with larger networks of up to 10,000. Results are not shown due to space limitations but observed trends are similar to what is reported here. For some of these results see [19] and [20].

Queries are propagated to a TTL of 5. For simplicity we assume the network structure does not change, though we simulate a node leaving and a new node taking its place in the network. Each timestep a query is generated and completely evaluated before the next query/timestep. Therefore, a simulation run of 100 timesteps processes 100 queries. For the results dealing solely with the local reputation system, which does not exchange reputation information between peers, all queries are sent from a single node randomly chosen at startup. Each simulation seed selects a different node. For experiments using the voting-based system a node is randomly chosen as the query source at each timestep.

The simulation component most specific to file-sharing (as opposed to general resource-sharing) is our query model. It is similar to the one proposed in [25]. We assume a total of 100,000 unique files. The number of copies of each file in the system is determined by a Zipf distribution with $\alpha = 1.2$. Each node is assigned a number of files based on the distribution of shared files collected by Saroiu et al [23]. The query popularity distribution determines which file each query searches for. For this distribution we use a two-part Zipf distribution with an α of 0.63 from rank 1 to 250 and an α of 1.24. This distribution better models query popularity in existing peer-to-peer systems [24]. Though our query model is based on data collected on today's file-sharing networks, we expect networks providing other content or services to have similar distributions.

We model node turnover by having a random node leave the network and a new node enter on average once per query from a single node. Therefore, a turnover occurs every timestep for the single query source experiments and every 1000 timesteps for the multiple query source experiments

Table 2: System parameters, and default values

Param.	Desc.	Value
n	Number of nodes	1000
π_B	Frac. of malicious nodes	0.3
p_B	Frac. of docs in subversion set	0.9
ρ_0	Initial reputation rating	0.3
ρ_T	Selection threshold.	0.2
$\overline{FC}$	Size of Friend-Cache	-
w_Q	Weight given to voters' opinions w.r.t. local statistics	0.1

in a 1000 node network. For the reputation system, this is equivalent to clearing all information in the ith row and column of R, when node i leaves. For the whitewash experiments all malicious nodes change identity every 10 queries from a single node (or every 10000 queries with multiple query sources), by clearing all the columns of R relating to malicious nodes.

Based on our analysis in [19] we use a selection threshold of 0.2 in all experiments reported in this paper. We use an initial reputation rating of 0 for the whitewashing experiments, and 0.3 otherwise. Unless otherwise specified, the experiments were run with $\pi_B = 0.3$ and $p_B = 0.9$.

6. RESULTS

In this section we discuss the following important issues:

1. How well does the voting-based system perform? How do the parameters w_Q and $\overline{FC}$ affect performance?

2. How do the Select-Best and Weighted methods compare in terms of overall efficiency?

3. How does the reputation system affect the distribution of load across well-behaved nodes?

4. Can maintaining a Friend-Cache reduce message traffic?

Due to space constraints we present a concise analysis of a subset of our experimental results. A list of system parameters, their descriptions, and values used (unless otherwise specified) are given in Table 2. For more information on these and other related topics please refer to [20].

6.1 Voting System Parameters

In this section, we analyze the performance of the voting-based reputation system for various parameter values. All experiments were performed using the multi-source query generator for a total of 100,000 queries. For this scenario the random algorithm obtained an $r_V = 28.2$, off the scale of the graphs. The relative performance of the local reputation system is given by the data point for a quorumweight of 0.

Figure 1 presents the effects of varying the quorumweight, w_Q. It shows results for both with whitewashing (WW) and without, both Neighbor (Nbr) and Friend (Frd) voting, and both the selfish lying (L) and colluding (C) malicious opinion-sharing models. No (N) opinion-sharing misbehavior mirrored the L curves, performing only marginally better across all experiments, and are not graphed. In the selfish lying model, malicious nodes give themselves a rating of 1 and all others a rating of 0. Since malicious nodes cannot vote for themselves and give everyone else an equal rating
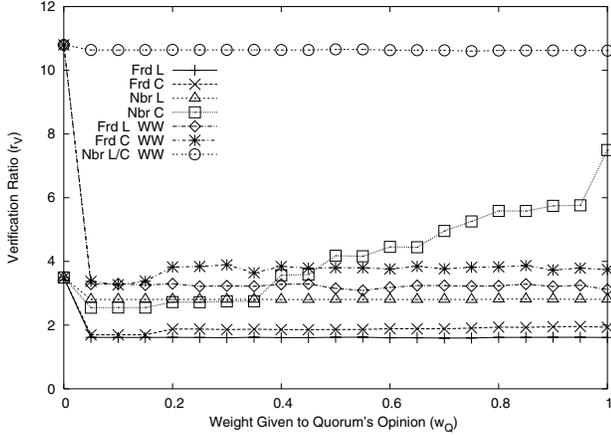
Figure 1: Efficiency of the voting reputation system (using Select-Best) with respect to varying quorumweight (w_Q). Lower r_V is better. 1 is optimal.

