WINNOWING : Protecting P2P Systems Against Pollution By Cooperative Index Filtering

Kyuyong Shin, Douglas S. Reeves, Injong Rhee, Yoonki Song
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-8206
Email: kshin2, reeves, rhee, ysong2@ncsu.edu

***Abstract*—Pollution (i.e., sharing corrupted files, or contaminating index information with bogus index records) is a *de facto* problem in many file sharing Peer-to-Peer (P2P) systems. Since pollution squanders network resources and frustrates users with unprofitable downloads (due to polluted files) and unproductive download requests (due to bogus index records), the future success of file sharing based P2P systems is questionable unless properly addressed.**

In this paper, we propose a novel anti-pollution scheme called *winnowing*. Winnowing aims to purify the index records (i.e. the information on files or the publishers) held by each index node in the system, so that download attempts based on these index records are more likely to yield satisfactory results. To attain this goal, index nodes block bogus publish messages by verifying the publisher and the contents of the publish message upon receipt of a keyword or content publish message. Second, index nodes collect feedback from the users who have downloaded files via their index records. The collected feedback is then processed and reflected in the matching index record in a novel way. Careful consideration is given to reducing the impact of false feedback, and malicious index nodes.

Publish message verification has been implemented on top of the latest eMule client and extensive data has been collected from the Kad network, using this modified client. The measurement results are summarized in this paper. The the findings from the measurement study are incorporated into our analytical model, which is used to investigate the performance of user feedback mediation. The model demonstrates the effectiveness of user feedback mediation: fast convergence to near-optimal performance and insensitivity to various pollution attacks including the attacks which attempt to bypass winnowing.

I. Introduction

Peer-to-Peer (P2P) systems have emerged as a dominant way to exchange information in the Internet. Unfortunately, the information currently shared in P2P systems frequently turns out to be mislabeled or corrupted, wasting network bandwidth, user time, and storage space. For example, Liang et al. [1] showed that up to 80% of the copies of popular files in the KaZaA network are contaminated. This contamination is caused in part by pollution techniques that are intentionally introduced by copyright holders [1–3]. Because pollution was not considered in the design of P2P systems, they are highly vulnerable to such intentional attacks [2].

Substantial research has been done on the prevention of pollution in P2P systems. Several modeling techniques [3–5] have been introduced, which are useful in understanding the proliferation of pollution. Some previous work [1, 2, 6, 7] has given general ideas on how pollution can be reduced. Peer reputations [8, 9] and object reputations [10] are the major solutions proposed thus far in this literature. Unfortunately, no reputation technique to date has been notably successful in practice at preventing or minimizing pollution. There may be multiple reasons for this failure, including the cold start problem for newcomers [11], vulnerability to Sybil attacks [1, 12, 13], and the complexity of implementation of proposed reputation systems [1, 10]. As a result, it has been found that a random based selection approach is most effective under severe pollution [4]. In addition, the fact that the version popularity of a title follows a Zipf distribution in most file sharing P2P systems [1] strongly indicates that users tend to select a file based on its popularity, which is also confirmed by our measurements on the Kad network [14] as indicated in section VI-A.

This paper addresses the pollution problem in DHT-based P2P systems, with a focus on the Kad network. We propose a novel anti-pollution scheme, called *winnowing*. Winnowing directly aims to reduce the number of fruitless requests to download non-existing files and the number of downloads of polluted files by offering "clean" (valid, pointing to unpolluted files) index records to requesting peers. To attain this goal, index nodes first detect (and ignore) bogus publish messages by verifying the publisher and the contents of the publish message upon receipt of a publish message. Second, index nodes collect feedback from the users who have downloaded files via their index records. The feedback is processed in a novel way to increase the likelihood that index records pointing to polluted files will quickly be eliminated or disregarded. Careful consideration is given to reducing the impact of false feedback, and malicious index nodes.

The key insight behind winnowing is that the downloads of polluted files can be greatly reduced if index nodes aggressively maintain clean or valid index records. Publish message verification frustrates any attempts to insert bogus index records (pointing to non-existing files or publishers) into the system. User feedback mediation helps prospective users to efficiently find unpolluted files with the help of index nodes and other downloaders. Winnowing has the following benefits:

- It can sharply reduce the number of bogus index records pointing to non-existing files in a P2P system. Our measurements (during Summer 2008) have shown that

between 27% and 35% of the index records currently shard in the Kad network are bogus, which have been successfully filtered out by winnowing.
- It is easy to implement, completely distributed, and scalable. Winnowing does not require any trusted third parties, centralized servers, or complex security mechanisms. Random numbers (cryptographic nonces) are used instead for security purposes
- It converges quickly, and provides considerable immunity to pollution even when accompanied by Sybil attacks.
- It does not have the "cold start" problem as newcomers can immediately benefit from the efforts of index nodes and former downloaders to maintain clean index records.

