Defending against Peer-to-Peer Pollution Behaviors

Eric Chang
Rice University

Abstract

In this report, we present a survey and comparison of various models on defending against content pollution. We categorize the various schemes into some groups and discuss the application-level model performance of each group. At end of the report, we present some future aspects on defending against content pollution; moreover, some important and useful evaluation results are in the report.

1. Introduction

Peer-to-Peer (P2P) content sharing systems, such as KaZaa [21] and BitTorrent [22], and today's cloud systems [44, 45, 46, 47, 48, 49] are one of the most important services in today's Internet, particular in terms of shared resources, participating users, and traffic volume [11]. From the first infrastructures (Napster system), P2P content sharing systems rapidly evolved to become a cost-effective platform to legally share and distribute terabytes of music, movies and software [43].

As the consequence of this popularity, P2P content sharing systems have been target of several opportunistic and malicious user behavioral patterns. Some typical attacks towards to the P2P content sharing systems are: *Sybil attacks* [14], P2P Worms [41] and *content pollution* [1, 6, 7].

- **Sybil Attacks:** In a typical sybil attack, a malicious user obtains multiple fake identities and pretends to be multiple, distinct nodes in the system. By controlling a large fraction of the nodes in the system, the malicious user is able to "out vote" the honest users in collaborative tasks such as Byzantine failure defenses.
- **P2P Worms:** This is a new type of worms that leverage the popular P2P overlay applications pose a very serious threat to the Internet. Generally, the P2P worms can be grouped into two categories: passive P2P worms and active P2P worms. The passive P2P worm attack is launched either by copying such worms into a few P2P hosts' shared folders with attractive names, or by participating into the overlay and responding to queries with the index information of worms. Unable to identify the worm content items, participating P2P hosts download these worms unsuspected into their own shared folders, from which others may download later without knowing that these are worms, thus passively contributing to the propagation of worms.
- **P2P Content Pollution:** The active polluter tampers with the content of a specific file, thus rendering it useless, and makes it available in the content sharing systems with the same metadata of the original unpolluted file. Unsuspecting users end up downloading the polluted content, thus consuming resources with unwanted traffic. Moreover, they tend to leave downloaded polluted content items in their shared folders [15], thus passively contributing to the dissemination of pollution in the systems. The lack of robustness of current P2P content sharing systems to this type of malicious behavior is evidenced by recent reports of as much as 80% of the copies of popular files in KaZaa being polluted [1]. Therefore, there is a need for cost-effective solutions to isolate active polluters, while

motivating users to remove the polluted content they download, thus minimizing passive pollution dissemination.

In this report, we only focus on the content pollution in P2P content sharing systems. A number of efforts towards reducing pollution dissemination are available in the literatures. Some previous works focused on modeling and analyzing the dissemination for different pollution mechanisms [6, 11, 17, 18]. Others have given general ideas on how to reduce pollution [1, 15, 18, 59]. Practical solutions that have been actually implemented are presented in [3, 5, 8, 19, 20, 60], although a comprehensive evaluation of their trade-offs is still required [61, 62, 63].

The rest of this report is organized as follows. Section 2 discusses pollution definition in P2P content sharing systems. Section 3 describes the existing representative mechanisms on defending against content pollution and analyzes their availability in the current P2P content sharing systems. Section 4 will present the conclusions and future work.

2. Content Pollution in P2P Content Sharing Systems

The first evidences of content pollution in P2P content sharing systems appeared in a measurement study by J. Liang et al., which reported that up to 80% of the copies of popular files in KaZaA were polluted [1]. Content pollution and its implication on content availability are also further analyzed in [15].

Since these first studies, there have been several efforts towards addressing the content pollution problem. One group of studies characterized three basic mechanisms through which pollution can be introduced and disseminated in current systems:

- ***Decoy insertion*** consists of inserting a polluted version of a content item with the same metadata of the original file but a different identifier [1].
- ***Identifier corruption***, a much more pervasive mechanism [6], consists of exploiting weaknesses of current techniques for generation identifiers to insert corrupted objects with the same object (i.e., version) identifier of an unpolluted one.
- ***Index poisoning*** [7] consists of inserting bogus records in P2P indices, making the system fail to locate an existing content item.

Due to existing research works mainly focused on *decoy insertion* and *identifier corruption*, we introduce these two attacks particularly:

Decoy Insertion is a common sabotage mechanism used in P2P content sharing systems where corrupted versions of a particular content item are inserted into the network in order to make it difficult for users to find an uncorrupted version of that content [50, 51]. The corrupted content that is inserted, which we call a decoy, contains the same metadata as the polluted content. Usually, when a user searches for a content item, the program in the P2P content sharing systems groups the available copies into different versions, and presents the versions with the largest numbers of copies to the user. If users do not remove polluted content items as soon as they are downloaded, the decoys inserted into the system may be copied many times, making it difficult to find non-polluted content.

