Cornell Technical Report cul.cis/TR2005-1980

Thwarting P2P Pollution Using Object Reputation

Kevin Walsh and Emin Gün Sirer

Cornell University, Ithaca, NY

Abstract

This paper describes a distributed object reputation management scheme that counteracts content pollution in peer-to-peer filesharing systems. The proposed scheme allows honest peers to assess the authenticity of online content by securely tabulating and managing endorsements from other peers. We employ a novel voter correlation scheme to weight the opinions of peers, which gives rise to favorable incentives and system dynamics. We present simulation results indicating that our system is scalable, efficient, and robust to attack.

1 Introduction

By all measures, peer-to-peer filesharing has become a significant feature of the Internet. Many existing networks, however, are rife with *pollution* in the form of corrupt or mislabeled content, and recent studies indicate that much of this is deliberate. Pollution makes it difficult to find sought objects, and can be a major security concern if if the polluted object caries a virus.

In this paper we introduce a robust and decentralized system for evaluating the reputation of objects. Our goal is to enable peers to confidently gauge *object authenticity*, the degree to which an object's data matches its advertised description. First, we outline a simple network-wide voting scheme, where users contribute positive and negative evaluations of objects. Second, we enable clients to weigh votes according to the statistical correlation between the client and its peers. And third, we allow clients to extend the scope of their correlations through selective information sharing. Our system is simple and effective, and can be deployed incrementally in existing structured and unstructured filesharing systems.

2 System Model

We consider peer-to-peer networks in which clients search for objects using queries over meta-data. Here, an object consists of *data*, and an associated *descriptor*, containing a unique identifier and serving as a pointer to the data. The descriptor also contains meta-data to facilitate searching, such as the object name, encoding, and description. A client issues queries to its peers in the network in the form of keywords. Peers respond by sending matching object descriptors back to the client.

At this point, the client must make a judgment about the authenticity of each descriptor, before attempting to fetch the associated data. In many networks, however, some of the descriptors may be polluted, and contain invalid meta-data or point to corrupt or malicious data. Lacking any sound basis on which to make a judgment, the client is often forced to use random selection, or ad-hoc indicators of object popularity, such as the frequency with which each descriptor was encountered. Our system addresses content pollution by enabling the client to make an informed decision about which descriptors are likely authentic, and by providing incentives for peers to contribute honestly in this evaluation process.

2.1 Endorsements

The basic mechanism in our system is a simple weighted voting scheme in which any client may vote positively or negatively on any object. A vote is a pair $(\langle objectID, value\rangle_K, cert_K)$ containing a cryptographic signature, under the client's key K, of the object identifier and a vote value from the set $\{-1,+1\}$, and a certificate of authentication for key K. Client keys and certificates need not be bound to real-world identities, but instead may rely on anonymous pseudonyms, such as those employed in many popular peer-to-peer networks. The certificate and signature allow other peers to verify the au-

thenticity and uniqueness of the vote.

A client interprets a positive vote as an endorsement of the object's authenticity (i.e., that the object data matches its descriptor). Since object descriptors in practice typically contain only factual and easily verifiable information, we assume that a large fraction of honest clients will generate votes of equal value for a given object. Under this assumption, a client judges the authenticity of an object by estimating its reputation among the client's peers.

2.2 Vote Evaluation

Votes are collected, evaluated, and aggregating by a client wishing to estimate the reputation of a given object descriptor. Although votes are generated with only one of two possible values, they provide varying levels of assurance to different clients. Unweighted voting is not sufficient in this context, since votes may come from untrustworthy or unknown peers. Worse, it would provide an incentive for an attacker to vote dishonestly on all objects since there would be no penalty for dishonesty.

From a client's perspective, the weight of a peer's vote depends most directly on the relationship between client and peer, and so the client weighs votes according to the observed strength and bias of this relationship. Intuitively, two peers that tend to vote identically (or inversely) on objects should develop over time a strong positive (or negative) weight for each other's votes, while peers having uncorrelated voting histories should essentially disregard each other's votes.

Statistical correlation captures precisely this notion of the historical relationship between a pair of peers. We compute a coefficient r_{AB}, using the method of Phi correlation, for two peers A and B as follows. Given a set of n objects on which both A and B have voted, let A_+ and B_+ be the fraction where A and B voted positively, respectively, and similarly let $(AB)_{++}$ be the fraction where both voted positively. Then $r_{AB} = ((AB)_{++} - A_+B_+) / \sqrt{A_+(1-A_+)B_+(1-B_+)}$ is the *coefficient of correlation*, which takes on values in the range $[-1,1]$. Positive values indicate that A and B tend to agree, negative that they tend to disagree, and $|r_{AB}| < 0.5$ indicating weak or no correlation.