We implement publish message verification on the latest version of eMule client [1]. Using this modified client, extensive data has been collected from the Kad network. The results are summarized in this paper and the findings from the measurement study are incorporated into our analytical model which is used to study the performance of user feedback mediation. The model demonstrates the effectiveness of user feedback mediation: fast convergence to near-optimal performance and insensitivity to various pollution attacks.

The remainder of this paper is as follows: In section II, some P2P terminology and a brief overview of the Kad network are introduced. Existing pollution strategies are discussed in the latter part of the section. Section III briefly describes and analyzes existing methods of anti-pollution approaches. Section IV presents the basic protocol description of winnowing. Some security considerations are discussed in section V. The results of the measurement and analysis are presented in section VI and VII respectively. Section VIII discusses limitations of the proposed method and possible solutions. Finally, section IX concludes this paper.

II. Background

In this section, some P2P terminology and a brief overview of the Kad network, the largest DHT-based P2P network in existence, will be introduced, emphasizing on the publish and search mechanisms. Existing pollution strategies will be discussed in the latter part of this section.

A. Basic P2P Terminology

Here, we investigate pollution problems in the DHT-based file sharing P2P networks, focusing on the distribution of movies or songs. A specific file, whether a movie or song or something else, is referred to as a *title*. There may be different *versions* of a title, and for one version of a title, multiple *copies* could exist if the version is downloaded and shared by many users in the network. There can also be many *decoys*, which appear to be versions of the title, but which do not in fact exist in the network, or which contain corrupted/lower-quality content. Each file has *metadata*, which is structured information that describes the file. Metadata includes the file name, the file size, and the file type or format, etc [1, 14]. A *keyword* is a single token extracted from the metadata of a file, usually from the file name. It must be composed of at least three characters [14]. Each file might have many keywords. A *key* is an identifier used to store and retrieve information in and from the DHT, and has the same bit size as all Kad_IDs. There are two types of keys, a *content key* and a *keyword key*. A content key may be generated by hashing the entire content of a file, whereas a keyword key is obtained by hashing a keyword of the file. A *user* is an operator of a P2P client application, whereas a *peer* (or interchangeably a *node*) is the client application itself. A user is called a *content owner* (or alternatively a *publisher*) when the user contributes a file to the P2P network. A user downloading a file from the the network is referred to as a *downloader*.

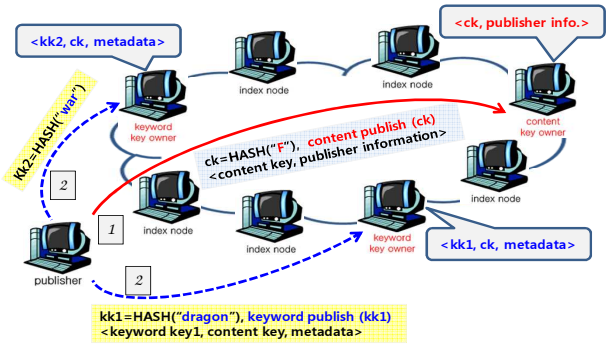


Fig. 1. A high level abstract overview of the publish mechanism in a DHT-based P2P system. In this example, we assume that the publisher wants to publish a movie file, "Dragon War.mpg". First, the publisher hashes the entire content to get the content key (ck) of the file and sends content publish messages to the content key owner(s). Second, the publisher hashes each keyword (i.e. "dragon" and "war") to get a keyword key (kk) and sends keyword publish messages to the keyword key owner(s) of each keyword.

B. Publishing and Retrieving

Publishing is used to *index* the information about a file or a publisher to the node(s) in charge of a portion of the Kad_ID space. A peer who wants to publish a file (i.e. publisher) first generates the content key by hashing the file contents. The publisher locates the node that has the closest Kad_ID to the content key[2] through iterative routing [14, 15]. Once determined, the closest node becomes a *content key owner* of the file. The content key owner updates its local index table with a record <content key, location information>, which is called a *content index record*. The location information includes the basic information on the publisher, such as the Kad_ID, the IP address, the service (TCP) port number, etc. In addition, the publisher hashes each keyword of the file to produce a keyword key, one per keyword. For each such keyword key, the publisher locates the closest node (*keyword key owner*) to the keyword key, in the same way as before,

[1] http://emulemorph.sourceforge.net/

[2] Currently, eMule uses a fuzzy algorithm which selects several peers as key owners (i.e. a tolerance zone). This means the IDs of the key owners are not necessarily the closest; the details do not affect our method, so we continue to use the terminology "closest" for simplicity.

and publishes the index information of the file. Each keyword key owner updates its index table with a record <keyword key, content key, metadata >; this is termed a *keyword index record*. For a given key, up to 10 closest nodes can be selected as key owners, referred to as a *tolerance zone*, to deal with high churn rates. This content publishing followed by keyword publishing is called a *2-level publishing* scheme [14]. Keys are periodically republished every 5 hours for content keys and 24 hours for keyword keys.