Identifier Corruption is a much more pervasive attack for P2P content sharing systems. In P2P content sharing systems, when client starts sharing a content item, a unique ID is associated with the content item. This ID allows applications to identify the content items that the clients share. Moreover, when a user receives the result of a search, the P2P client groups results with the same ID, so that a file can be downloaded from multiple sources simultaneously. This ID is generated by applying a hash function to the file content, and each system uses a different algorithm to create it. P2P content sharing systems assume that a version ID generated using the hash function is unique. However, it is possible to have two different files with the same ID. Some of the common used algorithms generate the version ID based only on parts of the content. In this context, a malicious peer can make changes on the parts of the content which are not used by the algorithm to generate the version ID, creating different content items with the same version IDs. When a user requests this content, it will receive a list of versions of that content, each one with a distinct version ID and a certain number of copies. Then, the user chooses a version, downloading pieces of different copies. If a downloaded piece is a corrupted part from the changed file, the entire download content will be corrupted. The hash corruption is the name we give to this technique to pollute content items.

Other studies have proposed analytical models, based on systems of differential equations and fluid modeling, to represent the pollution dissemination process [11, 18]. The impact of giving incentives to users to remove their polluted objects and of inserting polluted torrents into BitTorrent/Supernova is analyzed in [17] and [6], respectively.

3. Related Works Discussion

Recently, many reputation models have been proposed to address the problem of content pollution in P2P content sharing systems [42]. In general, these reputation models can be grouped into three categories: peer-based models, object-based models and hybrid models. In peer-based reputation models, e.g., EigenTrust [8], PeerTrust [9], DynamicTrust [23] and Scrubber [3], genuine users collectively identify malicious behaviors especially content pollution by calculating a reputation rank for each user, and then isolate these polluters from system. However, the studies in [24, 25] implied that these peer-based models are insufficient to defend against the content pollution attack. In object-based reputation models, e.g., Credence [24], genuine users determine the object authenticity through secure tabulation and management of endorsements from other users. Aiming at combining the benefits of both peer-based and object-based models, several hybrid reputation models, e.g., XRep [5], X^2Rep [4] and extended Scrubber [2], have been further presented. Nevertheless, due to the fact that most of the participating users in P2P sharing systems are rational in seeking to maximize their individual utilities, the reputation models are greatly penalized by the lack of reliable user cooperation.

Peer-based Reputation models

There has been some research on the peer-based reputation models from Aberer and Depotovic [16] introduced the first reputation management system for P2P content sharing systems. However, their trust metric simply summarizes the complaints a peer receives and content items and is very sensitive to the skewed distribution of the community and misbehaviors of peers.

P2PRep [26] proposed by Cornelli et al. is a P2P protocol where servants can keep track of information about the reputation of other peers and share them with others.

Their focus is to provide a protocol complementing existing P2P protocols, as demonstrated on top of Gnutella. However, there are no formalized trust metric and no experimental results in the paper validating their approach.

EigenTrust [8] proposed by Kamvar et al. is a representative peer-based reputation model. Their algorithm also focuses on the Gnutella like P2P content sharing systems and the main idea of theirs is to calculate a single performance reputation score for each peer, reflecting their past behavior in pair wise interactions. They want to base EigenTrust approach on the notion of transitive trust and addressed the collusion problem by assuming there are some peers in the network that can be pre-trusted. While the algorithm showed promising results against a variety of threat models, in my opinion, the pre-trusted peers may not be available in all cases and a more general approach is needed. Another shortcoming of their model is that the implementation of the algorithm is very complex and requires strong coordination and synchronization of peers.

PeerTrust [9] identifies five important trust factors and merges them into a general trust metric to quantify and assess the trustworthiness of peers, where a peer's trustworthiness is defined by an evaluation of the peer in terms of the level of reputation it receives in providing services to other peers. But their model solves the problems both malicious and good by the symmetry trend line. In other words, the sensitivities of PeerTrust do not depend on the number of accumulated experiences. The sensitivity of PeerTrust is the same for positive and negative changes, contradicting the principle of quick drop and lent raise of trust. Moreover, metric of PeerTrust can not penalize the oscillatory behavior of a malicious peer. Another critical problem of PeerTrust is that five parameters are headachy for input.

DynamicTrust [23] can be seen as the extend PeerTrust model. The metric of model is based on three factors: the short term reputation, the long term reputation and the penalty factors. The short term reputation is sensitive to fresh experiences and can quickly react to sudden misbehaving peers. The long term reputation equally weights the old and new experiences, assuring that lone term behavior will still be accounted for despite the possible short temporary excesses. Finally, the penalty factory is a measure of all the reputation that has been misused by a peer and has the role of inhibiting the oscillatory behavior. However, DynamicTrust also has several shortcomings: (1) Due to algorithm is very complex, the implement of the model become a problem; (2) It is too strict that the peer's reputation is the minimum of short term reputation and long term reputation; (3) Peer-based penalty factory is not a good solution for collusion attack.

