Experience with an Object Reputation System for Peer-to-Peer Filesharing

Kevin Walsh Emin Gün Sirer
Cornell University
{*kwalsh,egs*}*@cs.cornell.edu*

Abstract

In this paper, we describe Credence, a decentralized object reputation and ranking system for large-scale peer-to-peer filesharing networks. Credence counteracts pollution in these networks by allowing honest peers to assess the authenticity of online content through secure tabulation and management of endorsements from other peers. Our system enables peers to learn relationships even in the absence of direct observations or interactions through a novel, flow-based trust computation to discover trustworthy peers. We have deployed Credence as an overlay on top of the Gnutella filesharing network, with more than 10,000 downloads of our client software to date. We describe the system design, our experience with its deployment, and results from a long-term study of the trust network built by users. Data from the live deployment shows that Credence's flow-based trust computation enables users to avoid undesirable content. Honest Credence clients can identify three quarters of the decoys encountered when querying the Gnutella network.

1 Introduction

Establishing trust is a fundamental problem in distributed systems. Peer-to-peer systems, in which service functionality is distributed across clients, eliminate the centralized components that have traditionally functioned as de facto trust brokers, and consequently exacerbate trust-related problems. When peers lack meaningful measures on which to base trust decisions, they end up receiving services from untrustworthy peers, with effects that can range from wasted resources on mislabeled content to security compromises due to trojans. These problems are particularly evident in current peer-to-peer filesharing networks, which are rife with corrupt and mislabeled content [3]. Such content can waste network and client resources, lead users to download content they do not want, and aid the spread of viruses and other malware. Recent research confirms the vulnerability of deployed filesharing networks to corrupt and mislabeled content, and indicates that much of this pollution can be attributed to deliberate attacks [14].

The underlying problem facing clients of peer-to-peer filesharing systems is that they must assess the trustworthiness and intent of peers about which little is known. A simple approach is to use past experience with a peer to determine that peer's trustworthiness. But when client interactions are brief and span a large set of peers that changes dynamically, the opportunity to reuse information from past first-hand experience is limited. Another conventional approach is to interpret a shared file as an endorsement for that file's contents, similar to the way hyperlinks are interpreted as implicit votes by search engine ranking algorithms. Our data indicates that sharing is not a reliable indicator of a user's judgment. Peer-to-peer filesharing networks call for trustworthiness metrics that are robust, predictive, and invariant under changing network conditions and peer resource constraints.

In this paper, we describe Credence, a new distributed reputation mechanism for peer-to-peer filesharing networks that enables honest, participating peers to confidently determine *object authenticity*, the degree to which an object's data matches its advertised description. Credence allows clients to explicitly label files as authentic or polluted, and to compute reputation scores for peers based on a statistical measure of the reliability of the peer's past voting habits. Combined, these two techniques provide relevant and reliable data so that clients can make informed judgments of authenticity before downloading unknown content. Credence's mechanism for computing peer reputations is fully decentralized, is not affected by extraneous or transient properties of peers, and is robust even when peers collude to misrepresent the authenticity of files in the network.

In order to gauge the trustworthiness of a peer in the absence of direct interactions or observations, Credence incorporates a flow-based trust computation. Conceptually similar to PageRank-style algorithms for propagating reputation through links on the Web [18], Credence's algorithm differs in fundamental ways from previous approaches. Credence is completely decentralized, and does not assume consensus on a set of pre-chosen peers from which trust flows in the network. Instead, each Credence client propagates trust from itself outwards into the network, using local observations to compute reputations in its immediate neighborhood, and input gathered from its community to judge more distant peers.

We have built and deployed a fully functioning Credence client as an extension to the LimeWire [15] client for the Gnutella filesharing network. We have been actively probing and monitoring the status of the Credence network since its public release. In this paper, we present a long-term study of the emerging properties of the Credence network using data collected over a period of nine months. This data validates the underlying assumptions used in the design of Credence, confirms that Credence's trust mechanism allows users to discover and maintain useful trust relationships in the network, and provides new insights into the behavior of filesharing users.

This paper describes the goals and assumptions of Credence, details the design and implementation of our Gnutella-based overlay, and presents the results of our long-term study of the evolution and structure of the Credence network. We show that the collective actions of Credence users have produced a robust network of peer relationships that lets honest peers avoid a majority of the pollution they encounter during typical queries. More fundamentally, this work demonstrates that it is feasible to construct an effective, fully decentralized reputation system in a large-scale peer-to-peer network.

2 Approach

Participants in a peer-to-peer filesharing network cooperate by routing queries from a client interested in an object to a set of peers serving it. Each object consists of some opaque content and, to facilitate searching, a formatted object descriptor containing the object name, encoding, content hash, and other descriptive meta-data. Clients issue keyword queries to their peers in the network, receive matching object descriptors in response, and then may download selected objects from among the responses.

Pollution is a problem when clients cannot reliably distinguish between descriptors for authentic objects, malicious decoys, mislabeled content, and accidentally damaged files. In an effort to identify authentic descriptors, typical clients rank search results according to the advertised file quality or the relative popularity of the file among the search results received. Such measures are easily manipulated by simple adversaries, rendering them unreliable at best and deceptive at worst. Recent work has shown that listing search results randomly is substantially more reliable than these rankings [9].

In this paper, we focus on the problem of distinguishing authentic objects from polluted ones. Other types of attacks can, of course, disrupt networks, and clients must use additional techniques to guard against them. Since we abstract away the underlying peer-to-peer network, our work is applicable to any file sharing network in which a decentralized object reputation system is called for and no pre-existing trust relationships can be assumed.

2.1 Objectives

The design of Credence is guided by several goals that are necessary requirements for a successful peer-to-peer reputation mechanism:

- *Relevance:* The system must use only pertinent information when evaluating the authenticity of objects and the credibility of peers.
- *Distribution and Decentralization:* No participants should be trusted a priori, and no central computation should be required during online operation.
- *Robustness:* The system must be robust to attacks by large numbers of coordinated, malicious peers.
- *Isolation:* The decision to participate in the reputation system should be independent of decisions to participate in unrelated activities, such as sharing files, contributing bandwidth, or remaining online.
- *Motivation:* Users must have realistic incentives to participate honestly in the reputation system.

To meet these objectives, the basic operation of a Credence client is organized around three main activities. First, Credence users vote on objects based on their own judgment of the object's authenticity. Second, Credence clients collect votes to evaluate the authenticity of objects they are querying. And third, clients evaluate votes from their peers to determine the credibility of each peer from the client's own perspective. In the base case, peer weights are computed by examining correlations in the voting histories of a client and its peers. In the general case, pairwise weights are combined to yield a trustworthiness metric using a graph flow algorithm.

In Credence, clients express their judgments about the authenticity of files using explicit voting, and the reputation system avoids any reliance on implicit indicators of a client's judgment. This is in sharp contrast with past systems that rely on sharing as an implicit endorsement of a file. As we show later, honest users often share corrupt or malicious files, files that they themselves were fooled into downloading. Similarly, the decision not to share a file is typically independent of the file's authenticity.

A client collects votes in order to evaluate the authenticity of prospective downloads. Credence uses a decentralized algorithm for propagating votes from the peers that cast them to the clients that seek them. Our implementation uses the underlying filesharing primitives to perform vote routing and search services, and so does not require any centralized coordination or computation.

Since deployed networks contain many unreliable and malicious peers, a client needs robust methods for evaluating the credibility of its peers. By definition, a peer's credibility with respect to judging file authenticity depends only on the votes it casts. In the absence of global consensus on the correctness of past votes, each client must decide from its own perspective if a peer's past votes were useful. Specifically, a client can compute the degree to which each peer's judgments match its own past votes, and then preferentially rely on votes from like-minded peers. In cases where direct pairwise evaluation is impossible due to insufficient overlap in voting activity, Credence employs a novel flow-based computation which extends trust relationships transitively through known peers to more distant peers.