Figure 2: Efficiency of the voting reputation system with respect to Friend-Cache size ($\overline{FC}$).

of 0, they do not greatly impact a vote in favor of malicious nodes.

Note that the values of r_V in Figure 1 are relatively high. For example, an r_V value of only 3 means we would expect to download and verify three files for each query. In an actual system, it may not be feasible for a node to thoroughly check each downloaded file's authenticity. The node may simply trust the file to be valid. In this case, r_V can be viewed as the inverse probability that such a file is valid. Accounting for well-behaved nodes offering bad copies of files complicates the threat model. We have conducted experiments with this assumption and, as long as the probability of a good node offering a bad file is small, it does not noticeably affect our results.

Observing the drop in r_V from $w_Q = 0$ to $w_Q = 0.05$, we conclude that incorporating other nodes' opinions tends to improve the efficiency of the system. Except when malicious nodes collude to subvert the voting process, varying the weight of the voters opinions beyond $w_Q = 0.05$ has no effect on the system performance. This behavior indicates that the greatest benefit from voting is in the situation where the local node has no opinion of their own. When bad nodes collude (C), system performance decreases as the weight given to the quorum's opinion increases, reinforcing that there is no substitute for personal experience in an untrusted environment.

Comparing the Frd family of curves to the Nbr curves within the same whitewash scenario (e.g. Frd L vs. Nbr L), we clearly see that Friend-voting outperforms Neighbor-voting. Nodes that have given you good service in the past have demonstrated some effort to be reliable and well-behaved. Asking them for their opinions is more reasonable than relying on one's neighbors, a third of which, in this scenario, are likely to be bad. Not only does Neighbor-voting not perform as well, but it is more susceptible to malicious collusion as neighbors' opinions are given more weight (see Nbr C curve). Friend-voting, however, tends to avoid asking malicious nodes for their opinions, mitigating the effects of collusion.

Though the whitewash scenario performs worse than no whitewashing, it can benefit more from opinion-sharing. As the Frd WW curves between $w_Q = 0$ and $w_Q > 0$ illus-
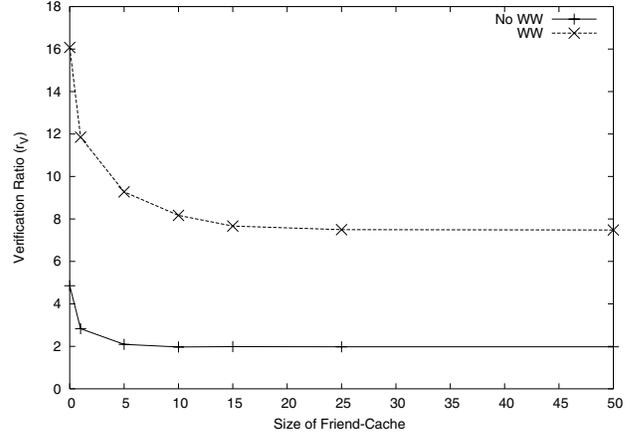
trate, efficiency for Friend-voting improves by a factor of 3 over the local reputation system. The Nbr L/C WW curve shows that Neighbor-voting in the WW scenario is almost completely unaffected by opinion-sharing, no matter the malicious opinion-sharing model. As stated before, in the WW scenarios an initial reputation rating of 0 is assigned to unknown nodes. Since this value is used for weighing the opinions of the voting nodes, any unknown peer in the neighbor quorum (including malicious nodes that have whitewashed) will have their votes ignored. Because the average number of neighbors is small (approx. 3.1) the probability of a well-behaved neighbor providing a query response that is tested, and thus becoming "known" and having their opinion used, is rare. In contrast, in the no WW scenario, since $\rho_0 = 0.3$, even untested peers' opinions are considered, explaining its poor performance when bad nodes collude.

In summary, this experiment shows that choosing a relatively small quorumweight around 0.1 with Friend-voting improves performance by a factor of 2 or more across all scenarios. But how many reputable nodes should one keep in the Friend-Cache? Does increasing the size of the Friend-Cache always result in better efficiency? In a real system, a larger cache means greater maintenance cost periodically checking the liveness of the nodes in the cache. Is this cost always justified?