Scrubber [3] peer-based reputation system is designed to identify and isolate malicious peers that actively disseminate polluted content. It also allows the rehabilitation of passive polluters (i.e., peers that share polluted content by negligence), by giving them an incentive to remove polluted content they have downloaded. Scrubber assigns each peer a reputation value, and the value is built from two components, namely, *Individual Experience* and *Peer Testimonial*. The *Individual Experience* represents the direct trust relationship between two peers, and can be calculated by penalty and reward. The same as DynamicTrust, in Scrubber, *Individual Experience* decrease faster than they increase. Even if a peer builds up a good reputation, it will quickly decrease once the peer starts acting maliciously. The *Peer Testimonial* on a peer captures the network opinion. Periodically, each peer sends a *Testimonial Query* to a number of randomly selected known peers to retrieve their *Individual Experience* with respect to the other peers. This information is used, before each new download, to update the *Peer Testimonial* of peer on each other peers as algorithms for Scrubber model. Before and after each download, peer updates the local reputation of each other peers using sum of changeable weights of *Individual Experience* and *Peer Testimonial*. Comparing with Credence [24], Scrubber has a much faster convergence to a competitive efficiency. But Scrubber can not defend collusion attack; especially the number of malicious peers is more than half of total peers. The same as other peer-based reputation systems, Scrubber cannot solve the content pollution completely.

Due to naturally shortcoming of the peer-based reputation models are insufficient to defend against the content pollution, next, we will mainly discuss the object-based and hybrid models that Credence [24], XRep [5], X^2Rep [4] and extended Scrubber [2].

From XRep, X^2Rep to Credence

XRep [5] is a notable reputation-based trust management system that can be straightforwardly piggybacked onto the Gnutella P2P content sharing protocol. XRep defines a secure protocol for the exchange of reputation information using the same message passing mechanisms as used in standard Gnutella *Query* and *QueryHit* exchanges. Thus, to provide XRep functionality, current Gnutella implementations require only modest modifications.

In XRep reputation information is associated with both peers and resources. XRep requires resources and peers to be uniquely identifiable. This is achieved by using the digest of a resource's content as the $resource_{id}$, and the digest of the public key of a peer as the $peer_{id}$. Using a cryptographic hash function ensures that the resources and the peers are uniquely identifiable.

When considering a content item download in Gnutella, the user selects the resource that best satisfies the request (using information such as the standard resource metadata string and offers connection speed). To assist the user in making the download decision, the network is 'polled' for any available reputation information on that resource and the peers that offer it. *Poll* messages are broadcast in the same way as *Gnutella Query messages*. All peers maintain repositories of their experiences (both good and bad) of resources they have downloaded and the peers with whom they have interacted. When a peer receives a *Poll* message, it checks its repositories for matching resource and peer identifiers. If it has some information to offer, it generates a set of binary votes based on its experiences, and returns them to the enquirer as a *PollReply* message.

The resource and peer votes are then processed and combined to produce a single value to the user as a reputation value for the download under consideration. Based on this reputation value, the user can make a decision whether or not to initiate a download.

Prior to the download, the offering peer for whom the highest peer reputation value was calculated is contacted directly to verify that it has really offered the target resource. This exchange is known as the *Best Peer Check*. The phases of XRep protocol is as follow:

Phase 1. A minor change to the Gnutella *Query* exchange is required; the resource identifier is added to the resource information contained in the *ResultSet* of the *QueryHit* message. This allows the polling peer to uniquely identify each offered resource.

Phase 2. The poll message consists of the identifier of the resource under consideration and the set of peers that offer it. Also included is a public key Pk_{poll} for which only the polling peer knows the private key. This may be a persistent key pair or a pair generated on the fly for each poll. Voting peers return their votes for some or all of the entities listed in the Poll message together with their IP address. The message is encrypted with Pk_{poll} to ensure confidentiality.

Phase 3. Once a set of votes are received, the polling peer must try to ensure the reliability of the votes and the honesty of the voters. The polling peer attempts this by carrying out the following steps.

- Decrypt each *PollReply* message and detect any tampering that may have taken place.
- Group votes from voters that are from the same IP network.
- Select a portion of peers from each group send a *TrueVote* challenge, from which the poller expects to receive a *TrueVoteReply*. This ensures that at least some of the votes are from genuine peers and not merely spoofed votes from non-existent IP addresses.

Phase 4. At stage the polling peer has evaluated trust for all the entities under consideration. The poller now carries out one further phase to ensure that the peer with the best trust evaluation exists and actually offers the resource. It is important for two reasons:

- A malicious peer is prevented from 'hijacking' the identity ($peer_{id}$) of a reputable peer.
- If it can be established that the resource has a good reputation and is offered by a peer with a good reputation, then it is possible to download that resource from any offerer and be assured that the resource is reliable. This can be considered as a load balancing technique.

The XRep protocol attempts to ensure the reliability of votes and protect against votes originating from colluding peers. The method of latter is by identifying voting cliques through clustering the votes that are provided by votes with the same network portion of their IP address. Such a correlation between colluding peers and IP addresses is tenuous [4]. XRep provides some safeguards against ID Stealth attacks. These attacks take place when a malicious peer 'hijacks' the identity ($peer_{id}$) of a reputable peer in order to deceive another peer into a malicious download. In such cases, the downloading peer believes it is interacting a peer with a good reputation. XRep provides safeguards against this attack in the *Best Peer Check* message exchange. Prior to downloading a resource, the downloading peer challenges the offering peer as to whether it really does offer the resource under consideration. The offering peer sends a response that is signed using its private key, and also supplies its public key. The downloading peer can be certain of the identity of the offering peer, firstly by verifying the signature of the message, and secondly by taking a cryptographic hash of the provided public key and comparing it against the $peer_{id}$ of the offering peer. If all verification is successful the downloading peer can initiate the download.