The dependence on a client's own voting record provides a natural incentive to participate by voting often and carefully, since the client otherwise will find itself relying on similarly careless peers. In contrast to previous systems that rely on characteristics of past interactions, such as network bandwidth, peers in Credence are judged only by the votes they have cast. This isolates the reputation system from other decisions a filesharing peer must make, such as which peers to interact with, or how to allocate resources. Negative votes play an important role in identifying honest peers since, even if user interests differ or there is little overlap in the authentic files seen by different users, we expect near universal agreement on negative votes for spam and decoys among honest users. We show later that this is borne out in practice – successful spam attacks on Gnutella help shape the Credence network, and help honest peers identify each other.

2.2 Vote Semantics

Credence works most effectively when a large fraction of users essentially share a common evaluation function, and so tend to agree when voting on objects in common. Widely accepted semantics for positive and negative votes will increase the chances of locating like-minded peers to rely on when evaluating object authenticity. We chose to base our system on file authenticity, rather than more subjective issues of taste or quality, for precisely this reason. This decision is in contrast to recommender systems, which try to identify a small number of peers with similar taste from which to make recommendations about new content. Recommender systems face significant challenges when clients have widely divergent tastes, and the need for a peer-to-peer approach is unclear given the existence of successful centralized recommender systems. Credence thus focuses solely on determining whether a file matches its description.

2.3 Cryptographic Keys

Credence ensures the integrity of votes by equipping every client with a cryptographic key pair K that is used to sign votes. Signatures prevent attackers from modifying existing votes or manufacturing new ones on behalf of other peers. Credence limits Sybil attacks [8] by requiring each client to possess a certificate cert_K signed by a central authority that vouches for K's validity. Our initial implementation rate limited the generation of certificates by requiring clients to download a large file for each requested certificate. The current implementation requires each client to solve a cryptographic puzzle, similar to the scheme proposed in [2]. The certificate authority in the download-for-key approach is entirely offline, while in the puzzle-for-key approach, it is online but contacted only once during initial installation. In either case, the certificate authority plays no role in the filesharing network itself, and could be distributed using well known distributed authentication schemes if desired [29]. In all cases, client keys are not bound to real-world identities, but instead use randomly generated key pairs without any identifying information. These keys provide anonymity comparable to the anonymous pseudonyms found in existing filesharing networks.

We are now in a position to specify the actual protocol that Credence clients implement. In Section 3, we will evaluate how the protocol behaves in a real network, and examine evidence supporting the above assumptions using our long-term study of the deployed system.

2.4 Voting on Objects

The underlying goal of Credence is to allow a user to judge the authenticity of search results, each consisting of a file content hash and meta data, including the file's name, size, and type. Each search result can be viewed as a claim about the file's attributes. For example, $\langle H\colon \mathit{gettysburg} \subseteq \text{name}, \mathit{mp}3{=}\text{type}, 128{=}\text{bitrate}\rangle$ makes the claim that the file with content hash H has the specified attributes, where the symbol $\subseteq$ is used to indicate that *gettysburg* is one of possibly many valid names for the file.

Credence clients express their observations by issuing votes, with each vote naming a file content hash and making specific claims about the file's attributes or contents. Other peers use these votes to evaluate the claims made by search results, by comparing if the claims specified the vote to those found in the search result. We say that a vote *applies* to a search result, either negatively or positively, if it either refutes or supports the search result's claims. For instance, a vote specifying $\langle H\colon \mathit{mp}3 \notin \text{type}\rangle$ applies negatively to the example search result above, since the two are mutually incompatible. Note that a vote's application may differ somewhat from the voter's original *intention*, as for example a vote $\langle H\colon \mathit{jpg} \notin \text{type}\rangle$ most likely intended to say something negative but would not apply at all to the above search result.

Clients must agree on a common syntax and semantics for making statements about objects. The language must be simple enough that ordinary users can encode their observations as votes, expressive enough to account for different types of pollution, and the semantics must be faithful to the user's intentions when voting. The voting language described here, and implemented in the current version of Credence, was carefully chosen to balance these three often conflicting goals.

Formally, each vote is a signed tuple $\langle H\colon S, T\rangle_K$ containing a file content hash H, a statement S about the file, and a timestamp T, together with the client's key certificate cert$_K$. A statement is an attribute value pair, combined with set operators $\subseteq$, $=$, $\notin$ and $\supseteq$. The attribute and value are arbitrary strings, though Credence clients currently recognize three globally defined attributes: name, type, and bitrate. For efficiency, we allow multiple statements to be concatenated using an implied logical conjunction and signed together as a single vote.

The statements in votes enable users to designate particular values as valid or invalid for a given attribute, as follows. The statement $v \subseteq a$ says that v is a valid value for attribute a, though a may take on other values as well. Some attributes, most notably a file's name, are naturally multi-valued in the sense that many different values may be appropriate and valid for given unique file content hash. This operator can be used to specify one of the possibly many names. In contrast, $v = a$ says that v is a valid value for attribute a and all other values are invalid. Attributes, such as the fixed bitrate of an audio file, can take on only a single possible value, and this operator enables users to express that single value. A negative statement $v \notin a$ says that v is an invalid value for attribute a. This operator is used to refute specific file advertisements, such as a single misleading file name or type. Finally, $v \supseteq a$ says that a may not take on values other than v. This operator allows clients to make strong, broadly applicable negative statements, without committing positively to the specified value. Such statements are particularly useful for thwarting relabeling attacks, where malicious peers change a file's metadata such that existing negative votes, when evaluated in a new context, might inadvertently apply positively to the modified metadata. Overall, these four operators enable Credence voters to express a wide range of statements through an easy-to-use graphical interface.

The Credence user interface gives users the option of voting in various ways on files that are shared locally, were recently downloaded, or are about to be deleted. The user can vote *thumbs-up*, which generates statements for the file's name, type, and bitrate, using $\subseteq$ and $=$ where appropriate. The user can vote *thumbs-down* to indicate that the file metadata was misleading, and select one or more attributes to include in the vote. Negative statements are generated using $\notin$ for the file's name and type. Since the true bitrate, by contrast, can be computed directly from the file, the vote can include a much broader negative statement using the $\supseteq$ operator for the bitrate. Credence also supports an unconditional *thumbs-down* vote that generates the statement $\langle H\text{: name} = \emptyset \rangle$, which appropriately refutes any search result, since all valid such results contain a non-null name. This unconditional vote is meant to be used when the file contains a virus or otherwise wholly inappropriate content, under the assumption that no Credence user wishes to download a virus even if it is correctly labeled as such.

The user interface and vote operators were carefully designed with the goal of faithfulness in mind. In particular, thumbs-down votes do not ever support the claims of a search result, even when the known correct bitrate is included. Thus, a user that downloads a file with an incorrect name and incorrect bitrate does not inadvertently vote up the same file with a different incorrect name but the correct bitrate. In contrast, positive votes can apply negatively if the file's attributes are altered to be incompatible, such as would happen if the file type were modified after a thumbs-up vote was generated.

Early versions of the Credence software allowed only for an unconditional thumbs-down vote, as above, or an unconditional thumbs-up vote. This thumbs-up vote is handled as a special case and applies positively to any search result with the specified file hash. Since much of the data collected in our traces comes from our initially deployed clients, the analysis in Section 3 considers only the binary, up/down voting logic. Non-legacy votes are simply translated to unconditional votes as necessary.