2.3 Voting Protocol

We now describe the protocol a client A uses to estimate the reputation of a given object. First, client A issues a *vote-gather* query to collect votes on the object and, as with regular queries, some number of peers will respond after inspecting their local state. Each response contains a subset of the votes known to the responding peer, potentially including the peer's own vote. The impact of this query on the network is bounded by having each peer sub-sample its vote information, rather than sending all known votes. The sub-sampling is biased in favor of votes having the highest weight (from the peer's perspective), in order to disseminate the most useful votes further in the network. In practice, the vote-gather query is implemented entirely using the underlying primitives of the network, and can partially be combined with the initial user query in a single round of interactions.

Once client A has obtained a set of votes for the object, each is cryptographically verified, stored locally for later use, and then tabulated using a weighted average. The weight for a vote from peer B is r_{AB}, the correlation between A and B, computed from information gathered during earlier rounds of the protocol. The client interprets the resulting average v as a personalized estimate of the reputation, and hence authenticity, of the object and so can make a more informed decision to accept (and fetch) or reject the object. Also, having locally cached the votes collected during the vote-gather query, client A is now able to respond to vote-gather queries for this object from its peers. Votes are shared regardless of if client A ultimately accepts or rejects the object, and regardless of whether A's vote, if any, agrees or disagrees with the votes gathered. We next describe how this caching takes place, and detail the state each client maintains in order to execute the voting protocol.

2.4 Client Local State

Conceptually, each client maintains a *vote database* containing votes it has encountered. This database serves both as a store of information from which to respond to vote-gather queries, and as a dataset from which to compute peer correlations. The database consists of, for each object, a row for containing a timestamp, the client's own vote, if any, and a list all other votes encountered for the object. Each time the client issues a vote-gather query, the entire set of collected votes is cached in the vote database. The size of a client's local state is bounded by retaining only the most recent additions to the database, and by subsampling during vote collection.

Peer correlations are stored in a separate *correlation*

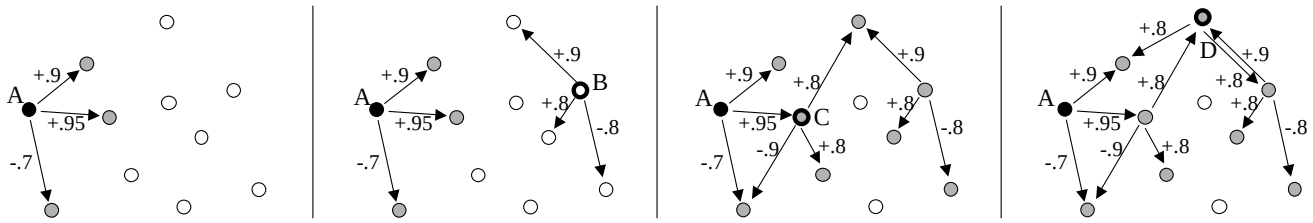

Figure 1: Illustration of client *A* building its correlation graph, obtaining data in turn from peers *B*, *C*, and *D*.

table, which is periodically updated by scanning the vote database. For each peer in the vote database, the client determines the set of objects for which it knows both the peer's vote and its own. These votes are then used to derive a peer correlation value, with weak correlations immediately discarded. Any strong correlations discovered are cached in the correlation table for use later, both for weighing votes during estimation and for selecting which votes to send in response to vote-gather queries.

2.5 Transitive Correlation

Computing correlations directly from the local vote database works well for peers having the most overlap in interests, but often fails to discover relationships between peers with little in common. Nor does it allow a client to leverage work done by other peers, but instead forces all members of a closely correlated group to discover pairwise correlations independently. Clients can avoid these problems by utilizing *transitive correlations*. This mechanism captures the notion that a strong positive correlation between *A* and *B*, and again between *B* and *C*, should be taken as an indication that all three peers tend to vote in a correlated manner.

To effect this computation, each client maintains a directed graph in which nodes represent network peers, and a directed edge (A, B) with weight r_{AB} represents the correlation between a client *A* and its peer *B*. Initially, a client populates its graph with the correlations it computes directly from its local vote database. The remainder of the graph is built by periodically selecting peers in the network and requesting their known correlations, as illustrated in Figure 1. This selection is random, but biased towards peers for which a strong positive correlation is already known, preferentially expanding the parts of the graph that will be of most use.