There are three basic strategies that can be employed by a single malicious peer or a group (collusion) of malicious peers with the intention of circumventing or degrading the reputation system in order to continue to share malicious resources unchallenged. We outline these strategies in the following:

Strategy A: This strategy is the simplest way for a malicious peer to share malicious resources. The peer actively participates in the network by offering good resources. However, occasionally the malicious peer will offer malicious resources. The malicious peer must carefully monitor the amount of good and bad resources it supplies in order to maintain a network-wide reputation that is sufficiently high for other peers to deem it trustable.

Strategy B: In this strategy a malicious agent attempts to degrade the quality of the reputation system by generating spurious votes when polled. The principal objective of this strategy may either be to simply degrade the correctness of reputation values to the point where these information are no longer trustable, or to attempt to increase the peer's relative standing by voting positively for itself and negatively for all others.

Strategy C: This strategy shares a similar objective with *Strategy B*. The principal differentiator is that more effort and resources are required on the part of the malicious peer(s) and such activity is harder to counteract by the reputation system. A group of peers systematically vote positively for each other whilst sharing malicious resources. Each peer in the group may also share some good resources in order to enhance its own reputation. The difficultly in detection of this strategy results from the evaluating peer receiving what appears to be a set of valid votes sent by real peers.

Due to above problems of XRep, X^{2}Rep is designed to address the weaknesses of XRep protocol.

X^{2}Rep [4] is an enhanced trust semantics algorithm that can be seamlessly incorporated into the XRep protocol. The contribution of X^{2}Rep is that the algorithm provides substantial improvements against the weaknesses of XRep using extensive simulations. X^{2}Rep gives more expressive power to peers to express their opinion about resources that they have downloaded and the peers that have downloaded from, and allows collusions of malicious peers to use a range of strategies and use the reputation to protect against these attacks. Due to the XRep protocol uses a complex process of challenge and response messages to ensure that a vote is supplied by a 'real' peer, the X^{2}Rep eliminate this complexity by employing extensive vote generation and evaluation system that make use of voter *credibility* information. In X^{2}Rep, voter credibility is an additional piece of information that helps an evaluating peer to determine the trustworthiness of a voter's vote through the evaluation of the voter's previous voting activity. X^{2}Rep can be divided by four logical parts: (1) Local Reputation Repository; (2) Voting; (3) Evaluating Ratings for Downloads and (4) Updating State on a Peer. The principle of each part is as follow:

Local Reputation Repository: Each peer will store data expressing its experiences with peers and resources that it has interacted with. For each downloaded resource, a pair can be stored with an identification of the resource ID and a real value between 0 (poor or malicious) and 1 (good), that is a measure of satisfaction of the peer with the resource. For each peer P_j that P_i has interacted with, P_i maintains a vector of length n storing its past n experiences with that peer. The peer *Experience Vector* v_{ij} is denoted by $v_{ij} = (P_j, (q_{ij,1}, q_{ij,2}, q_{ij,3}, \ldots, q_{ij,n}))$ where $q_{ij,k}$, $k = 1, \ldots, n$ are real values between 0 (poor or malicious) and 1 (good). On completion of each transaction with the peer P_j, P_i evaluates the transaction and generates a number that reflects his satisfaction and appends it to the end of the Experience Vector associated with peer P_j. The vector stores the results of the most recent n experiences and so as new experiences are appended the oldest ones are removed. During the initialization phase all data items will be set to zero.

Voting: In X^{2}Rep, voting can be divided two categories that *Resource Vote* and *Peer Vote*.

The former means that vote of P_i for a resource with resource ID. This allows the polling peer to learn precisely how the voting peer rated the resource. *Peer Vote* is that voting for a peer uses the content of the *Experience Vector* associated with that peer. This information will be used to generate a vote that is a number in the interval [0, 1]. The method of *Peer Vote* is very easy that calculate the average value of *Experience Vector* between two peers.

Evaluating Ratings for Downloads: After a specified time period, the polling peer will have received zero or more *PollReply* messages. The peer now must convert these votes into an evaluation for a possible transaction. X^2Rep presents a parameter called credibility c_{ij} that is given by the peer P_i for the peer P_j that has provided votes in previous transactions and will be stored in the *Local Credibility Repository* of the peer P_i. Credibility c_{ij} is a real number in the interval [0, 1] and is initialized to zero for an unknown peer. Credibility values will be used to adjust peers' votes to either peer vote or resource vote for the current download. A peer P_i that sent a resource vote u_{ik} to polling peer P_k, and the polling peer will reply his vote u_k as 1 or 0. P_i can calculate the $u_k = u_k c_{kj}$ and store this value as resource vote of polling peer P_k. After collecting all vote, peer P_i can calculate the *Resource Trust Value* $RT = \sum u_k$. A peer P_i, that sent a peer vote v_{ij} about offering peer P_j to peer P_l , will have the *Adjusted Peer Vote* v_{ij} as: $v_{ij} = v_{ij}c_{li}$. *Peer Trust Value* $PT = \sum v_{ij}$. The final trust value presented to the user will be a combination of the resource and peer trust values. The simplest approach would be to find the average of the two values. Users can use trust categories combined with other criteria, for example accepted level of risk, to make the final decision.