Figure 2 shows the performance of both the whitewashing and no whitewashing scenarios for various Friend-Cache sizes ($\overline{FC}$) with no bad opinion-sharing (N).[2] Both scenarios stabilize so that increasing the size of the cache yields no performance improvement, but a system dealing with whitewashing benefits from a larger cache. For instance, while a Friend-Cache of 10 is sufficient when there is no whitewashing, the whitewash scenario can benefit from a cache as large as 25. As expected, when tested with the malicious opinion-sharing models (N, L, C), all three models produced similar r_V values, with the C values being slightly greater than that of the N and L values by about 0.4 in the no WW scenario. Thus, we only plot the N curve in Figure 2. A surprisingly small cache is needed for this technique to be efficient. We ran this experiment with a variety of network sizes, ranging from 100 to 5000 nodes and found very little variance in the

---

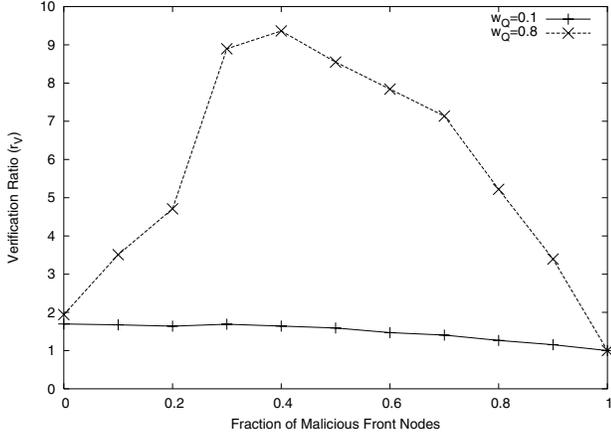[2]$\overline{FC} = 0$ corresponds to the local reputation system

Figure 3: Effects of front nodes on efficiency.

(a) Voting system

(b) Local system

Figure 4: Comparison of the efficiency of the two reputation systems with the random algorithm as a function of π_B.

shape of the curve or the point at which it stabilizes. Therefore, the Friend-Cache does not need to grow linearly with the size of the network to give best performance.

Friend-voting is effective against collusion because it only considers the opinions of nodes that have shown to behave well by providing good files. Given our threat model, this quickly bars malicious nodes from the Friend-Cache. One technique malicious nodes may employ to defeat Friend-voting would be to set up *front nodes*. These nodes properly trade only authentic files, but when asked for their opinion of other nodes, act according to the collusion model, C, promoting only malicious nodes.

We have run simulations where a fraction of the malicious nodes are set to be front nodes. We present the results for both a quorumweight of 0.1 and 0.8 in Figure 3. These experiments show that, in the case of $w_Q = 0.8$, front nodes can cause considerable harm to the system. The damage peaks when 40% of the malicious nodes are front nodes, decreasing the system performance by more than a factor of 3! For a larger number of front nodes, r_V steadily drops, indicating that too many malicious nodes are behaving well to promote a smaller group causing actual damage. To be optimally effective, attackers would need to use the right balance of front nodes and actively malicious nodes. Surprisingly, front nodes appear to have no adverse effect when $w_Q = 0.1$. We believe this shows that a very low quorumweight limits the impact of front nodes' bad opinions sufficiently that the damage caused by front nodes is negated by the benefit of having fewer actively malicious nodes.
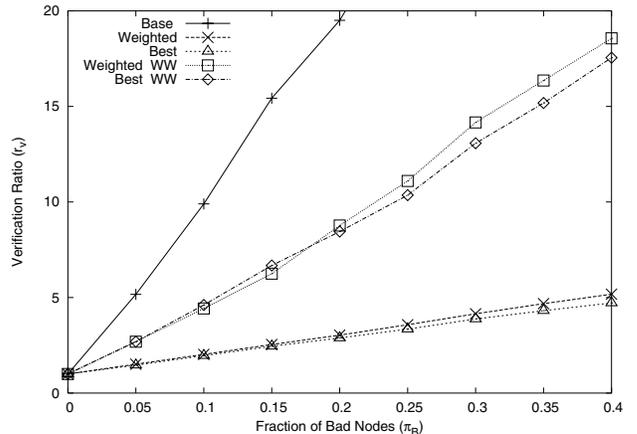
6.2 Efficiency Comparisons

Given the results of our analysis on the voting parameters, we wish to evaluate the system with respect to varying threat parameters. Specifically, we demonstrate how overall efficiency is affected by varying the percentage of malicious nodes in the system (π_B). We have run similar experiments varying the probability of a unique file being in the subversion set (p_B) and obtained similar results and performance comparisons.