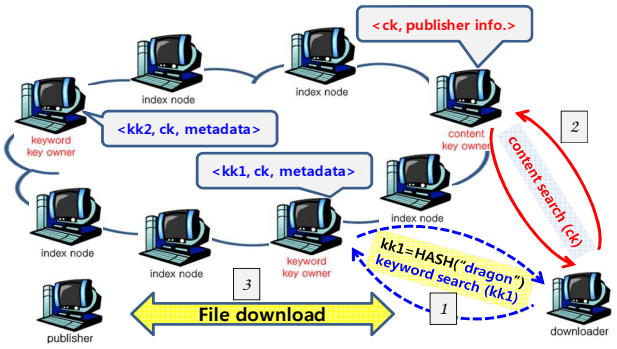


Fig. 2. A high level abstract overview of the search mechanism in a DHT-based P2P system. In this example, we assume that the downloader wants to find the movie file, "Dragon War.mpg", from the system. To find the file, first, the downloader hashes a possible keyword (e.g. "dragon" in this example) to get the keyword key. Then, sends keyword search messages to the keyword key owner(s). Second, the downloader selects one content key amongst all possible content keys returned by the keyword key owner(s) based on the metadata of each index record and sends content search messsages to the content key owner(s). Finally, the downloader downloads the file from the content owner(s) based on the publisher (i.e. location) information returned by the content key owner(s).

Retrieving file information from the DHT is essentially the inverse of publishing. A peer wishing to locate a file obtains a content key through a *keyword search*. This content key is then used to obtain location information via a *content search*. Finally, the user attempts to download the file from the publisher(s) based on the location information obtained.

C. Existing Pollution Attacks

Depending on the strategies taken by polluters, pollution can be classified into three categories; *content pollution*, *metadata pollution*, and *index pollution*.

Content pollution [1, 3, 7] changes the contents of a shared file to degrade its quality by inserting white noise, shuffling contents, inserting or substituting unrelated information, etc. Polluters can easily generate multiple fake files which have the same content hash value (i.e. the same content key) by exploiting the weakness of the current hash function of the Kad network, MD4 [16]. Metadata pollution [1,2] tampers with the metadata of a file rather than its content; as a result, peers downloading files based on that metadata will obtain content different from what they expect. Index nodes may unwittingly store index records pointing to polluted files affected by content or metadata pollution, which are referred to here as *corrupted index records*. Under these two strategies, polluters indeed upload the corrupted files at their own expenses of upload bandwidth.

As opposed to the above two, index pollution [17] directly attacks the index structure of a P2P network. On the one hand, a polluter can generate a random content key which might not match any file in the network, and publish the information <keyword key, *random* content key, metadata> to the matching keyword key owners. One the other hand, a polluter can generate an invalid IP address that does not correspond to any nodes (or an unavailable service port number), and publish the information <content key, *spurious* location information> to matching content key owners. Index nodes may unconsciously store index records pointing to non-existing files or publishers, which are referred to here as *bogus index records*. With index pollution, users fail to locate the target file (or the publisher) when relying upon the information in a bogus index record. Since index nodes in most file-sharing based P2P systems today do not authenticate or verify publish messages, polluters easily contaminate the index records [17]. Moreover, polluters don't need to transfer any files, which makes this pollution strategy prevalent in current P2P systems.

III. Related Work

Due to the importance of anti-pollution in P2P systems, there has been considerable work on the problem, starting with modeling and measurement. Dumitriu et al. [4] analyzed the influence of file- and network-targeted attacks on file sharing networks. They concluded that randomized reply selection provides considerable immunity to pollution attacks, but significantly hurts system performance in the absence of an attack. Kumar et al. [5] developed fluid models to capture the spread of polluted files, and user behavior. Lee et al. [3] measured the awareness of real users of polluted files and incorporated the results into their analytic model, concluding that awareness is a key factor in pollution dynamics.

One of the major directions in resolving the pollution problem is peer reputation [8, 9]. EigenTrust [8] calculates and maintains a global reputation for each peer, based on the opinion of all other peers who have interacted with that peer in the past. With Scrubber [9], a peer identifies malicious peers (who intentionally distribute polluted files) based on its direct experience, and the peer testimonials given by its neighbors. Scrubber also offers a peer rehabilitation mechanism by giving incentive to passive polluters who voluntarily remove polluted content. Its decentralized architecture facilitates easy deployments. Previous studies [4, 10], however, have shown that peer reputation approaches do not effectively deal with index pollution, particularly in a large P2P system. Winnowing can address index pollution with publish message verification as indicated in our measurement results in section VI.