Updating State on a Peer: After the completion of a transaction, the state information of the downloading peer must be update. This includes the following.

- Updating the downloading peer's *Local Reputation Repository* with peer and resource evaluation values.
- For each peer that provided a vote:
  - If the voting peer P_i provided an accurate vote $c_i = c_i + 0.05$.
  - If the voting peer P_i provided an inaccurate vote $c_i = 0$.

In my opinion, reducing c_i to zero for a single inaccurate vote may seem harsh. In fact, peer can control the trust value according to experience vector. Another shortcoming, resource trust and peer trust should have to be valuable information, however in X^2Rep, they are calculated or combined very hash. But I think we can get several important enlights from X^2Rep, for example, peer vote and experience vector. These information can be used of calculating some better results.

Credence [24], which is the best paper of *NSDI'06*, introduces a decentralized distributed system, where users assign reputations to the objects they download regarding their authenticity. It is based on a distributed vote gathering protocol for disseminating the object reputations in the network, and on a correlation scheme which gives more weight to votes from like-minded peers.

The system works as follows. Before peer A downloads an object o, it issues a vote-gather in the network to collect votes about o, providing o's identifier. Collected votes are either -1, if the voter considers o polluted, or +1, otherwise. The *object reputation* is computed, weighting each vote by the relationship A has had with the vote owner.

The relationship between two peers, expressed by the correlation of their voting histories, captures whether they tend to vote identically (positive correlation), inversely (negative correlation) or whether their voting histories are uncorrelated. The correlation between peers *A* and *B* is computed as $\theta(A,B)=(p-ab)/\sqrt{ab(1-a)(1-b)}$, where *a (b)* is the fraction of positive votes given by *A (B)* in the past, and *p* the fraction where both peers vote positively. When computing the reputation of an object, peer *A* weights the vote from peer *B* by *r(A, B)*, which is equal to $\theta(A,B)$ if $|\theta(A,B)| >= 0.5$, and 0 otherwise (i.e., *A* and *B* disregard each other's votes if they have uncorrelated voting histories). 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 un-weighted 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 object reputation is interpreted as a personalized estimate of the authenticity of the object, and can be used to make a more informed decision to accept or reject the object.

Each peer stores all collected votes in a local *vote database*. All strong correlations (i.e., $|\theta(A,B)| >= 0.5$) are also stored in a local *correlation table*, which is periodically updated. But pairwise correlations cannot robustly evaluate the relationship between a client and peers having only a few interests in common with the client. Credence overcomes 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 this peer and a more distant peer, to be combined into an estimate of the relationship between the client and 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. So, 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. Considering expensive of calculating the so large graph, Credence also periodically runs the *gossip protocol*. As a simplification and optimization in implementation of the Credence, 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. The calculation 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 is a novel object-based reputation framework, but there are several shortcomings existing in it. In my opinion, the mainly future works about it are as follow:

- If there is a new peer joins the networks, the correlation between it and the other peers will be built after downloading several content items and these operations can create some pollution for the peer and the whole networks.
- Credence proposes two strategies about auditing to protect the reliability of its local trust graph against peers that lie about correlations when exchanging information. But the auditing is not implemented in the deployed version of the Credence software, and whether we can research a novel metric to protect the authenticity of exchanging information.
- There are some another factor can be considered in the model, such as time, social and so on. If some of above factors can be introduced in Credence or the other model, the correlation trustiness will be get a better value.
- Collusion attacks can be defend by Credence, but if there are several disguised malicious peers who make trouble in the system. So can we design a new method to defend disguised peers.
- Credence present an object-based reputation framework to address decoy insertion, however, it cannot defend identifier corruption attack. Can we research a new method to help Credence to address the corruption attack [55].

Extended Scrubber [2] is built on the previous work of [3] and [24]. The design of the new hybrid peer and object reputation system is motivated by the improvement of Scrubber, because Scrubber is not always able to clean polluted objects shared by peers, despite a quick convergence. Credence, on the contrary, converges much more slowly, but is eventually able to isolate all polluted objects. So the authors of Scrubber combine the works of Scrubber [3] and Credence [24] to create the new hybrid reputation system – Extended Scrubber. The hybrid system has two key components, the *object reputation* and *peer reputation*. As in Credence, the object reputation is built from peer votes on the object authenticity. Before downloading an object *o*, peer *i* issues an *Object Voting Query* to collect votes on object *o*. The vote $V_{j(o)}$ of peer *j* on object *o* can be either -1 or +1. Peer *i* then calculates the reputation of object *o*, $R_{i(o)}$ as :

$$R_{i(o)} = \frac{\sum_{j \in N_{i(o)}} V_{j(o)} R_{i(j)}}{\sum_{j \in N_{i(o)}} R_{i(j)}}$$

The formula is very easy to understand. Extended Scrubber calculates the object reputation from peer *i* to object *o* using the reputation from peer *i* to peer *j* who votes the object *o* as the weights of vote of peer *j* on object *o*. So the reputation of peer *i* to object *o* means a weighted averaging calculating of the vote of peer to object.