We test the voting system with $w_Q = 0.1$ and $\overline{FC} = 10$, and using the two selection procedures both with and without whitewashing. Malicious nodes did not lie or collude with their opinions (N). These experiments were also run using the multi-source query generator for 100,000 queries.
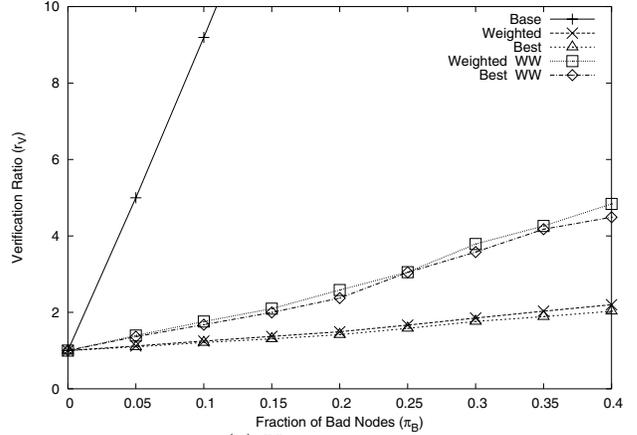
We evaluate the efficiency of the reputation systems for values of π_B between 0 and 0.4 using the default p_B of 0.9.

It may seem unlikely that a network would have 40% malicious peers attacking 90% of the files. However, in the real world, there are large entities, with access to vast resources, which have an interest in subverting peer-to-peer networks. We have simulated across several degrees of malicious activity (varying both π_B and p_B) and the relative performance of the different reputation system variants is comparable in weaker threat scenarios to those presented here.

Figure 4(a) shows the performance of the voting reputation system. Clearly, using any statistics when selecting a provider results in significantly better efficiency than purely random selection (base case). While the base case climbed to 42.5 at 40% malicious nodes, the voting reputation system attained an efficiency of 2 (with no whitewashing), a factor of improvement of 21! Whitewashing adversely affects the performance of the system, but not as badly as expected. For example, with a verification ratio of 4.5 the reputation system in the whitewash scenario performs 2.3 times worse than when there are no whitewashers. This means that on average a node would have to fetch more than twice as many copies of a file before finding a valid one, showing a clear ad-

vantage to preventing whitewashing by requiring users to log in through a trusted authority that can verify each real user has only one system identity.

We also executed the experiments using the local reputation system under equivalent conditions (100 queries from a single querying node). The results, shown in Figure 4(b), were only a factor of 2 worse performance than the voting system in the non-whitewashing scenario.[3] The performance difference between the two systems was greater in the whitewashing scenario, a factor of 4. These results support our findings in Section 6.1 that opinion-sharing is worthwhile in spite of its slightly higher implementation complexity.

When comparing the performance of the Select-Best (*Best*) and Weighted selection procedures in either graph of Figure 4, we see no large efficiency advantage of one procedure over the other, though the Select-Best method outperforms Weighted across all values of π_B. As expected, selecting the best known provider is slightly more efficient than probabilistically choosing a provider, but this comes at a cost, which we discuss in the following section.

6.3 Load on Good Nodes

One critical issue is that reputation systems may unfairly burden some of the good nodes in the network. We now look at the amount of load placed on well-behaved nodes in the network in terms of the number of files they upload. We are interested only in the effect produced by requests from well-behaved nodes running the algorithms correctly. We use the same setup as above with no whitewashing.

In an ideal system with no malicious nodes, we would expect exactly 1 download per query, giving a value of $\ell_G = 0.001$ for a 1000 node network. In our case, when there are no malicious nodes, the value of ℓ_G is 0.00098. This value is less than ideal because a few queries go unanswered by any node in the network.

As the fraction of malicious nodes increases, so does ℓ_G. For instance, when $\pi_B = 0.3$ the average load is 0.00138. With only 70% as many good nodes to service requests, we would expect $\ell_G = \frac{1}{0.70 \cdot 1000} = 0.00143$. Both the fact that malicious nodes provide some good files, and that the probability of a successful query is lower, account for the difference between the observed and expected loads. Comparing the two selection procedures shows an insignificant difference in average load. Both procedures fetch the same number of files from good nodes overall.

Although there was little difference between the selection procedures in terms of average load, it is important to consider the load distribution. In a homogeneous network where all nodes have similar bandwidth, it is preferable if load is distributed evenly across all nodes, as opposed to a few nodes handling most of the traffic while the majority are idle. To study load distribution we measured the load (using Eq. 4) on each individual node using the two voting-based reputation system selection procedures and compared them to the random selection algorithm. The values were then sorted in descending load order. The results, averaged across 10 runs with different seeds, are shown by the three line curves on the left y-axis in Figure 5(a). Here we see that, using the Select-Best selection procedure, the most heavily loaded node (with rank 1) has a load of almost 0.015. This value is more than 10 times the average load of 0.00138.