2.5 Collecting and Storing Votes

In Credence, a client evaluating an object's authenticity actively queries the network to find, collect, then aggregate a sample of relevant votes. We implemented vote collection using the existing query infrastructure by issuing a *vote-gather* query, specifying the hash of the file of interest, to the underlying Gnutella network. This reactive, pull-based dissemination of votes is motivated by the Zipf popularity distribution of objects, since any given vote is unlikely to be of interest to many users.

The query is routed by Gnutella to peers sharing votes for the object, who respond by sending their own matching votes and any matching votes they have seen recently. If a peer knows many votes for the given hash, it sends only those with the most weight (from its own perspective), both in order to bound the overall cost of vote collection, and to ensure that the most useful votes are disseminated further in the network. Sending these additional votes improves vote availability and overall dissemination, and incurs little marginal cost. Specifically, voters are not required to remain online, since their votes can still be propagated by other peers.

In order to be able to respond to vote-gather queries as they arrive from the network, each peer maintains a *vote database* from which matching votes can be drawn. For each file content hash, the database stores a row with a timestamp, the peer's own vote, if any, and a list of other votes encountered recently for the object. Note that votes are maintained in the database regardless of how the peer voted on the file, or if the other votes agree with its own. As older entries expire, the database is constantly replenished from the peer's own voting activity and from votes the peer receives after issuing its own vote-gather queries. The peer can further augment its database using a straightforward gossip exchange with its peers. The resulting database size is proportional to a peer's gossip rate and frequency of voting and estimation, and independent of the number of files in the network.

2.6 Weighing Votes

After collecting a set of votes for an object, the client verifies the signature and key certificate on each of the votes, then aggregates the set into a single reputation estimate to present to the user. Simply tabulating the available votes using unweighted averaging would be prone to manipulation, as attackers could simply flood the network with votes. Instead, each Credence client computes a trust metric for each vote, and uses weighted averaging to compute an estimate of the object's overall reputation. The resulting score is interpreted as a personalized estimate of the authenticity of the object, and can be used to make a more informed decision to accept (and fetch) or reject the object. In cases where no votes could be found, the user must resort to ad hoc estimates of authenticity, as used in past systems. Such cases are unavoidable during the initial deployment of any reputation system that does not rely on prior trust relationships.

The first step in aggregating votes is to evaluate how the claims in each vote apply to the relevant search result. Votes that apply positively are given an initial value of $+1$, and those that apply negatively -1. From the perspective of a client evaluating a set of votes, however, the usefulness of a particular vote depends on the relationship between the client and the peer that cast the vote, and so each client weighs the initial vote values according to the strength and bias of this relationship. Intuitively, peers that tend to vote identically (or inversely) on objects should develop strong positive (or negative) weights for each other's votes over time, while a client should disregard votes from peers that, from its perspective, appear to vote randomly.

2.7 Computing Correlations

Statistical correlation precisely captures this notion by comparing the shared voting history of each pair of peers. A Credence client determines, for each of its peers, a correlation coefficient θ to use as a weight during vote aggregation, based on the files voted on in common between the client and the peer. Conceptually, θ is calculated by examining the instances when both peers make statements about some file, and taking into account whether the statements have a positive or negative intention.

Normally, a single vote applies either positively, negatively or not at all, depending on the search result in question. The correlation computation takes place without reference to any particular file or search result, however, and so uses the original intent of each vote to directly compare the voting history of the client and peer. Specifically, we say that a pair of votes *conflict* if there is a search result to which the votes apply oppositely. We say the votes *agree* if they do not conflict and there is a search result that both support or both refute.

The coefficient θ is computed by examining all pairs of votes between two peers A and B that either conflict or agree with each other. Let a (respectively b) be the fraction of such votes from peer A (respectively B) with positive intention, and let p be the fraction of such pairs that agree with both votes having positive intention. Then $\theta = (p - ab) / \sqrt{a(1-a)b(1-b)}$ is the *coefficient of correlation*, taking on values in the range $[-1, 1]$. This computation represents a standard technique for computing correlations on binary data. Positive values indicate agreement between peers, negative values indicate disagreement, and small $|\theta|$ indicates the absence of any significant relationship between the two voting histories.

Client A normally uses weight $r_{AB} = \theta$ for the votes cast by a peer B. When peers lack sufficient voting history to establish a robust estimate of θ, or when the correlation value itself is statistically insignificant, the client sets $r_{AB} = 0$ and so disregards votes from the peer. For peers whose votes are all negative or all positive, θ is usually undefined even if the peers are mostly or completely in agreement. Such cases may be common for clients newly joining the network, and so Credence uses a heuristic to allow such clients to quickly begin establishing tentative relationships. When θ is undefined, the software resorts to a simple vote agreement counting metric in the range $0 \leq |r_{AB}| \leq 0.75$. This range was chosen to be below the majority of existing correlation values, and reflects the decreased confidence in such heuristics as compared to the correlation computation. Clients may disable this heuristic, and it may be removed altogether if users can be convinced of the importance of voting both positively and negatively early on.

A client can only compute accurate and strong peer correlations if it has itself cast a sufficient number of both positive and negative votes. This restriction provides a strong incentive for users to participate in voting, since users that do not vote will find the quality of the estimates they compute noticeably degraded. A user can still benefit from Credence by voting honestly but privately, suppressing the sharing and dissemination of their votes to other users.

Credence clients use the information stored in their vote databases to periodically compute correlations for known peers. 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, derives from this set a peer correlation value, and caches any strong correlations found in a *correlation table*. The correlation table is consulted when weighing votes during the evaluation of an object's authenticity, and for selecting votes to send in response to vote-gather queries from other clients.

2.8 Flow-based Peer Reputation

Computing correlations directly from the local vote database works well for peers that vote on overlapping sets of objects, and are thus well represented in the local vote database. But pairwise correlations cannot robustly evaluate the relationship between a client and peers having only a few interests in common with the client. We overcome this limitation by allowing clients to leverage the correlations discovered by their peers, effectively expanding their horizon along paths of correlated peers. Credence incorporates a notion of *transitive correlation* which enables strong correlations between a client and a nearby peer, and again between this peer and a more distant peer, to be combined into an estimate of the relationship between the client and the distant peer.

Transitive correlations are computed by building and maintaining a local model of the pairwise trust relationships between peers in the network, then periodically executing a flow-based algorithm on the resulting trust graph. Nodes in the trust graph represent peers in the network, and a weighted edge between nodes represents one peer's correlation estimate for another. Initially, a client populates the trust graph using locally computed correlations from its local vote database. The remainder of the graph is built using a gossip protocol, where each client randomly selects peers in the network and exchanges locally computed correlation coefficients. The selection of these gossip partners is biased towards peers with known positive correlations to preferentially expand the most useful parts of the graph.

Intuitively, votes from peers distantly connected in the graph can used to approximate the votes of peers more closely connected, by emulating the weighted voting computation at each step along the path. Performing a potentially large graph computation in this way during every search result evaluation is likely too expensive. We can approximate this computation by multiplying the weights along paths with strong weights in the graph, and so precompute an approximate effective weight to be used to weight the votes from each distantly connected peer. As a simplification and optimization in our implementation, each client periodically computes only a single maximum weight path to every other peer in its local graph, where path weight is the product of weights along edges. This computation is constrained to use paths where negative weights appear only on the last edge in the path, since a client cannot trust a negatively correlated peer to provide useful judgments about correlations to more distant peers. The resulting transitive correlations are cached for later use in weighting votes when a local correlation is not available.

Credence proposes two strategies to protect the reliability of its local trust graph against peers that lie about correlations when exchanging information. First, because only locally computed correlations are exchanged, a client can choose to audit the computation by requesting some or all of the inputs from its peer. Recall that inputs to the correlation computation are votes from peers, signed to maintain integrity. Second, in practice, the trust graph contains significant amounts of redundant information in the form of cycles and densely connected cliques. Auditing the graph itself can help the client identify misbehavior in the form of inconsistent information, and can also help guide decisions of which peers to audit directly. Auditing is not currently implemented in the deployed version of the Credence software.