A client computes transitive correlations by multiplying weights along paths in the correlation graph. One way to view this computation is that votes from distant peers in the correlation graph are propagated back towards the client using weighted voting at each step of the process. As a simplification and optimization, instead of performing a computationally expensive graph flow computation, each client pre-computes only the max-path from itself to every other node in the graph, and stores any strong results found in its regular correlation table.

Trust is a concern whenever information is collected from an untrusted, anonymous peer. Here, the peer may lie about its correlations in an attempt to manipulate the client's correlation results. Our system employs two strategies to minimize the effect of such actions. First, the client can request a full or partial audit of the correlations gathered from each peer it contacts for graph edge updates. The audit consists of those votes that the peer originally used to compute the edge weights. This is made possible by having each peer share only correlations it has directly computed from its local vote database. Second, the correlation graph itself can be audited for inconsistencies, since it will typically contain significant redundancies. For instance, the two edges between a pair of peers should be approximately equal, and cliques of strongly correlated nodes should be internally consistent.

The remainder of this paper presents simulation results detailing the dynamic behavior of our system in a highly polluted network.

3 Evaluation

We evaluate our object reputation system in a custom discrete-event simulator written in Java. We model the system as a growing set of objects, and a fixed set of clients following a synthetic workload and implementing the voting, correlation, and transitive correlations protocols. We use a simple randomized topology with a fixed search width to model the underlying peer-to-peer network. In the next sections, we briefly describe the workload and client behaviors driving our simulations.

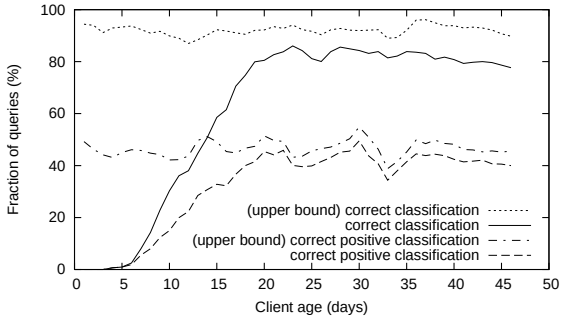

Figure 2: Probe client classification success rate.

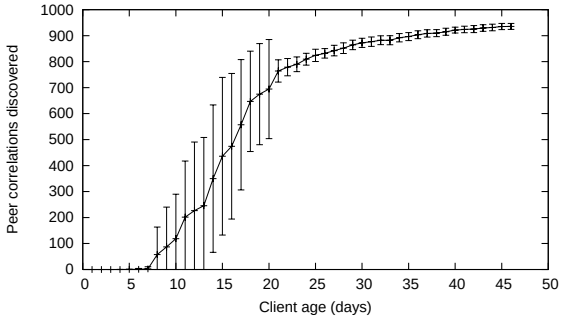

Figure 3: Average number of entries in probe client correlation tables, with standard deviations.

3.1 Clients, Objects, and Workload

We follow the abstract model of the Kazaa filesharing network described in [2]. The network consists of 1000 clients initiating 5 queries per day on average for 100 days. Each client draws queries from a population of 40000 objects, with new objects introduced at a rate of 5475 objects per year.

It is not possible to derive from [2] the behavior of individual clients. Intuitively we would not expect clients to behave uniformly, but rather that *clusters* of clients will share similar interest in objects. For example, recent studies find a high degree of clustering in the eDonkey filesharing network [1]. We model this effect by partitioning the objects into 20 *genres* and randomly assigning clients to 4 genres each, such that the popularity of genres, both in terms of object and assigned clients, follows a Zipf(1.0) distribution. Within a genre, object popularity follows a Zipf(1.0) distribution. A client makes a query by selecting first a genre, then drawing without repeats an object from the genre. Our measurements indicate that the overall query popularity distribution matches that reported in [2].

Two factors help model noise in the system. First, a client that has computed a reputation estimate v will accept and fetch the object with probability 1 if $v = 1.0$, with probability 0 if $v = -1.0$, and with a linearly varying probability otherwise. Second, clients generate votes on all objects that are accepted, but vote correctly with only 90% probability, and randomly the rest of the time.

We model a highly polluted network, where approximately half the objects are tagged as pollution, and the remainder tagged as authentic. The remaining simulation parameters are set with reasonable values to bound the bandwidth and storage costs associated with the reputation management system, and to ensure that correlations are robust to statistical variation.