Another major direction is object (or file) reputation [10, 18]. Credence [10] is a distributed vote gathering protocol in which a peer collects votes from its neighbors to assess the authenticity of the content it wants to download. This method determines correlation between two peers based on

the similarity of their voting histories. Practical difficulties of the method are the slow convergence time to find strong relationships [13], and the use of cryptographic keys to protect votes. Different from Credence, a downloader in winnowing needs not individually find such correlation with other users in that responsible index nodes (i.e. keyword key owners) evaluate the reputation of an object on behalf of individual downloaers. LIP [18] is a variant of object reputation. LIP uses a lifetime and popularity based ranking approach to filter out polluted files. However, the major assumption that "users tend to retain a real file longer and delete a fake file more quickly", is somewhat controversial if the discoveries in [3, 19] are considered. Costa et al. [13] take a hybrid approach between the peer reputation and object reputation methods.

Even though reputation systems are the main stream of anti-pollution in current P2P systems, there do exist some other approaches. BlackListing [6] is a way to find the IP address ranges that include the large majority of polluters. This is done by the use of crawlers specially designed to collect metadata from P2P networks. This methodology is efficient and effective. The centralized nature of this approach, however, could suffer from scalability, fault-tolerance, and security problems as mentioned in [9]. User Filtering [1, 19] could reduce a large fraction of pollution if users are aware of the pollution. This approach is, however, not robust to index pollution. The Query Result Ranking approach [2], Trust based approach [1], and Traffic Encryption approach [7] are also introduced in this literature without implementation.

Recently Wang et al. [20] introduced several attacks that exploit critical design weaknesses of the Kad routing, which hinders users from searching the index information itself. These attacks can be reduced by some minor modifications of the Kad routing mechanism, most of which are already implemented in up-to-date eMule clients (e.g. eMule 0.49a). Moreover, since they are inserting malicious index nodes in the system, winnowing can handle the attacks with the imbalanced feedback mechanism explained in section V-B.

IV. Our Approach : Winnowing

In this section we present the basic protocol description of *winnowing*. Winnowing aims to reduce *decoy* (i.e. bogus or corrupted) index records in P2P systems[3]. This is accomplished by *publish message verification* and *user feedback mediation*.

A. Publish Message Verification

To prevent index pollution (section II-C), index nodes in winnowing verify the publisher and the contents of each (keyword and content) publish message whenever received. The important principle here is that no information published is accepted at face value; it is confirmed by the index nodes before acceptance.

[3]We focus on keyword index records in this paper, for the sake of brevity of explanation.

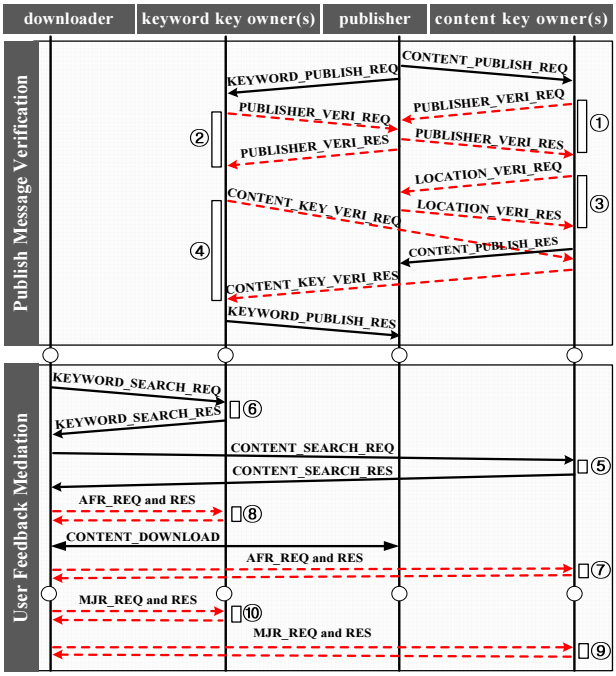


Fig. 3. Message processing in eMule with winnowing. Black solid arrows are the normal eMule messages; red dashed arrows are messages added by winnowing. Winnowing uses publish message verification (① ~④) to prevent index pollution and user feedback mediation (⑤ ~⑩) to address content and metadata pollution.