The remainder of this paper presents our analysis of data gathered from the deployed Credence network.

3 Evaluation

Credence is the first peer-to-peer reputation system to be deployed widely on a live network, with over 10,000 downloads of our software since its initial public release in March, 2005. In this section we present an analysis of data collected in a long-term study of the deployed Credence network. The data presented here gives a unique view into the individual and collective behavior of file-sharing peers, and demonstrates the feasibility of a fully distributed reputation system in real settings.

We collected data on Credence clients using a continuously running crawler and have compiled more than 200 daily snapshots of the structure of the network over a span of nine months. Each snapshot contains the cumulative set of votes discovered by the crawler. Clients are identified in the data set only by their randomly chosen public key. Since a vote identifies the file to which it applies only by the file's hash, our vote data set does not contain the corresponding names of the files being voted on. However, over a span of six months a second crawler collected the names and hashes of files publicly shared by Credence clients, enabling us to contrast the sharing and voting habits of users. In order to compile a consistent view of the network for analysis, both crawlers ran simultaneously, and all analysis was performed off-line after data collection was finished. Cumulatively, the data contains over 39,000 votes cast and 84,000 files publicly shared by over 1200 Credence clients.

Our dataset likely comprises only a portion of the Credence user population, since peers that join and leave the network rapidly may be missed by our crawler. In general, Credence clients do not make complete information about their trust computations available, such as the list of known transitive relationships. In the analysis below, we simulate the perspective of clients in the network by recomputing the pairwise and transitive correlations each

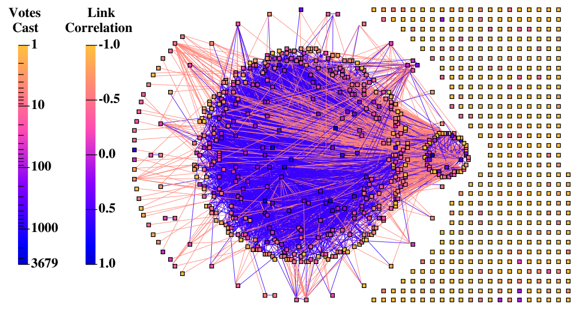

Figure 1: A global view of the Credence reputation graph with nodes and links shaded to reflect the number of votes cast by a peer and the correlations between peers. The large set of nodes in the center tend to correlate positively with each other, while clusters around the periphery are internally coordinated but conflict with the rest of the network.

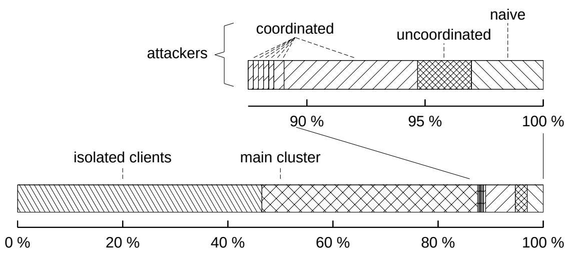

Figure 2: Classification of Credence users. Isolated client have no correlations to any other clients. Clients outside the main cluster are further classified according to type: *naïve attacker* for those that achieve only negative correlations; *uncoordinated attacker* for those that achieve some positive correlations but appear to be acting independently; and *coordinated attacker* for those that appear to have coordinated voting patterns different from those in the main cluster.

peer normally computes, under the assumption that our vote set is representative of the complete set of votes in the network. This has the additional advantage of allowing us to evaluate different parameters and thresholds for the pairwise and transitive correlation computations.

In the next section we present a high level view of the Credence reputation graph as it exists at the end of our data collection, and in subsequent sections we evaluate the effectiveness of the Credence approach, examine assumptions behind the Credence design, and discuss the underlying factors driving the evolution of the network.

3.1 Graph Structure

Central to the Credence protocol is the ability to discover relationships between peers, so we begin with a global view of the trust relationship graph, derived from the cumulative set of votes collected by our crawler. Figure 1 presents the correlation values between any pair of peers with overlapping vote histories, formatted to reflect the clustering due to positive and negative correlations.

The most striking feature of the network is that, aside from the completely isolated nodes at the right, the graph is completely connected and has a very dense link structure. On average, each connected node is directly correlated with 27 other peers in the network. When combined with Credence's flow-based algorithm, this enables peers to derive reputations for a significant portion of the entire network. The isolated clients have no correlations, a result mainly of their very low voting activity. At the end of our study, isolated clients had cast on average fewer than 5 votes, compared with 82 votes on average for the connected nodes. The isolated clients are typically new clients, and make their way into the main cluster as they produce a more substantial voting record.

Among the active, connected clients, several vote completely oppositely as their peers, resulting in a negative correlation value for every incident edge. These peers are placed at the left of the figure. The remaining peers have a mix of positive and negative correlations with their peers. The large central component contains nodes with mainly positive correlations among each other. The smaller clusters of nodes that can be seen around the periphery of the central cluster have largely positive correlations internally, but mainly negative correlations with the rest of the network. Note that we have presented a global view of the graph structure, and do not show how any particular client would view the network, since each client uses its own votes to independently decide which nodes it considers inside its own cluster (positively correlated) and outside its cluster (negatively correlated). We show in the next sections that users in the main cluster can make such classifications accurately.

Coordination and Disagreements

Further examination of the reputation graph produced by Credence voters reveals the overall level of coordination and disagreement in the reputation system. If many users share a common notion of authenticity and pollution, then clients will more easily find correlated peers in the network. Figure 2 provides a classification of Credence users based on clustering observed in the global reputation graph. Some users vote so rarely or on such obscure files that they cannot derive any correlations, and are classified as *isolated*. A large majority of the remaining users are members of the single central cluster, and tend to agree on the authenticity of most files.

Approximately 3% of users are classified as *naïve attackers*, since they are easily identified as voting in direct contradiction to all other connected nodes. We also find an additional 10% of nodes that vote more often in contradiction to the overall network than they vote in agreement. More than half of these belong to a single large cluster of *coordinated attackers*, which can be seen to the right of the central cluster in Figure 1. The remaining nodes either participate in smaller coordinated attacks, or act as *independent attackers*.

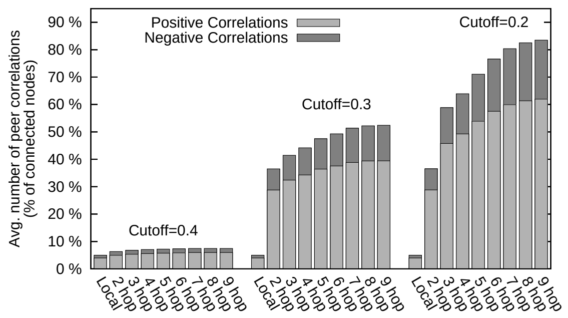

Figure 3: Number of local and transitive correlations computable under varying correlation strength threshold.

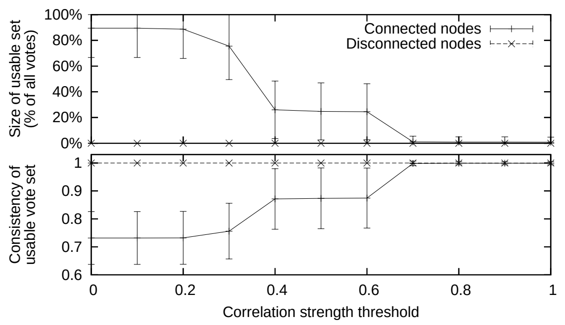

Figure 4: Characterization of votes usable by a client using various correlation strength thresholds.