3.2 Overall Success in Estimation

We evaluate the convergence and accuracy of our approach by tracking 20 probe clients inserted into the network starting on day 50. We first measure the total fraction of correct classifications across these probe clients. A correct classification is when a client computes a strong positive estimate $v > 0.5$ for an authentic object, or a strong negative estimate $v < -0.5$ for a polluted object.

Figure 2 shows how the success rate varies as the probe clients participate in the system. Also shown are upper bounds, the values that would have been attained had the clients shared complete global knowledge of all past votes on each object. The upper bound does not approach 100%, because a fraction of queries are for never-before-seen content, which cannot be correctly classified even using global information. By day 15, our scheme achieves a high rate of success, maintaining roughly 80% correct classification.

Of the queries not classified correctly, almost all are instances where no estimate is obtained at all, and only a few due to misclassification. These cases are split evenly between authentic and polluted objects, and less than 9% are for the 2000 most popular objects. This implies that our system gives very accurate estimates for the most popular objects, and rarely misclassifies even the least popular.

3.3 Correlation Table Behavior

To explain the factors driving the overall success rate, we turn our attention to the clients' correlation tables. If a client has discovered few or no strong correlations, then the votes it collects during vote-gather queries are of no immediate use, but rather only aid in building up a

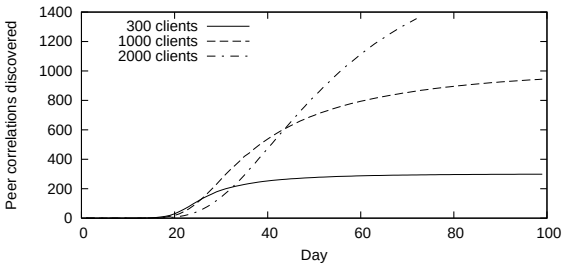

Figure 4: Average number of entries in initial client correlation tables.

database from which to later compute peer correlations. Conversely, a client with many entries in its correlation table will be able to use more of the votes it gathers, and so obtain more robust and accurate object reputation estimates. Figure 3 shows the average size of clients' correlation tables, which can be seen to closely match the overall success rate observed earlier in Figure 2.

After an initial period with an empty correlation table, the rate of discovery of peer correlations quickly rises, so that the probe nodes have discovered 500 strong correlations on average after 16 days. As will be illustrated in the next section, the 1000 initial clients' tables grow much more smoothly and slowly, reaching a size of 500 only at day 40. This highlights the key role played by transitive correlations in the quick convergence of the system. The initial clients do not have the advantage of existing peers from which to gather correlation data, whereas a probe client's table grows fast by taking advantage of the work done by earlier joining clients.

The initial lag time of approximately seven days, when the average correlation table size remains zero, is due to the lack of locally computable correlations. Recall that all edges leaving a client in the correlation graph must be computed by the client itself using only its local vote database. In effect, until a newly introduced client establishes some voting history of its own, it cannot directly or transitively compute correlations with any peers. This initial delay can easily be eliminated in practice with the use of a semi-trusted "sponsor" client from which the local correlation table can be initialized.

3.4 Scaling

In order to assess the scalability of our protocol as the size of the network varies, we ran the simulations with proportionally larger and smaller networks. Figure 4 shows the average correlation table size of the initial clients for each network. The total convergence time appears to grow approximately linearly in the size of the network when measured in terms of the time needed to derive correlations for half of the clients. However, note that the overall success rate grows quickly as soon as clients discover a few strong correlations, and that in practice a limited number of entries is likely to be sufficient to compute robust estimates, due to the high degree of clustering of client interests expected in practice.

3.5 Dynamic Behavior

Our system contains subtle feedback loops that give rise to resilience against attack, and help explain the system's overall dynamics. For instance, an authentic object tends to follow an exponential increase in reputation, since correct positive votes induce a positive feedback cycle: Each such vote increases the object's reputation among correlated clients. This in turn leads to more likely acceptance and retrieval, and thus to additional positive votes. An attacker voting negatively on this same object, however, induces a damped response, since a lowering of perceived value will not result in more negative results, but instead will simply lower slightly the rate of acceptance.

For an object on which honest clients tend to vote negatively, but an attacker votes positively, the system will react strongly against the attacker. Each positive vote in this case raises the expectations of the honest clients, leading eventually to an increase in correct negative votes which counteracts the initial, incorrect vote. Additionally, the attacker will see a decrease in its correlation values with honest clients. The net effect is a disincentive for an attacker to be dishonest too often.