1) Publisher Verification: Index nodes must verify whether the publisher is a member of the network whenever a publish message is received from a publisher, termed *publisher verification*. Since the Kad network adopts the iterative routing, any peer could send publish messages directly to matching index nodes once identified. If not verified, one single polluter, which may or may not be a member of the network, could repeatedly send bogus publish messages to the index nodes. This would cause the index nodes to keep many bogus index records. Publisher verification can simply be accomplished by sending an application level hello message to the publisher (① and ②), and waiting for a response. Any publish messages from the publishers with no response are disregarded. Publisher verification could reduce the effect of IP spoofing presumably taken by polluters.

2) Message Content Verification: The publisher verification above cannot refrain polluters from sending publish messages with bogus information if they join the network as a legitimate member. To block the bogus publish messages received from such polluters, index nodes should attempt to verify the content of each publish message.

Each keyword key owner must verify the content key in a keyword publish message, which is called *content key verification*. A polluter can generate a random content key which might not match any file in the network, and publish the information <keyword key, *random* content key, metadata> to the matching keyword key owners, which is prevalent in current P2P systems [17]. Verification of the keyword publish

message is accomplished by issuing content search messages, using as a target the content key in the keyword publish message (④). A successful reply indicates the content key is valid [4]. Only valid content keys are indexed by matching keyword key owners. To bypass content key verification, the polluter may send content publish messages to the matching content owners corresponding with the random content key. This, however, costs even more time, efforts, and resources of the polluter. In addition, since each content publish message is again verified by the matching content key owner, the effect of this kind of attack is highly restricted. Moreover, winnowing will further filter out such bogus index record with user feedback mediation explained below. *Location verification*, performed by each content key owner (③), is very similar.

B. User Feedback Mediation

The verification steps explained above cannot prevent content and metadata pollution by themselves, as index nodes are unable to judge the authenticity or quality of file contents. Therefore, polluters can still store corrupted information (i.e. corrupted index records) in the P2P system. To deal with these pollution attacks, an index node makes use of *feedback* provided by the users who download files based on information maintained and provided by the index node. The index node collects this feedback (through voting) and disseminates it (in condensed form) to potential downloaders, along with the other index information. In fact, user feedback mediation is a kind of reputation and the use of reputation is a rational choice in a distributed system with malicious attackers. The novelty of our scheme lies in the way of implementation of the reputation.

1) User Feedback Collection: To collect user feedback, each keyword key owner maintains two kinds of lists: *keyword reference lists* (KRLs) and *content key voter lists* (CKVLs). One KRL is allocated to each keyword key and one CKVL is assigned to each content key. Once a keyword key owner receives a keyword search query from a peer P, it generates a random number R [5] and inserts <IP address, R > into the matching KRL (⑥). If there is feedback to a content key from a peer, the keyword key owner checks the KRL to see whether the peer has ever issued a keyword search query before. This can be done based on the peer's IP address and the random number presented as the proof of the peer's keyword search. If yes, then it checks the mapping CKVL to see whether the peer has previously cast its opinion to the content key. If not, then it reflects the feedback by increasing (for a positive vote) or decreasing (for a negative vote) the voting credits for the content key (⑧ and ⑩). Finally, the keyword key owner inserts <IP address, R> into the CKVL, preventing the peer from casting the same vote multiple times. User feedback collection done by each content key owner is virtually the same (⑤, ⑦, and ⑨).

2) User Feedback Reports: To help eliminate polluted content, some fraction of users will report their file download experiences to the appropriate index nodes. The user feedback report comprises the *automated failure report* (AFR) and the *manual judgement report* (MJR).

An automated failure report message is automatically generated and marked as "polluted" by the client program for every failed download trial (⑧, ⑦) without any user intervention. The report is directly sent to the mapping keyword key owner if no location information is returned for the content key attempted (⑧) or to the mapping content key owner if a download trial with given location information has failed (⑦). This failure could be caused by either a bogus index record not being properly filtered out by the matching index nodes or by a stale index record.

If a download attempt is successful, and upon viewing or listening to the file's contents, the user may send a manual judgement report (⑩, ⑨) by simply marking it as either "clean" or "polluted" at his or her own discretion. This report is sent directly to both index nodes, keyword key owner (⑩) and content key owner (⑨), so that the results are reflected to the voting credits for the mapping index records.

3) Voting Credit Update Approaches: With regard to the voting credit updating, 3 different approaches are envisioned: Additive Increase Additive Decrease (AIAD), Additive Increase Multiplicative Decrease (AIMD), and Multiplicative Increase Multiplicative Decrease (MIMD). Note that, in all approaches, the initial amount of voting credits for an index record is 1, assuming that the original publisher of the file automatically casts a positive vote.