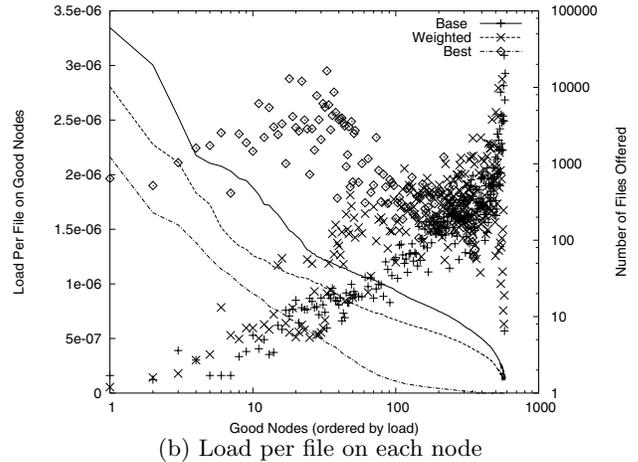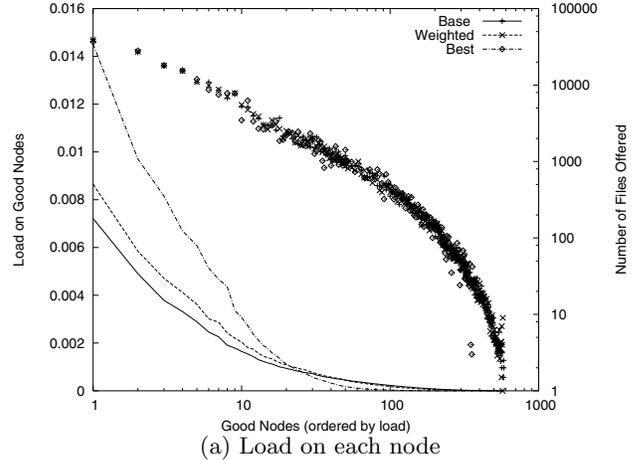Though both selection procedures incurred greater load

(a) Load on each node

(b) Load per file on each node

Figure 5: Distribution of load on good nodes (and their corresponding number of files shared). The x-axis corresponds to nodes sorted by amount of load in (a) and load per file stored on node in (b) (note logscale axis). The curves relate to the left y-axis and specify the amount of load measured at each node. The points map to the right y-axis and indicate the number of files on the corresponding node.

on the highest ranked nodes than the base case, Select-Best concentrated the load on a few nodes while Weighted distributed the load better. The maximum load on a node with the Select-Best method was almost twice that of Weighted. This is expected since Select-Best locates a few good nodes and tries to reuse them when possible, while the Weighted model encourages fetching files from new nodes (broadening its pool of known good nodes). If load-balancing in a homogeneous system is an important requirement, then the Weighted selection procedure would be preferable.

Another factor to consider is how load relates to the number of files shared by each node. It would be expected that good nodes with more files are more likely to be able to answer queries, increasing the number of files they upload and thus their load. Figure 5(a) plots as points the number of files on each good node on the right-hand y-axis. For exam-

---

[3]Note the difference in scale between the two graphs in Figure 4.

ple, for rank 1, there are three points around 38,000. This means that, for all three systems, the most heavily loaded node shared an average of around 38,000 files. As expected, all distributions show a strong correlation between nodes sharing more files and higher load.

In Figure 5(b) we divide the load on each node by the number of files it provides and reorder the distribution. For instance, the node at rank 1 has a load *per file* of 2.8×10^{-6} for the Weighted selection procedure, but only 2.2×10^{-6} for the Select-Best procedure. The result is surprising. The Select-Best method generated much less load per node than the Weighted or random methods. To understand this result we again plot the number of files offered by each node on the right y-axis. Here we see two trends. The base case and the Weighted method both curve from the bottom left upwards, showing that the nodes with highest load per file offer very few files. This effect is due to the sublinearity of the answering power of a node with respect to the number of files it is offering. For example, if node i has twice as many files as node j, we expect node i to be able to answer less than twice as many queries as j. In general, given a probability p that any individual file in the system matches a query, the probability that a node with f files can respond to a query equals $1 - (1 - p)^f$. In a purely random selection model this probability is an indicator of the expected load on a node; as f increases, so does the probability, and thus the likely load. This is corroborated by our results in Figure 5(a). Now if we divide this probability by f we have an indicator for the load per file: $\frac{1-(1-p)^f}{f}$. This equation has a maximum value when $f = 1$ and decreases as f increases. This explains the behavior we see from the random base case and the Weighted case in Figure 5(b).

The Select-Best method, on the other hand, shows a different trend. The most heavily loaded (per file) nodes share a very large number of files. The Select-Best procedure selects nodes which have proven reliable in the past. This behavior favors well-behaved nodes which respond to queries early in the simulation and often, nodes sharing many files. This procedure gives nodes with many files an even greater chance of being chosen with respect to the random model.