The peer reputation component is an extension of Scrubber. It is also built from *Individual Experiences* and *Peer Testimonials*. *Peer Testimonials* formula is the same as the one in Scrubber but *Individual Experience* is introduced penalty factor both of polluter and liar. After calculating both *Individual Experiences* and *Peer Testimonials*, the *peer reputation* will be used to compute the *object reputation*.

Through the experiments of Extended Scrubber, we can find the Extended Scrubber is better than Credence on defending decoy insertion and address the identifier corruption attack that Credence cannot defend. Comparing the effectiveness against Credence system, Extended Scrubber performs better. We can summarize the advantages of Extended Scrubber that: (1) Extended Scrubber converges much faster to a maximum efficiency than the other systems, even under collusion and Sybil attacks; (2) Extended Scrubber is less sensitive to parameter setting than Scrubber, providing cost-effectiveness for various configurations and (3) Even in very uncooperative and unreliable communities, Extended Scrubber is still able to restrain pollution dissemination, but the others, such as Credence and Scrubber, cannot do so.

Of course, Extended Scrubber has several shortcomings: (1) Under traitor attack, the model cannot work as usual; (2) Due to lack of incentive metric, the capability of implementation of Extended Scrubber cannot be predicted; (3) The penalty of entity is inadequate, under badly collusion attack, Extended Scrubber system cannot perform as well.

Table3.1 presents comparing between a few representative reputation models.

Table 3.1 Comparing between Models

	EignTrust	***PeerTrust***	***Credence***	***Extended Scrubber***	***X^2Rep***
Introduced Time	2003	2004	2006	2007	2004
Type	Peer-Based	Peer-Based	Object-Based	Hybrid	Hybrid
Main Principle	Iterative Calculation of Relationship Matrix	Peers' Behaviors Capturing as Input Factors	Coincidence of Experiences of the Same Files Downloaded	Using Peers' Reputations as the Weights of Object Reputation Computing	Combining both Peers' Reputations Calculated by Average of Experience and Objects Reputation Calculated by Votes
Collusion Defense	low	low	high	high	medium
Research Goal	malicious users	malicious users	decoy insertion	content pollution	decoy insertion
Users Feedback	Yes	no	yes	yes	yes
Improvement	New Metric [27]	DynamicTrust [23]	Extended Scrubber [2]	None	Credence [24]
P2P Topology	Gnutella	Gnutella	Gnutella and Structure	Gnutella and Structure	Gnutella and Structure

In this section, we summarize the whole reputation models to defend malicious behaviors especially content pollution attacks. The next section we will present some conclusions and future work.

4. Conclusions and Future Work

Content pollution in P2P content sharing systems has become a serious problem, and several metrics are introduced to combating the problems. In the report, we summarize the existing reputation-based representative models for defending content pollution, and both advantages and shortcomings have been presented for every model. Through researching for these reputation models, we find that the metrics that are on peer-based reputation are not enough to defend content pollution, but object-based reputation model is a little roughness. So the future work will focus on the research of hybrid reputation model.

In my opinion, the future work of content pollution defending can be the aspects as follow:

- Find a novel object-based reputation framework to defend both decoy insertion and identifier corruption. The metrics will be referenced by existing theories that trusted recommending. Of course, the new model must be better and easier than Credence.
- Adapt some spam-defense mechanisms [50, 52, 53, 54, 56, 57, 58] into cloud spam detection and P2P network area.
- Hybrid models will be a future work on defending content pollution. From Extended Scrubber, we haven't found how effective it can address identifier corruption. So a effectively and reliability framework based on hybrid reputation model will be a good topic on solving the content pollution.
- Incentive mechanism is the critical metric for addressing the problem of content pollution, and as we know, there are few incentive mechanisms on defending this problem. How can design a reliability incentive mechanism to address the problem will be a good future work.
- Some shortcomings introduced in the report should have been addressed in future work; this kind of improvement work will create the new reputation model.

References

[1] J. Liang, R. Kumar, Y. Xi, and K. Ross. Pollution in P2P File Sharing Systems. In *INFOCOM*, 2005.

[2] C. Costa and J. Almeida. Reputation Systems for Fighting Pollution in Peer-to-Peer File Sharing Systems. In *IEEE P2P*, 2007.

[3] C. Costa, V. Soares, J. Almeida and V. Almeida. Fighting Pollution Dissemination in Peer-to-Peer Networks. In *ACM SAC*, 2007.

[4] N. Curtis, R. Safavi-Naini and W. Susilo. X^2Rep: Enhanced Trust Semantics for the XRep Protocol. In *ACNS*, 2004.

[5] E. Damiani, S. 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 Computer and Communications Security*, 2002.