First, in the AIAD approach, index nodes add or subtract 1 point to or from the amount of voting credits for the index record (i.e. content key or location information), if a positive or negative vote is received for a index record. This approach would be effective when the probability that a downloader casts a vote is high and the cost of casting a vote is almost the same regardless of positive or negative. The approach, however, may not fit well with winnowing if the imbalanced feedback mechanism (section V-B) is considered. Second, in the AIMD approach, the amount of voting credits for an index record increases by one whenever a positive vote is received and conversely decreases by half for a negative vote. This approach is effective when more value needs to be given to a negative vote, which is the case in winnowing due to the imbalanced feedback. Third, in MIMD approach, the amount of voting credits for an index record doubles for a positive vote but decreases by half for a negative vote. This approach is appropriate when the user feedback rate is relatively low because it increases the amount of voting credits for the index records of clean files much faster than AIMD.

Throughout this paper, we mainly focus on the analysis

[4]Conversely, if there is no reply for the content searches with the content key within given period of time, the content key will be considered as bogus. Note that search queries succeeds 99.9% of the time in current eMule [21], which means the this assumption is rational.

[5]This random number (a cryptographic nonce) prevents attackers from using IP addresses on the outside of their local sub-networks.

of AIMD and MIMD approaches, considering that (1) more value should be given to a negative vote due to the imbalanced feedback of winnowing (the reason for the use of AIMD) and (2) it's difficult to expect extremely high user feedback in real P2P systems (the reason for the use of MIMD).

V. Security Analysis of Winnowing

The purpose of winnowing is to utilize the index structure of a P2P system to remove decoy index records; non-polluted files will then be easily located by downloaders through the remaining valid index records. The obvious response of polluters will be to preserve decoy index records as long as possible.

A. Reverse Voting Attack

A likely target of attack will be the user feedback mediation mechanism of winnowing. That is, malicious nodes will lie (i.e., vote positively for decoy index records and vote negatively for clean index records). This is referred to here as *reverse voting*. Such users may also attempt to amplify the impact by forging false identities (i.e., resorting to the Sybil attack [12]).

Winnowing addresses this problem by the *IP/24 prefix based binning strategy with weighted voting*. In this approach, one IP/24 address prefix [6] is mapped into a bin, and the *weight* of a vote in the same bin decreases as the number of votes in the bin increases. This prevents multiple votes in the same IP range from being weighted too much. The motivation for this approach is that it has been found that the majority of users in P2P systems are benign, and only a small faction of users in a restricted IP range are polluters. This was shown in previous work [6] and confirmed in our own measurement results VI-A. In addition, note that attackers can easily change or create their IDs (i.e. Sybil nodes) with little cost, but the use of multiple IP addresses outside of their local sub-nets is highly restricted due to the nonce, R, explained in IV-B1.

B. Index Node Insertion Attack

Alternatively, attackers can try to insert themselves as an index node of the target file so that they can easily falsify the index records as they wish. This will be possible if an attacker intentionally manipulates its Client ID so that the ID matches the hash of a keyword or a content key of the target file. Once the attacker becomes the index node(s) of the target title and returns only decoy index records whenever asked, a clean copy of the desired file cannot be located by benign requesters. Attackers may use the Sybil attack to enhance the effectiveness of the attack.

To deal with this problem, winnowing adopts a so called *imbalanced feedback* mechanism in which the size of voting messages differs based on whether the voting is positive or negative. That is, the size of a positive vote is very small (e.g. a few tens of bytes) whereas the size of a negative vote is relatively large (e.g. a few megabytes). This technique has two unique advantages. First, it penalizes malicious index nodes in that index nodes with decoy index records will suffer from a high volume of negative votes in terms of quantity. Second, the technique makes it more difficult for attackers to cast many reverse votes for clean index records since that will consume their own resources. Note that even though the size of negative vote is large, it will not burden compliant users much. This is because since winnowing converges fast, the number of negative votes which benign users need to cast will be very small in normal case. However, active polluters need to cast many negative votes to nullify user feedback mediation offered by winnowing, which demands exorbitant upload bandwidth.

[6]IP/24 prefix is used for the binning strategy here based on our observation, but any range of prefixes of IP addresses could be applicable depend on the circumstance.