Whether or not it is desirable to send greater traffic to nodes with more files is dependent on the environment. Some have suggested that in some peer-to-peer systems, the number of files a node offers correlates to its available bandwidth. If so, using the Select-Best selection procedure, which gives preference to nodes with more files, may result in more effective bandwidth usage. But if peers have similar resource constraints or fair load-balancing is a priority, then we would prefer the Weighted selection procedure, which better equalizes load yet is almost as efficient at locating authentic files.

6.4 Message Traffic

Finally, we present our experiments on mitigating message traffic using the Friends-First technique. As explained earlier, Friends-First takes advantage of the Friend-Cache to try and locate a positive query response among the known reputable nodes, before querying the entire system. As we will see, in a flood-based querying system, this can result in 85% less message traffic!

Before presenting the results, we redefine the general formula for relative message traffic, given in Equation 6, in terms specific to our model. The numerator is the total message traffic for Friends-First. For all queries, messages are
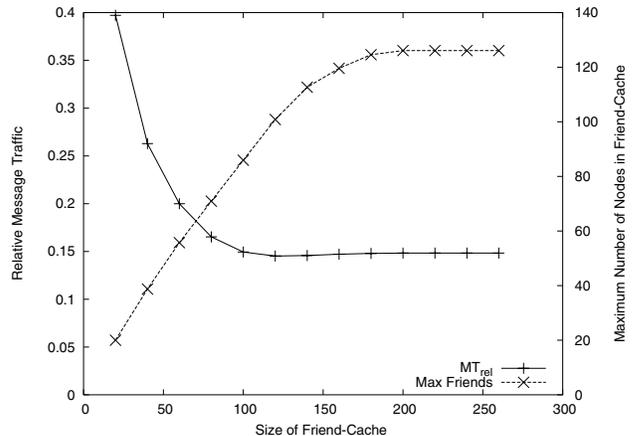

Figure 6: Relative message traffic of Friends-First and maximum Friend-Cache utilization as a function of the size of the cache.

sent to all nodes in the Friend-Cache ($q_{tot} \cdot \overline{FC}$). In addition there is the cost in messages of flooding the network when a valid response is not located from the Friend-Cache. The number of messages generated in the network to propagate a query will be at least equal to the number of nodes which hear the query, and most likely much larger due to several occurrences of two nodes forwarding the query to the same node. We roughly estimate the number of messages generated by a query flood as the average number of nodes reached by a query flood ($\overline{n}_{fld}$). Therefore, the additional cost of flooding for Friends-First would be the number of queries *not* answered by a node in the Friend-Cache ($q_{tot} - q_{FC}$) times $\overline{n}_{fld}$. The denominator is the number of messages generated assuming every query is a flood ($q_{tot} \cdot \overline{n}_{fld}$).

Note that $\overline{FC}$ is greater than or equal to the actual number of nodes in the Friend-Cache at any time, so not all queries will have $\overline{FC}$ nodes to query directly. Let $\overline{FC}_i$ be the number of nodes in the Friend-Cache after $i-1$ queries. $\overline{FC}_i$ is the number of messages sent directly to reputable nodes for the ith query. Note that for all i $\overline{FC}_i \leq \overline{FC}$ and $\overline{FC}_1 = 0$ since all nodes are initially unknown. We can define our message traffic metric as

$$MT_{rel} = \frac{\sum_{i=1}^{q_{tot}} \overline{FC}_i + (q_{tot} - q_{FC})\overline{n}_{fld}}{q_{tot} \cdot \overline{n}_{fld}} \qquad (7)$$

Note that this is still a conservative calculation of relative traffic since $\overline{n}_{fld}$ is less than or equal to the total number of messages generated due to a query flood.

We conducted these experiments using the local reputation system and the single-source query generator. For the results in this section, we ignored whitewashing and node turnover. We ran simulations for various numbers of queries (1000, 10,000, 50,000, etc).

The solid line in Figure 6 plots the relative message traffic of Friends-First with respect to regular flooding (MT_{rel}) as a function of the maximum Friend-Cache size, after 50,000 queries. We see that, as the size of the Friend-Cache increases, the query message traffic drops quickly to approximately MT_{rel}=0.15 until it reaches a point where growing the cache no longer provides any benefit. This means that the Friends-First method is generating only 15% as much message traffic as flooding without any loss in effectiveness!