The Credence network has no authoritative source of content or a priori trusted peers. In particular, although we have labeled nodes outside the main cluster as attackers, it is not possible from our data to ascribe malicious intent to these nodes. They may simply have a different concept of file authenticity and pollution than the majority of nodes, or may not understand the voting process. From the perspective of the nodes in the central cluster, however, peer intent is irrelevant and all outside nodes can be fairly labeled as attackers. In Credence, however, such attackers are not necessarily damaging to individuals or the network as a whole. The votes from naïve attackers, for instance, are simply inverted by all other clients in the system, and so actually provide a tangible benefit to the system. Overall, the high level of agreement indicates that file authenticity is a fairly universal concept among filesharing users, justifying Credence's reliance on voting and correlation as a method of identifying authentic files and credible peers.

Local and Transitive Relationships

The set of peer correlations computed by a client plays a significant role in Credence, since it defines the set of votes that are normally used by the client when evaluating objects in the network. In this section, we show that clients participating in Credence's reputation system are able to identify both positively and negatively correlated peers in the network, and so can take advantage of a large fraction of the votes in the network.

Credence clients use direct, pairwise correlations to peers when possible, and use transitive correlations to propagate trust through these correlations to more distant peers in the network. Figure 3 shows the impact of both pairwise and transitive correlation computations on a client's view of the network, under varying strength criteria. We can see that the number of correlations computable directly from local information is fairly small on average, but that, depending on the choice of threshold value, a much larger set of correlations can be computed by clients using transitive information. The greater number of positive correlations found is due mainly to the overall trend toward agreement and cooperation observed in the network, and in part to a bias against negative values in the transitive correlation computation.

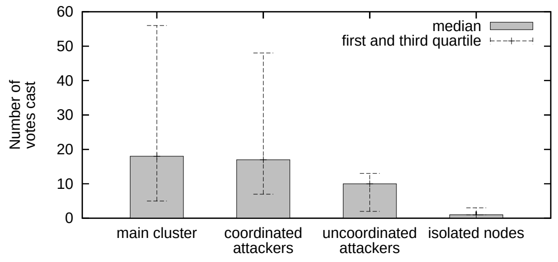

Figure 5: Number of votes cast according to client type. The low voting rate of isolated clients can be attributed to users that are either inactive, or have just recently joined the network.

Setting the correlation threshold allows a client to make an important tradeoff: a larger number of peer correlations could allow a client to take advantage of more votes from the network, but can also decrease the quality of estimates by including votes from weakly correlated and frequently inconsistent peers. Figure 4 illustrates this tradeoff by comparing the size of the set of usable votes for a given correlation threshold, and the consistency of this set of votes. To measure consistency, we compute the number of pairs of votes in agreement divided by the number of pairs in agreement or conflict. As a client increases its correlation threshold, it will have fewer peer correlations available, prompting both a decrease in the number of votes that can be used, and an increase in the consistency of the usable vote set.

Assimilation of New Clients

Clients newly joining the Credence network must be able to quickly discover peers with which they are correlated, so that the accuracy of their authenticity calculations can quickly increase. Our previous work, based on simulations, showed that transitive correlations play an important role in allowing new clients to quickly join the network [24]. After an initial period to establish a local voting record and a few pairwise correlations, the flow-based computation can immediately take advantage of the correlation results computed by already established

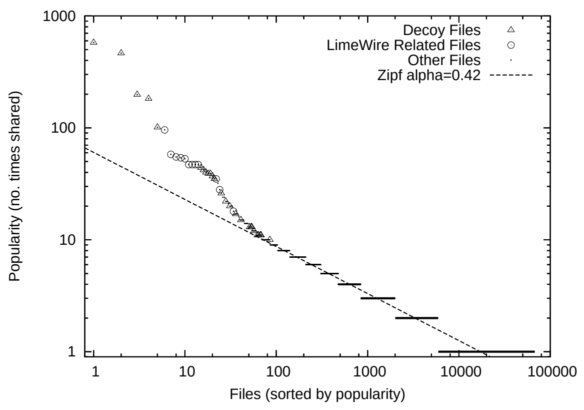

Figure 6: File popularity by number of times shared is approximately Zipf in the Credence network, but the most popular files are nearly all either decoys or related to LimeWire, and are noticeably overrepresented in the distribution.

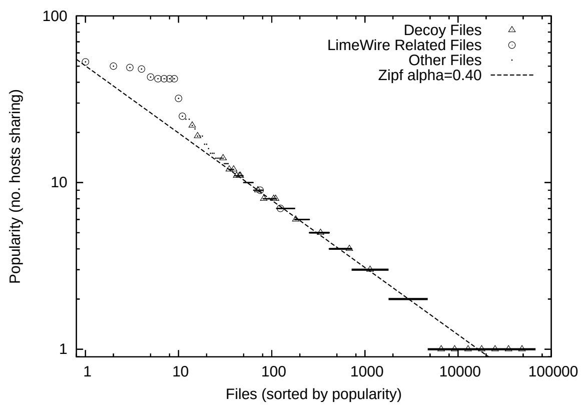

Figure 7: File popularity by number of hosts sharing a file is Zipf over the entire range, with the exception of certain files included in our software distribution.

peers. Figure 5 shows the number of votes cast by each type of client at the end of our study, and highlights the small number of votes necessary to become connected, and the relative inactivity of clients that are not yet connected. Looking at the data over time, 70% of connected clients discover their first peer correlation after casting fewer than 18 votes, which is the median for honest clients in the figure above.

3.2 Files in Credence

In this section we examine the overall distribution of files in the network, estimate the extent of pollution in our data set and its impact on clients, and examine how this pollution helps shape the overall structure of the Credence trust network. In Section 3.3, we show that clients are able to avoid this pollution using Credence's object authenticity rankings.

We collected lists of shared files from a set of 681 Credence clients. These users advertise a total of 84,838 files, of which 67,794 are unique, roughly following a Zipf popularity distribution as shown in Figure 6.

Decoys and Artifacts

The most frequently shared files are noticeably overrepresented, and are shared an order of magnitude more often than would be predicted by a strict Zipf distribution. Here we show that this deviation is due almost entirely to the effect of decoy attacks, demonstrating the substantial impact of pollution on the overall characteristics of the filesharing network. The remaining discrepancy can be explained by several files that appear to be widely, but inadvertently, shared by Credence users.

We manually examined all files shared under at least nine distinct names, and found all to be clear examples of decoy attacks – typically, movies or pictures containing advertisements (not surprisingly, the advertisements were often misleading; a movie containing an ad for a "free" iPod giveaway is highly prevalent, in part because a set of nodes on the Gnutella network sends it in response to every query and leads inattentive users to download it). Some of the other highly prevalent files are artifacts of the LimeWire and Credence software distributions, such as icons, source files, and software updates. Figure 6 labels all known decoy and LimeWire related files, discovered through manual examination of the 100 most frequently encountered files. These cases account for nearly all of the deviation from a strict Zipf popularity distribution.

The apparent popularity of many decoy files does not come from wide distribution in the network, but rather from a small number of hosts sharing the decoy files many times under different names. We can more accurately see the effects of decoy attacks on the network by measuring file popularity as the number of hosts sharing the file, shown in Figure 7, rather than as the number of replicas of the file. The figure shows two distinct types of decoy files. Those decoys that remain at the left of the distribution have spread to several peers in the network. The remaining decoys are shifted to the low end of the popularity distribution, meaning that they are shared by only a few peers. This shows that the apparently high popularity of some decoy files observed in Figure 6 is due to large-scale replication on a small number of clients, and provides clear evidence of malicious pollution in the underlying network.

Overall, our sharing data provides strong evidence that ongoing decoy attacks are targeting Credence clients, and that some of these decoy files are propagating successfully through the network. Other decoys rely on large scale replication by only a small number of clients, and these files do not propagate widely. We account for the difference in propagation in Section 3.3, showing that the larger decoy attacks are being suppressed by voting from honest Credence participants.