3.6 Attacks

We have investigated the impact of several simple attacks on our system, both analytically and through simulation. Due to the dynamic effects discussed above, we can immediately see that several attacks that are quite effective in existing systems have little effect and, in many cases, actually provide a tangible benefit to the system. For example, a peer that consistently lies about the authenticity of objects is just as useful as a peer that consistently generates honest votes, since in the former case the votes will simply be multiplied by a negative weight. Voting randomly, however, will lead to the votes being essentially discarded as peer correlation values will converge to zero in this case. A rational attacker, then, has an incentive to

vote honestly in order to keep from approaching either of these extremes, and so must carefully balance the amount of information leaked to the network.

Similarly, an attacker that deviates from the normal protocol by gathering and disseminating only incorrect votes is actually beneficial to honest clients. Indeed, it typically allows an honest client to more quickly identify those peers with which it is less correlated. Other similar attacks can slow down the rate of information diffusion, but do not alter the overall trends seen above.

4 Related Work

Our system addresses a fundamental weakness in open peer-to-peer networks that is only recently gaining attention. An overview of the uses and social factors involved in reputation systems can be found in [8]. Closely related to our approach are recommender systems, which filter content based on user ratings. GroupLens [7] and related systems, for instance, provide a centralized content rating system similar to ours, but implemented for Usenet articles. Our algorithms also resemble a distributed version of those discussed in [10] for use in electronic marketplaces.

In P2P networks, Eigentrust [4] computes global reputation values for peers to address the *free-loading* problem. Our work differs in that our simple object rating system allows the reputation system to function without need for direct peer ratings or pre-trusted peers. In addition, we believe that global consensus on object reputation will be increasingly difficult to achieve as networks become more heterogeneous.

The peer correlations we compute partially rely on the level of semantic clustering, which is a measure of the locality of interest between peers. Semantic clustering has also been used to improve P2P search performance and network structure [3, 9].

Evidence for pollution in peer-to-peer networks varies depending on the structure of the network and its trust model. Ross [5] finds that pollution is rampant in the decentralized Kazaa network, which has no trusted participants or central authority. Conversely, Pouwelse [6] reports that BitTorrent maintains a nearly pollution-free system by using human intervention and centralized admissions control.

5 Conclusion and Future Work

The object reputation system presented here is robust to attack and converges quickly, on the order of a few tens of queries for clients joining the system. However, we are exploring several approaches that could further improve the effectiveness and decrease the costs of our system. We are experimenting with a randomized algorithm for vote-set compression, which allows clients to store and transmit more votes, resulting in more responsive and more statistically robust peer correlation estimates. At the same time, we are exploring the use of recent cryptographic techniques that may allow clients to more efficiently verify large numbers of signatures.

References

[1] Fabrice Le Fessant, Sidath Handurukande, Anne-Marie Kermarrec, and Laurent Massoulié. Clustering in peer-to-peer file sharing workloads. In *International Workshop on Peer-to-Peer Systems*, February 2004.

[2] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *ACM Symposium on Operating Systems Principles*, October 2003.

[3] Sidath Handurukande, Anne-Marie Kermarrec, Fabrice Le Fessant, and Laurent Massoulié. Exploiting semantic clustering in the eDonkey P2P network. In *ACM SIGOPS European Workshop*, September 2004.

[4] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The EigenTrust algorithm for reputation management in P2P networks. In *International World Wide Web Conference*, May 2003.

[5] Jian Liang, Rakesh Kumar, Yongjian Xi, and Keith W. Ross. Pollution in P2P file sharing systems. In *IEEE INFOCOM*, 2005.

[6] J.A. Pouwelse, P. Garbacki, D.H.J. Epema, and H.J. Sips. A measurement study of the BitTorrent peer-to-peer file-sharing system. Technical report, Delft University of Technology, April 2004. PDS-2004-003.

[7] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. GroupLens: an open architecture for collaborative filtering of netnews. In *ACM conference on Computer supported cooperative work*, 1994.

[8] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Communications of the ACM*, 43(12), December 2000.

[9] Kunwadee Sripanidkulchai, Bruce Maggs, and Hui Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *IEEE INFOCOM*, 2003.

[10] Giorgos Zacharia, Alexandros Moukas, and Pattie Maes. Collaborative reputation mechanisms in electronic marketplaces. In *Hawaii International Conference on System Sciences*, January 1999.