Interestingly, for cache sizes greater than 120, the traffic overhead actually *increases* slightly, before levelling off at around 200. For small $\overline{FC}$, increasing the cache size greatly reduces message traffic because of the high likelihood of locating future query answers at the additional nodes stored in the cache. Every additional query satisfied by a node in the cache saves the system a query flood, outweighing the cost of the additional messages sent to the new nodes in the Friend-Cache for every query. But when $\overline{FC}$ is large, any node added to the cache will likely be sharing few files (thus rarely provide a response in the future). If the node had more files, it would have been located earlier and already be in the Friend-Cache. We find that well-behaved nodes sharing many files tend to be located quickly and be placed in the Friend-Cache early. Nodes added later offer fewer (approx. 5) files and do not offer any more query responses, thus wasting bandwidth on query messages sent directly to them.

As stated earlier, we performed experiments for varying lengths of time. In our shorter simulations (e.g. 1000 queries) there were no rise in relative traffic for large $\overline{FC}$. Instead MT_{rel} drops quickly and levels off, with no single minimum. These shorter simulations end before the Friend-Cache begins collecting useless nodes with very few files. Runs of 20,000 and 100,000 queries, on the other hand, also showed a preferred $\overline{FC}$ around 130. This result supports our hypothesis that, once only small nodes remain outside the cache, adding a node to the cache increases overall traffic because the cost of sending them a direct query outweighs the slim probability of their answering a request and avoiding a query flood.

In studying the efficiency of Friends-First, it is useful to consider the utilization of the Friend-Cache. The right y-axis of Figure 6 corresponds to the number of reputable nodes in the Friend-Cache when the simulation ended, represented in the graph by the points on the dashed line. Notice that the number of nodes in the cache increases linearly until MT_{rel} reaches the minimum, and levels off when MT_{rel} levels off. Interestingly, the value it reaches is 126, approximately the same value as the optimal cache size. We believe this is not a coincidence. This value is an average of several simulation runs with different seeds. Some runs had lower values and others higher, but it does indicate that, on average, the system did not use responses from more than 130 reputable nodes. Any peers entering the cache after the first 130 are highly unlikely to provide any further useful and unique responses. Therefore, limiting the Friend-Cache to a size of 130 prevents useless nodes from entering the cache and worsening performance.

In Section 6.1 we used the Friend-Cache to choose peers from which to gather opinions for response selection. We saw that there was little benefit from gathering opinions from more than the 10 or 15 most reputable peers. Yet the traffic results indicate that we can take advantage of Friend-Caches larger than 100. Should we use our entire large cache for gathering opinions? No. Though a large Friend-Cache is easy to maintain (it is a list of known nodes ordered by their reputation statistics), asking a large number of nodes to share their opinions, either per query or periodically, will greatly increase the amount of message traffic produced yet not improve our selection performance. Thus, though we may maintain a large Friend-Cache for direct querying, we would only ask the top nodes to participate in our quorum.

7. DISCUSSION

With our experiments we have studied the benefits and costs of our voting reputation system using a variety of different metrics. The results show that even dynamic peer-to-peer systems benefit from limited opinion-sharing in terms of efficiency in locating good results. The cost of implementing the voting system is the additional messages sent to share opinions on other peers and maintaining ordered reputation statistics in a Friend-Cache. The messages can be limited to periodically sending deltas of a node's sparse reputation vector. In addition, the Friend-Cache has beneficial side effects of decreasing both query traffic and latency.

As expected, high rates of whitewashing significantly decrease the effectiveness of reputation systems. For e-commerce systems it may be necessary to impose a large cost of participation [17] or strongly tie network identities to real-world identities [10]. High turnover likewise limits the effectiveness of reputations systems as reputable peers disappear quickly. We hope to study turnover rates in more detail in future work.

For simplicity we have considered all files equal. In a real e-commerce system this would likely not be the case. Each transaction has some value which may differ widely from one to another. The value of a transaction will likely determine the amount of risk the user will accept. In our system this may translate to dynamically changing the quorumweight, the quorum size, or the selection threshold to adapt to the cost risk of the current transaction.

For further discussion on these and other related experiments please refer to the extended technical report [20].

8. RELATED WORK

Extensive research has been done on general issues of reputation (eg. [13] [14] [18]). Much work has been done in the area of locating reputable nodes in resource-sharing peer-to-peer networks and many interesting reputation systems have been proposed (eg. [6] [16] [11]). Here we describe a few related examples.

In [16], Lai et al. propose a reciprocative incentive strategy to combat freeriders, based on the Evolutionary Prisoners Dilemma [2]. They compare the performance of private history versus shared history and develop an adaptive stranger response strategy that balances punishing whitewashers with overly taxing new nodes.

Reference [15] presents EigenTrust, a system to compute and publish a global reputation rating for each node in a network using an algorithm similar to PageRank [21]. Reputation statistics for each node are maintained at several nodes across a content-addressable network to mitigate the effects of bad nodes colluding.