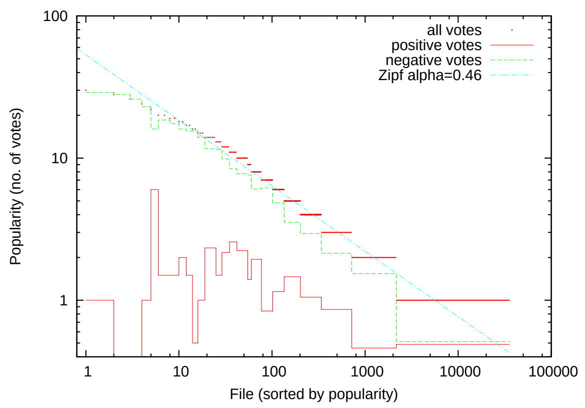

Figure 8: File popularity by number of votes received is Zipf. The most popular files, all decoys and artifacts, are slightly underrepresented. Solid and dashed lines show the number of positive and negative votes for each file, averaged within each popularity level, and indicate that positive votes are spread evenly across the entire range of file polarities, while negative votes cluster especially at the most popular files.

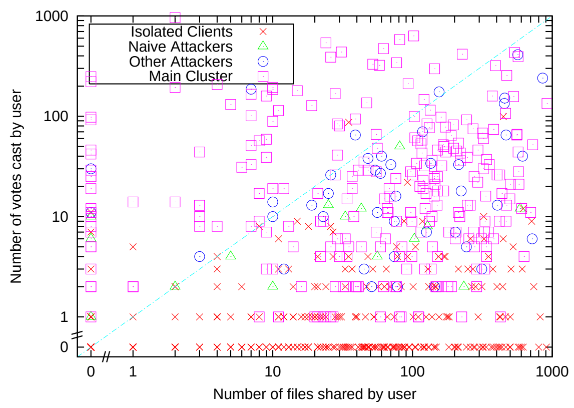

Figure 9: Credence user voting and sharing rate are largely independent, with users outside the main cluster found mainly along the x axis with no voting activity, and in the lower right of the graph with less voting activity than sharing activity.

File Voting Popularity

Our voting data set comprises 39,761 votes cast on 35,690 unique files. The file most often voted on had 30 votes, while 33,530 files received only a single vote. The distribution of votes among files follows a Zipf popularity distribution, as shown in Figure 8, with the exception of the most popular files which are slightly underrepresented. Also shown is a breakdown of positive and negative voting frequency for each popularity level. This data shows that positive votes are spread across files fairly evenly, without regard to the popularity of the file. Negative votes, however, have a more skewed distribution, with many negative votes concentrated on a small fraction of all files.

These results indicate that many Credence users are encountering the same polluted files, consistent with a deliberate decoy attack. We observe in our data set that a few particular decoys are extremely common in search results, and these files are precisely those shown as the most popular in Figure 6, indicating that users are voting on those decoys that most affect them.

Voting is Independent of Sharing

A key design goal of Credence is to ensure that users can control the factors that influence their network reputation, and that users rely only on relevant data when judging the authenticity of files. Past systems have relied on sharing as proxy for users' explicit evaluations of files. We show in this section that not only are sharing and voting behavior largely independent of each other in our data set, but that they are often contradictory as well. This validates our reliance on explicit voting rather than implicit sharing indicators, and confirms that many users do not effectively monitor their sharing behavior.

Comparing the sharing and voting activity of individual users in Figure 9 shows that there is little correlation between the number of files voted on and the number of files shared, especially among the users in the large central cluster. Notably, nearly all Credence users outside the central cluster display much more sharing than voting behavior, whether the user is disconnected, or connected to the central cluster through purely or mostly negative correlations. This confirms that users who vote infrequently will tend to find themselves isolated in the reputation network, and provides incentive for honest users to participate in voting.

Voting Can Contradict Sharing

There is some overlap in voting and sharing activity of individual clients, but we find that a client's explicit votes often directly contradict the implicit indicators that sharing would have given. From our data, we can extract some 1754 instances spread over 123 users of a single user sharing and voting on the same file. A rational, honest user would only share files they vote positively on. This behavior accounts for roughly two thirds of the observed files. The remaining third of cases represent contradictory behavior, where a client has voted down a file they are also sharing. Interestingly, strictly rational behavior is observed on less than half (46%) of the users in the set. More surprising however is that nearly three quarters (72%) of the users display at least one case of inconsistent behavior, actively sharing files they explicitly voted against. This justifies our reliance on explicit voting rather than implicit sharing as an indicator of a user's judgment of file authenticity.

3.3 End-to-End Performance

As an end-to-end test of the effectiveness and utility of Credence, we examine how Credence performs when users execute typical queries in the Gnutella network, and how the Credence algorithms modify the relative ordering of the search results returned to the user. We used a load generator to repeatedly query the Gnutella network for typical keywords over a 24 hour period, and logged the search results returned. Query strings were generated from our earlier data set by extracting the four longest words from each file name shared by a Credence user. We selected terms in this way to mimic the interests of actual Credence users. This should not bias our query results, as sharing and voting behavior within Credence clients are essentially unrelated to each other, as our previous analysis has shown.

Queries that returned no search results were discarded. Each search result returned to our server consists of a file hash, a file name, various quality metrics, and a set of network addresses for the file. Using our voting data set, we matched the file hits returned to votes found in the data set generated by our crawler.

In all, Credence was able to provide some input to the search result ordering for 50.2% of the queries. Of the queries for which Credence could provide no input, the majority (79%) matched two or fewer files in the network. Of all the query replies received, more than a third were for files that had been voted on by Credence users. This result is quite encouraging, considering the very small number of Credence voters in comparison to the size of Gnutella network being queried.

Resistance to Decoy Attacks

In the above analysis, the distribution of query replies is skewed due to the popularity of certain files. Here, we examine the impact of decoy attacks on our query results and the ability of Credence clients to identify the decoy attacks encountered by our load generator. The specificity and scale of decoy attacks vary: some decoys are returned only for specific queries, while others are encountered for nearly all queries, and in both cases the number of responses naming the decoy can range from just one to several hundred. Thus, among the entire set of search results returned to our engine, only 12% contained unique file hashes, and only 1% of these are files for which Credence users have cast votes. In other words, although Credence users voted on only a small fraction of files encountered, less than 80 in all, these files represent more than a third of the total query replies. In previous sections we showed that Credence users tend to vote negatively more often on a small fraction of files. This suggests that many Credence users are likely being affected by a few very large decoy attacks, and are responding by voting negatively on them.

Using the list of decoy files identified by hand from our earlier analysis, we examine the impact of decoy attacks on filesharing search results, and characterize Credence's ability to identify such files when they are encountered. For this experiment, we selected individual clients in the central cluster to serve as vantage points, and computed how the clients would score each search result. In all cases where a decoy was encountered during a query, the clients were able to successfully identify 246 (75%) as pollution on average. The clients were not able to identify as pollution the remaining 82 (25%) decoy attacks, since no one appears to have voted on them. The deviation in results between clients was near zero, because the negative votes for the decoys come from peers that are weighted positively by honest clients.

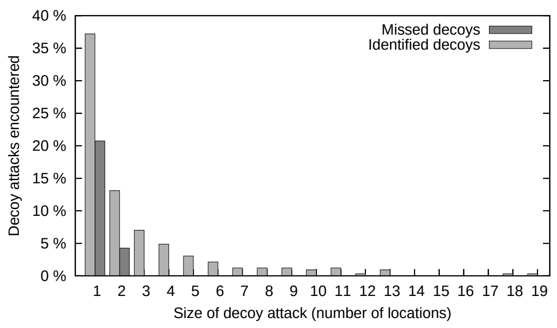

Figure 10: Distribution of the size of decoy attacks encountered, showing fraction of attacks successfully identified as pollution and those missed by Credence clients in the main cluster. Error bars are not visible, because these clients show nearly identical performance in identifying the decoys encountered.