C. Keyword Index Hijacking

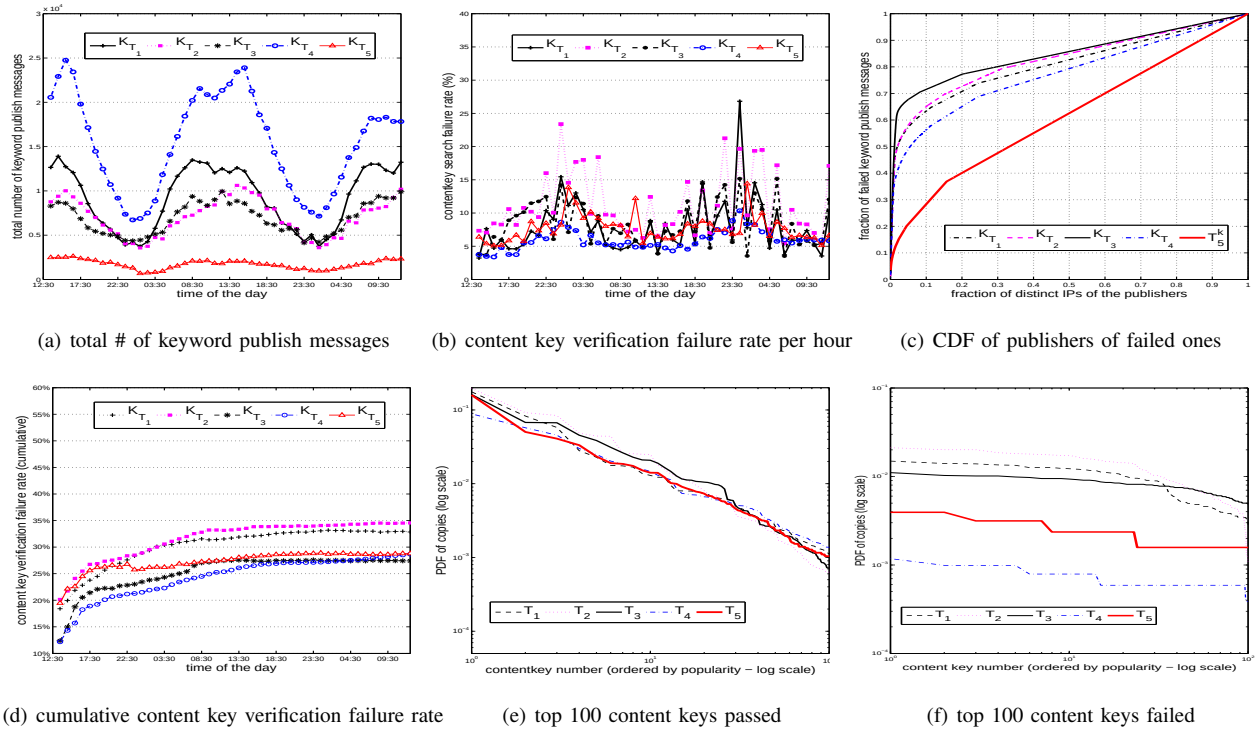
A final attack is simply to create malicious index nodes (i.e. keyword owners in this case) which own a portion of the keyword address space, and fail to provide any keyword index records for selected files. By doing so, the attacker prevents legitimate users from finding a content key for the title. Unfortunately, the imbalanced feedback mechanism of winnowing won't help against this kind of attack, since no search results are provided. We believe such attacks will be difficult and expensive, given the degree of redundancy of keyword index records in current systems. For instance, it has been found that there are an average of 19 keyword key owners for a single keyword in the KAD network [21], and each title is indexed by a number of keywords. In addition, Kad network adopt a so called α query (i.e. loose concurrent lookup) [14] where three searches for one search request are launched in parallel to avoid stale routing table entries, which could further reduce the effect of this type of attack.

VI. Measurement Results

To check the effectiveness of winnowing and for better understanding of the Kad network, partial functionalities of winnowing, the publish message verification (section IV-A), is implemented on top of the up-to-date eMule client (0.49a MorphXT version 11.0 [22]). Even though the winnowing client does not implement user feedback mediation (section IV-B), it suffices to understand the index pollution problem in the Kad network and how efficiently winnowing deals with the problem.

Five mp3 songs are investigated in the measurement. The first 4 famous songs ($T_1 \sim T_4$) were selected from the Top 10 Songs [23] in June of 2008. The last one (T_5) is selected for comparison from the late 1970's billboard charts, which hit #1 at that time but is relatively less popular nowadays.

To collect results, we inserted the winnowing clients into the Kad network. First, for the keyword key owners, one keyword per each title (K_{T_i}) is extracted from the file name and hashed into the 128-bit keyword key. Then, the client ID of the mapping keyword key owner is configured to have the same key value as the keyword key. By doing that, each winnowing client can receive keyword publish messages for the mapping keyword. Each keyword key owner verifies the



(a) total # of keyword publish messages (b) content key verification failure rate per hour (c) CDF of publishers of failed ones

(d) cumulative content key verification failure rate (e) top 100 content keys passed (f) top 100 content keys failed

Fig. 4. Keyword publish statistics in the Kad network

publisher whenever it receives a keyword publish message by sending a KADEMLIA_HELLO_REQ message (i.e. PUBLISHER_VERI_REQ). Content key verification follows for the keyword publish message which has passed the publisher verification by sending KEDEMLIA_SEARCH_REQ (i.e. CONTENT_KEY_VERI_REQ) messages. If any reply is returned within 45 seconds, the content key is treated as successful and will not be tested further in following keyword publications. The failed content keys, however, are continually being verified. Only the content keys which have never passed the content key verification throughout the measurement period are marked as bogus. Second, for each content key owner, the topmost popular content key (i.e. version) amongst all content keys of each title is selected. The content key must pass the two verifications by the matching keyword key owner. The topmost popular content key is then configured as its client ID. Content key owners carry out the publisher verification and location verification in a similar way. Data is collected for 48 hours for keyword publish messages and 10 hours for content publish messages, thus reflecting a twice long time period of the normal republish cycle.