[6] F. Benevenuto, C. Costa, M. Vasconcelos, V. Almeida, J. Almeida and M. Mowbray. Impact of Peer Incentives on the Dissemination of Polluted Content. In *ACM SAC*, 2006.

[7] J. Liang, N. Naoumov and K. Ross. The Index Poisoning Attack in P2P File-Sharing Systems. In *IEEE INFOCOM*, 2006.

[8] S. Kamvar, M. Schlosser and H. Carcia-Molina. The EigenTrust Algorithm for Reputation Management in P2P Networks. In *WWW*, 2003.

[9] L. Xiong and L. Liu. PeerTrust: Supporting Reputation-Based Trust for Peer-to-Peer Electronic Communities. *IEEE Trans. Knowl. Data Eng*., 2004.

[10] I. Stoica, R. Morris, D. Karger, F. Kaashoek and H. Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[11] R. Kumar, D. Yao, A. Bagchi, K. Ross and D. Rubenstein. Fluid Modeling of Pollution Proliferation in P2P Networks. In *ACM SIGMETRICS*, 2006.

[12] E. Adar and B. Huberman. Free Riding on Gnutella. *First Monday*, 2000.

[13] M. Feldman, K. Lai, I. Stocia and J. Chuang. Robust Incentive Techniques for Peer-to-Peer Networks. In *ACM Conf. on Electronic Commerce*, 2004.

[14] J. Douceur. The Sybil Attack. In *IPTPS*, 2002.

[15] N. Christin, A. Weigend and J. Chuang. Content Availability, Pollution and Poisoning in File Sharing Peer-to-Peer Networks. In *ACM Conf. on Electronic Commerce*, 2005.

[16] K. Aberer and Z. Despotovic. Managing Trust in a Peer-2-Peer Information System. In *International Conf on Information and Knowledge Management*, 2001.

[17] J. Pouwelse, P. Garbacki, D. Epema and H. Sips. The BitTorrent P2P File-Sharing System: Measurements and Analysis. In *IPTPS*, 2005.

[18] R. Thommes and M. Coates. Epidemiological Modeling of Peer-to-Peer Viruses and Pollution. In IEEE Infocom, 2006.

[19] J. Liang, N. Naoumov and K. Ross. Efficient Blacklisting and Pollution-Level Estimation in P2P File-Sharing Systems. In AINTEC, 2005.

[20] K. Walsh and E. Sirer. Fighting Peer-to-Peer Spam and Decoys with Object Reputation. In Economics of P2P Systems, 2005.

[21] KaZaa. http://www.fasttrack.com

[22] BitTorrent. http://www.bittorrent.com

[23] C. Duma and N. Shahmehri. Dynamic Trust Metrics for Peer-to-Peer System. In Workshop on Database and Expert Systems Applications, 2005.

[24] K. Walsh and E. Sirer. Experience with an Object Reputation System for Peer-to-Peer Filesharing. In NSDI, 2006.

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

[26] F. Cornelli, E. Damiani, S. Di Vimercati, S. Paraboschi and P. Samarati. Choosing Reputable Servents in a P2P Network. In WWW, 2002.

[27] D. Donato, M. Paniccia, M. Selis, C. Castillo, G. Cortese and S. Leonardi. New Metrics for Reputation Management in P2P Networks. In AIRWeb'07, 2007.

[28] N. Borisov. Computational Puzzles as Sybil Defenses. In Peer-to-Peer Computing, pages 171–176, 2006.

[29] R. Chen, W. Guo, L. Tang, J. Hu, and Z. Chen. Scalable Byzantine Fault Tolerant Public Key Authentication for Peer-to-Peer Networks. In Euro-Par, pages 601–610, 2008.

[30] P. Gauthier, B. Bershad, and S. D. Gribble. Dealing with Cheaters in Anonymous Peer-to-PPer Networks. In Technical Report of University of Washington, 2004.

[31] J. M. Kleinberg. The small-world phenomenon: an algorithm perspective. In STOC, 2000.

[32] J. Liang, N. Naoumov, and K. W. Ross. Efficient Blacklisting and Pollution-Level Estimation in P2P File-Sharing Systems. In AINTEC, 2005.

[33] A. Mislove, M. Marcon, P. K. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In Internet Measurement Comference, 2007.

[34] B. Parno, D. Wendlandt, E. Shi, A. Perrig, B. M. Maggs, and Y.-C. Hu. Portcullis: protecting connection setup from denial-of-capability attacks. In SIGCOMM, pages 289–300, 2007.

[35] J. Pouwelse, P. Garbacki, J.Wang, A. Bakker, J. Yang, A. Iosup, D. Epema, M. Reinders, M. van Steen, and H. Sips. Tribler: A Social-based Peer-to-Peer System. In IPTPS, 2006.

[36] P. Resnick, K. Kuwabara, R. Zeckhauser, and E. Friedman. Reputation systems. In Communications of the ACM, 2000.

[37] H. Rowaihy, W. Enck, P. McDaniel, and T. L. Porta. Limiting Sybil Attacks in Structured P2P Networks. In INFOCOM, pages 2596–2600, 2007.

[38] N. Tran, B. Min, J. Li, and L. Subramanian. Sybil-Resilient Online Content Voting. In NSDI, 2009.