Resistance to Collusion

Not all decoy attacks have equal impact on Credence users. In this section, we show that the decoys missed by Credence are those that have little impact on users in any case. Figure 10 shows the distribution of the sizes of each decoy result encountered, measured in terms of the number of locations purportedly offering the file. The data shows that any decoy offered by three or more peers is successfully identified. Credence misses several smaller decoy attacks, but even at these low levels still identifies nearly twice as many as it misses.

When Credence fails to identify some search results as polluted or authentic, it sorts these results according to their apparent popularity, just as non-Credence clients do. Larger attacks are more likely to rise to the top of such rankings, attracting the attention and negative votes of honest users, while the smaller attacks are more likely to be ignored by users. This tendency accounts for the lack of votes for such small decoy attacks.

We can conclude from these results that Credence users in the large central cluster of honest users are able to identify most decoy files as pollution by virtue of their own negative votes, or by the negative votes of other honest users, and that Credence is especially successful when attacks become sufficiently damaging to draw the attention of typical users.

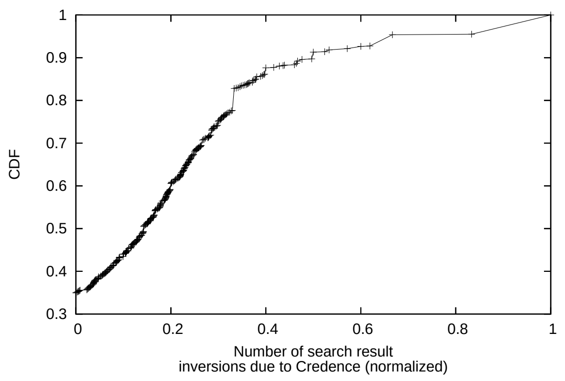

Figure 11: CDF of the normalized number of inversions in the search result rankings due to Credence. Credence considers legacy ordering at least 15% incorrect for half of the queries.

Ranking Performance

In addition to searching for and collecting votes for individual search results selected by the user, Credence is able to estimate file authenticity based on votes in the client's local vote database, and modify the sorting order of the results accordingly. Here we show that Credence's authenticity estimates have a significant impact on the presentation of typical search results. Specifically, we look at how the ordering in a non-Credence client, ranking files by the number of peers sharing the file, differs from the ordering in a Credence client, ranking files using known votes weighted by known peer correlations.

We use the number of inversions required to transform one sorted listing to the other, normalized to the range $[0, 1]$, as a measure of difference. A score of 0 would indicate that Credence's result ordering coincides exactly with the non-Credence ordering, a score of 1 indicates that the Credence client orders the list exactly backwards compared to the legacy ranking, and a score of 0.5 would result if the legacy ranking were purely random with respect to Credence's ranking. Credence's default behavior of sorting according to the legacy ordering when no votes are available introduces a conservative bias towards zero in this difference metric. Figure 11 shows a CDF of the normalized number of inversions due to Credence, showing that the legacy ordering is no better than purely random, with respect to Credence's order, in about 10% of cases. We conclude that Credence can provide users with substantial credibility rankings in most cases.

3.4 Response to Attack

The design of Credence promotes subtle feedback loops and incentives that give rise to the resilience and overall dynamics shown in the previous sections. In our data set, we observe that polluted files that begin with an apparent high popularity, and a correspondingly high legacy ranking, are quickly discovered and voted down sufficiently often by honest users so as to put them at the bottom of the sorting order. An attacker's positive votes in this case will serve only to induce additional negative votes from honest participants, and more importantly, reveals the attacking peer to not be credible. This effect can be seen as the nodes in main cluster of the network identify peers with consistently strong negative correlations.

A client's correlation values play a key role in its resistance to attack, and renders many attacks ineffective. A naïve attacker that votes consistently in opposition to honest clients will find itself cut off from the main cluster of honest nodes. At the other extreme, an attacker that votes randomly also becomes isolated, since it will generate no correlations with other peers. A rational attacker must carefully trade off honest votes on some files with dishonest votes on others, and so is required to leak a certain amount of correct information to the network.

The strongest attack we have explored is a *whitewashing attack*, in which the attacker first votes honestly on a large set of sacrificial files before voting dishonestly on a smaller target set. In our study, we find that such attackers may be included or excluded depending on each client's perspective. An individual client excludes the attacker if, from the clients own perspective, the information gained from the attacker's honest votes is outweighed by the damage from the dishonest votes. In other instances, clients find that the attacker's negative votes does so little damage and provides so much honest information as to make it worthwhile to include the attacker as a partially credible peer. Client perspectives vary, making it more difficult for attackers to select the sacrificial files on which to vote honestly. Further, multiple independent attacks of this form can easily cancel each other, and lead to an overall net gain of honest information in the system.

A more complete discussion of this and other potential attack scenarios can be found in a previous work [24], based on simulations of the Credence protocol.

3.5 Credence Overhead

Credence performs various operations in the background, and so requires some processing and network resources on client machines. A client representative of the most active existing users, having cast 250 votes and having learned over time of an additional ten thousand votes, can expect to receive approximately 100 bytes per second of additional background traffic due to Credence. This is in comparison to LimeWire's approximately 60 bytes per second of incoming background traffic. Outbound traffic depends strongly on the popularity of the client's votes, the client's reputation, and Gnutella connectivity. Ongoing background processing of additional Gnutella queries and Credence gossip requests in the same scenario demands less than 1% of a 1.7GHz processor, while a complete recomputation of all correlations can be completed in under three seconds.

4 Related Work

Deployed peer-to-peer systems are known to be vulnerable to many forms of malicious activity. A recent study [3] gives clear evidence of intentional pollution attacks in four large filesharing networks, and discusses the lack of reliable tools available for peers to avoid this pollution. Similarly, Ross [14] finds evidence of rampant pollution in the decentralized Kazaa network.

Distributed Peer Reputation: Past work on peer-to-peer reputation focuses mainly on service differentiation, which refers to the ability of clients to make intelligent decisions about which peer among many to select for service. Typically, the goal of such systems is to discourage freeloaders by excluding them from the network, or optimize download performance by selecting high performing peers. Thus, past work relies mainly on peer reputation based on performance measures of peers, rather than object reputation as we propose.

Eigentrust [13] computes a single performance score for each peer, reflecting their past behavior in pairwise interactions. Although the protocol is distributed, it ultimately relies on a fixed set of trusted nodes at which it roots the computation of trust. Collusion can also disrupt straightforward eigenvalue computations, and techniques to make eigenvalue-based systems more resistant to collusion [28] rely heavily on centralized computation.

Other approaches [1, 25, 7, 4] enable each client to compute a personalized, rather than global, performance score for peers in the network, and also distinguish peer performance and peer credibility. A client considers a peer credible if the client and peer tend to agree on most performance observations made in the past. However, peers in such a system are unlikely to be found credible due simply to the wide variation in network perspectives, changing performance characteristics over time, locality-based peer selection, and the high degree of object replication in typical filesharing networks.

XRep [6] and X^{2}Rep [5] extend the work in [4] by additionally computing object reputations based on weighted peer voting, with the weights based on past voting behavior of peers. These protocols require peers to be online during each object evaluation phase in order to compute and transmit their votes, and do not share the computed weights among peers. Information sharing and offline operation are critical features for a peer-to-peer reputation system due to the sparse workloads and session lengths observed in peer-to-peer networks [21].