In [5], Damiani et al enhance their previous work on reputation [4] by proposing the concept of resource reputation, where peers give opinions on a resource's authenticity based on its reported digest. This technique complements the process of maintaining peer reputations, which is still necessary in situations where the resource is rare and no other peers have encountered it.

9. CONCLUSION

We have presented a simple voting-based reputation system that significantly mitigates the deleterious effects of malicious nodes, by sharing information with a small group of

nodes. Even with 40% of the network attempting to subvert 90% of the resources, a node would expect to only have to attempt twice before locating a good provider, though it increases to four or five tries if the system is vulnerable to whitewashing. In addition, system performance is largely unaffected by malicious users using some of their network identities to distribute legitimate files in an attempt to promote accessing nodes which are providing bad files.

We compared two methods for selecting providers given reputation information and showed that, while one provides better efficiency, it also significantly skews the load on the well-behaved nodes in the network. Depending on the amount of heterogeneity in the network this may be acceptable.

Finally, we showed how the Friend-Cache developed for the reputation system can be applied to significantly reduce message traffic in unstructured peer-to-peer networks.

The possible applications of P2P reputation systems towards e-commerce are considerable. As peer-to-peer networks are increasingly used as an efficient medium for the exchange of valued goods or money, the impetus for users to exploit the system will also increase. Reputation systems have succeeded at maintaining consumer confidence in centralized online trading systems, such as eBay [8]. They will undoubtedly be a component of any successful P2P exchange system where users risk losing money, not just idle bandwidth and CPU.

10. REFERENCES

[1] Amazon.com. http://www.amazon.com/.

[2] R. Axelrod. *The Evolution of Cooperation*. Basic Books, 1984.

[3] B. F. Cooper and H. Garcia-Molina. Peer to peer data trading to preserve information. *ACM TOIS*, April 2002.

[4] F. Cornelli, E. Damiani, and S. D. Capitani. Choosing reputable servents in a p2p network. In *Proc. of the 11th International World Wide Web Conference*, 2002.

[5] E. Damiani, D. C. di Vimercati, S. Paraboschi, P. Samarati, and F. Violante. A reputation-based approach for choosing reliable resources in peer-to-peer networks. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 207–216. ACM Press, 2002.

[6] R. Dingledine, M. J. Freedman, D. Hopwood, and D. Molnar. A reputation system to increase MIX-net reliability. *Lecture Notes in Computer Science*, 2137:126+, 2001.

[7] J. R. Douceur. The Sybil Attack. In *Proc. of the International Workshop on Peer-to-Peer Systems*, 2002.

[8] eBay - The World's Online Marketplace. http://www.ebay.com/.

[9] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *SIGCOMM*, pages 251–262, 1999.

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

[11] M. Gupta, P. Judge, and M. Ammar. A reputation system for peer-to-peer networks. In *ACM 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 2003.

[12] B. Horne, B. Pinkas, and T. Sander. Escrow services and incentives in peer-to-peer networks. In *Proceedings of 3rd ACM Conference on Electronic Commerce*, 2001.

[13] B. A. Huberman and F. Wu. The dynamics of reputations. www.hpl.hp.com/shl/papers/reputations/, 2002.

[14] R. Jurca and B. Faltings. Towards incentive-compatible reputation management. In *Proceedings of the AAMAS 2002 Workshop on Deception, Fraud and Trust in Agent Societies*.

[15] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[16] K. Lai, M. Feldman, I. Stoica, and J. Chuang. Incentives for cooperation in peer-to-peer networks. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[17] P. Maniatis, M. Roussopoulos, T. Giuli, D. S. H. Rosenthal, M. Baker, and Y. Muliadi. Preserving peer replicas by rate-limited sampled voting. In *19th ACM Symposium on Operating Systems Principles (SOSP 2003)*, 2003.

[18] R. Marimon, J. Nicolini, and P. Teles. Competition and reputation. In *Proceedings of the World Conference Econometric Society*, 2000.

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

[20] S. Marti and H. Garcia-Molina. Examining metrics for reputation systems (in progress). Technical report, 2003. dbpubs.stanford.edu/pub/2003-39.

[21] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[22] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster. To share or not to share: An analysis of incentives to contribute in collaborative file sharing environments. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[23] S. Saroiu, P. K. Gummadi, and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN '02)*, San Jose, CA, USA, January 2002.

[24] K. Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. Featured on O'Reilly's www.openp2p.com website, February 2001.

[25] B. Yang and H. Garcia-Molina. Comparing hybrid peer-to-peer systems (extended). Technical report, 2000.

[26] B. Yang and H. Garcia-Molina. Ppay: Micropayments for peer-to-peer systems. In *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS)*, 2003. Washington D.C.