[39] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao. SybilLimit: A Near-Optimal Social Network Defense against Sybil Attacks. In IEEE Symposium on Security and Privacy, pages 3–17, 2008.

[40] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. SybilGuard: defending against sybil attacks via social networks. In SIGCOMM, pages 267–278, 2006.

[41] L. Zhou, L. Zhang, F. McSherry, N. Immorlica, M. Costa and S. Chien. A First Look at Peer-to-Peer Worms: Threats and Defenses. In IPTPS' 05.

[42] Yonggang Wang, Ennan Zhai, Jian-bin Hu, and Zhong Chen. Claper: Recommend classical papers to beginners. In FSKD, August 2010.

[43] Ennan Zhai, Ruichuan Chen, Zhuhua Cai, Long Zhang, Huiping Sun, Eng Keong Lua, Sihan Qing, Liyong Tang, and Zhong Chen. Sorcery: Could we make P2P content sharing systems robust to de- ceivers? In 9th P2P, September 2009.

[44] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. An untold story of redundant clouds: Making your service deployment truly reliable. In 9th HotDep, November 2013.

[45] Ennan Zhai, Ruichuan Chen, David Isaac Wolinsky, and Bryan Ford. Heading off correlated fail- ures through Independence-as-a-service. In 11th OSDI, October 2014.

[46] Ennan Zhai, Liang Gu, and Yumei Hai. A risk- evaluation assisted system for service selection. In ICWS, July 2015.

[47] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. Sorcery: Overcoming deceptive votes in P2P content sharing systems. Peer-to-Peer Networking and Applications, 4(2):178–191, 2011.

[48] Ennan Zhai, David Isaac Wolinsky, Hongda Xiao, Hongqiang Liu, Xueyuan Su, and Bryan Ford. Auditing the Structural Reliability of the Clouds. Technical Report YALEU/DCS/TR-1479, Department of Computer Science, Yale University, 2013. Available at http://www.cs.yale. edu/homes/zhai-ennan/sra.pdf.

[49] Ming Zhong, Kai Shen, and Joel I. Seiferas. Replication degree customization for high availability. In EuroSys, pages 55–68, 2008.

[50] Bo Liu, Ennan Zhai, Huiping Sun, Yelu Chen and Zhong Chen. Filtering spam in social tagging system with dynamic behavior analysis. In ASONAM, Jul 2009.

[51] Thierry Titcheu Chekam, Ennan Zhai, Zhenhua Li, Yong Cui, and Kui Ren. On the synchronization bottleneck of openstack swift-like cloud storage. In INFOCOM, 2016.

[52] Jianchun Jiang, Liping Ding, Ennan Zhai, and Ting Yu. VRank: A context-aware approach to vulnerability scoring and ranking in SOA. In SERE, 2012.

[53] Yonggang Wang, Ennan Zhai, Eng Keong Lua, Jian-bin Hu, Zhong Chen. iSac: Intimacy based access control for social network sites. In UIC/ATC, 2012.

[54] Ennan Zhai, Qingni Shen, Yonggang Wang, Tao Yang, Liping Ding, Sihan Qing. SecGuard: Secure and practical integrity protection model for operating systems. In APWeb, 2011.

[55] Cong Sun, Ennan Zhai, Zhong Chen, and Jianfeng Ma. A multi-compositional enforcement on information flow security. In ICICS, 2011.

[56] Ennan Zhai, Liping Ding, and Sihan Qing. Towards a reliable spam-proof tagging system. In SSIRI, 2011.

[57] Yonggang Wang, Ennan Zhai, Cui Cao, Yongqiang Xie, Zhaojun Wang, Jian-bin Hu, and Zhong Chen. DSpam: Defending against spam in tagging systems via users' reliability. In ICPADS, 2010.

[58] Ennan Zhai, Huiping Sun, Sihan Qing, and Zhong Chen. SpamClean: Towards spam-free tagging systems. In CSE(4), 2009.

[59] Ennan Zhai, Zhenhua Li, Zhenyu Li, Fan Wu, and Guihai Chen. Resisting tag spam by leveraging implicit user behaviors. In VLDB, 2017.

[60] Ennan Zhai, Ruichuan Chen, Eng Keong Lua, Long Zhang, Huiping Sun, Zhuhua Cai, Sihan Qing, and Zhong Chen. SpamResist: making peer-to-peer tagging systems robust to spam. In GlobalCom, 2009.

[61] He Xiao, Zhenhua Li, Ennan Zhai, Tianyin Xu, Yang Li, Yunhao Liu, Quanlu Zhang, and Yao Liu. Towards Web-based delta synchronization for cloud storage services. In FAST, 2018.

[62] Ennan Zhai, Ruzica Piskac, Ronghui Gu, Xun Lao, and Xi Wang. An Auditing Language for Preventing Correlated Failures in the Cloud. In OOPSLA, 2017.

[63] Ennan Zhai, David Isaac Wolinsky, Ruichuan Chen, Ewa Syta, Chao Teng, and Bryan Ford. AnonRep: Towards tracking-resistant anonymous reputation. In NSDI, 2016.