Alternative approaches to the freeloader problem have been proposed without resort to peer reputation. Gupta, Judge, and Ammar propose an economic model [12] where peers earn credits for participation and pay credits to gain service, and explore the tradeoffs between reliability and overhead when accounting is distributed in the network. Micropayments can be used to induce cooperation (e.g. [23, 26, 17]). Fair exchange systems [10] provide similar properties and incentives, but without relying on currency. These schemes do not address content pollution, and so are complementary to our approach.

Credence does not address freeloading or service differentiation problems, but rather content pollution. In a broad study of denial-of-service vulnerabilities in peer-to-peer networks, Dumitriu et al. [9] discuss pollution based attacks and factors that make them successful in existing networks, and note the tendency of clients to inadvertently share corrupted files. The authors estimate the potential impact of a general class of peer-based reputations systems, and find them to be insufficient to counter pollution-based attacks. Using simulation and epidemiological models of the spread of pollution in filesharing networks, Thommes and Coates [22] show that Credence's object-based approach can have a significant impact on network-wide pollution levels, even when only a fraction of participants use Credence.

Centralized Pollution Control: Problems similar to filesharing pollution have long been recognized in other domains, prior to the emergence of modern peer-to-peer networks. For instance, recommender systems (e.g. [20]) aim to distinguish wanted and unwanted content in Usenet and in other domains, and online marketplaces often provide some form of reputation system so buyers can avoid untrustworthy sellers. Although Credence is not a recommender system, our distributed flow-based correlation computations resemble the centralized recommender algorithms in [27] for use in online marketplaces. Guha et al. [11] examine how both positive and negative evaluations might be propagated through a web of pairwise observations, and we share a similar model of information propagation in Credence. In general, however, past approaches to pollution-like problems rely heavily on centralized components and are not directly applicable to peer-to-peer networks.

BitTorrent has remained relatively free of pollution, partly due to human moderation of BitTorrent lists and tight binding of trackers to nodes [19]. Recent trends indicate that decentralized tracking and auto-generated torrent lists are opening BitTorrent up to pollution [16]. Credence techniques can be applicable in such contexts.

5 Conclusions

Credence is a new approach for combating the widespread presence of decoys, malware, and other malicious content in peer-to-peer filesharing systems. The system provides incentives for peers to participate honestly in voting, enables peers to compute object reputations that reflect their authenticity, and is robust to coordinated attacks. We have made a complete Credence implementation, with source code, freely available. Data from a long-term study of the emerging properties of the deployed network suggests that Credence users are able to identify malicious filesharing activity and mitigate the impact of dishonest peers in the Credence reputation system.

The techniques we have developed in Credence are not specific to the Gnutella network in which they are currently deployed, but are applicable to a broader class of distributed systems. These systems are characterized by the need for users to make local trust decisions about networked objects and services, without a priori trust relationships imposed by a central authority. As critical network infrastructure services become more decentralized, conventional centralized trust decisions will necessarily become unsuitable. Such systems can benefit from the Credence approach.

Acknowledgments

This work was supported in part by NSF Career grant 0546568, and TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (CCF-0424422) and the following organizations: Cisco, ESCHER, HP, IBM, Intel, Microsoft, ORNL, Qualcomm, Pirelli, Sun and Symantec.

References

[1] S. Buchegger and J.-Y. L. Boudec. A Robust Reputation System for P2P and Mobile Ad-hoc Networks. In *Workshop on the Economics of Peer-to-Peer Systems*, Boston, MA, June 2004.

[2] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. Secure Routing for Structured Peer-to-Peer Overlay Networks. In *Symposium on Operating Systems Design and Implementation*, Boston, MA, December 2002.

[3] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks. In *ACM Conference on Electronic Commerce*, Vancouver, Canada, June 2005.

[4] F. Cornelli, E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Choosing Reputable Servents in a P2P Network. In *International World Wide Web Conference*, Honolulu, HI, May 2002.

[5] N. Curtis, R. Safavi-Naini, and W. Susilo. X^2Rep: Enhanced Trust Semantics for the XRep Protocol. In *Applied Cryptography and Network Security*, Yellow Mountain, China, June 2004.

[6] E. Damiani, S. 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 *ACM Conference on Computers and Communications Security*, Washington, DC, October 2002.

[7] R. Dingledine, M. Freedman, and D. Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *Workshop on Design Issues in Anonymity and Unobservability*, Berkeley, CA, July 2000.

[8] J. R. Douceur. The Sybil Attack. In *International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.

[9] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-Service Resilience in Peer-to-Peer File Sharing Systems. In *ACM SIGMETRICS*, Banff, Canada, June 2005.

[10] P. Gauthier, B. Bershad, and S. D. Gribble. Dealing with Cheaters in Anonymous Peer-to-Peer Networks. Technical Report 04-01-03, University of Washington, January 2004. Computer Science and Engineering.

[11] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins. Propagation of Trust and Distrust. In *International World Wide Web Conference*, New York, NY, May 2004.

[12] M. Gupta, P. Judge, and M. Ammar. A Reputation System for Peer-to-Peer Networks. In *ACM Intl. Workshop on Network and Operating System Support for Digital Audio and Video*, Monterey, CA, June 2003.

[13] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *International World Wide Web Conference*, Budapest, Hungary, May 2003.

[14] J. Liang, R. Kumar, Y. Xi, and K. W. Ross. Pollution in P2P File Sharing Systems. In *IEEE INFOCOM*, Miami, FL, March 2005.

[15] LimeWire. http://www.limewire.com/.

[16] T. Mennecke. New Breed of Corrupt Torrent Infiltrates BitTorrent, September 2005. http://slyck.com/news.php?story=296.

[17] MojoNation. http://www.mojonation.net/.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web, 1998. Stanford Digital Libraries Working Paper.

[19] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System. Technical Report PDS-2004-003, Delft University of Technology, April 2004.

[20] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *ACM Conference on Computer Supported Cooperative Work*, Chapel Hill, NC, October 1994.

[21] S. Saroiu, K. P. Gummadi, and S. D. Gribble. A Measurement Study of Peer-to-Peer File Sharing Systems. In *Multimedia Computing and Networking*, San Jose, CA, January 2002.

[22] R. Thommes and M. Coates. Epidemiological Models of Peer-to-Peer Viruses and Pollution. Technical report, McGill University, June 2005. Department of Electrical and Computer Engineering.

[23] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer. KARMA: A Secure Economic Framework for P2P Resource Sharing. In *Workshop on the Economics of Peer-to-Peer Systems*, Berkeley, CA, June 2003.

[24] K. Walsh and E. G. Sirer. Fighting Peer-to-Peer SPAM and Decoys with Object Reputation. In *Workshop on the Economics of Peer-to-Peer Systems*, Philadelphia, PA, August 2005.

[25] L. Xiong and L. Liu. PeerTrust: Supporting Reputation-Based Trust in Peer-to-Peer Communities. *IEEE Transactions on Knowledge and Data Engineering, Special Issue on Peer-to-Peer Based Data Management*, 16(7), July 2004.

[26] B. Yang and H. Garcia-Molina. PPay: Micropayments for Peer-to-Peer Systems. In *ACM Conference on Computers and Communications Security*, Washington, DC, October 2003.

[27] G. Zacharia, A. Moukas, and P. Maes. Collaborative Reputation Mechanisms in Electronic Marketplaces. In *Hawaii International Conference on System Sciences*, Maui, HI, January 1999.

[28] H. Zhang, A. Goel, R. Govindan, K. Mason, and B. V. Roy. Making Eigenvector-Based Reputation Systems Robust To Collusion. In *Workshop on Algorithms and Models for the Web-Graph*, Rome, Italy, October 2004.

[29] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A Secure Distributed On-line Certification Authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.