A. *Keyword Publish Statistics*

Figure 4(a) shows the total number of keyword publish messages received by each keyword key owner. It clearly demonstrates the time of day effect on the total number of keyword publish messages. It also describes load inequalities in keyword key owners by an order of magnitude.

Figure 4(b) presents the content key verification failure rates per hour with respect to the total number of keyword publish messages which have passed the publisher verification. The average failure rate was 1.60% for the publisher verification (graph is not shown) and 7.69% for the content key verification respectively. Interestingly, similar to 4(a), the content key verification failure rate also shows the time of day effect but exactly in reverse. This indicates that some fixed amount of bogus keyword publish messages continue to be published by polluters. Thus, when there exists a relatively high number of legitimate keyword publishes, the rate is low, but as the number of legitimate keyword publishes decrease, the rate increases.

Figure 4(c) plots the cumulative distribution function (CDF) for the fraction of keyword publish messages failed with respect to the fraction of distinct IPs of the publishers. In this figure, IPs are reordered according to the number of keyword publish messages they advertised (the most, the first). As indicated, more than 60% of failed keyword publish messages are published from less than 20% of IPs for the 4 famous mp3 songs. This clearly shows that there exists a small fraction of users who cast massive bogus keyword publish messages for the famous songs. In fact, we identified two IP/24 ranges in which most polluters reside, which appear in all the 4 mp3 songs. It is, however, not true for the last less popular song.

The pollution so far is described in terms of the total number of keyword publish messages received by each keyword key owner. It is noted, however, that there exist multiple keyword publish messages for the same content key. In addition, each keyword key owner receives the keyword publish messages not only for the target keyword, but also for the keywords whose hash values are numerically close enough to its client ID. Since a user first sends a keyword search message to find a title, and the results include only the content keys whose file

name includes the keyword, we need to consider the index pollution level in terms of distinct content keys. Figure 4(d) demonstrates the cumulative content key verification failure rates with regard to the total number of distinct content keys whose file name truly includes the target keyword. This falls in the range from 27% to 35%. Meaning that, if a user sends a keyword search message with the keyword "K_{T_2}" to find the title of "T_2", up to 35% of content keys amongst all the content keys returned are bogus, therefore the user cannot find any location information with the content keys.

Finally, version popularity of each target song is investigated. In the Kad network, the number of keyword publish messages for a content key represents the popularity of the version (i.e. content key) because there must be a keyword publish message whenever a version is downloaded. To check the popularity, content keys are ordered based on the number of keyword publish messages received during the measurement period. Figure 4(e) demonstrates the PDF on a log-log scale for the number of copies with respect to the top 100 content keys which have passed the content key verification. The near linearity of the curves confirms that the version popularity of a title in the Kad network follows a Zipf distribution, which additionally confirms the results of previous studies [1, 9] on other P2P systems. Similar examination was made for the content keys failed in the content key verification. Figure 4(f) shows the PDF on a log-log scale for the number of copies with respect to the top 100 content keys failed. The results shows that most bogus content keys are published only once.

B. Content Publish Statistics

Similar measurements are conducted for content publish messages, but only the results for the statistics of users who have downloaded the same version in the same IP/24 ranges are shown here because the results are most relevant to the analytic models in section VII.

TABLE I
THE NUMBER OF USERS PER IP/24

users / IP24	T1	T2	T3	T4	T5
1	21,362	6,827	4,292	14,110	708
2	2,435	190	101	1,075	4
3	397	20	2	117	0
4	66	0	1	16	0
5	11	0	0	4	0
6	4	1	0	1	0
7	1	0	0	2	0
$\geq$8	5	0	0	1	0

Table I shows the number of users in each IP/24 for the topmost popular content key of each title. Interestingly, the average number of users per IP/24 range who have downloaded the same version was only 1.1. Note that the data was collected for 10 hours, which is twice longer than the content republish cycle, meaning that the actual number might be lower.

VII. MODEL BASED ANALYSIS AND RESULTS

In this section, we develop an analytical model for winnowing to study the relative performance of the approach, focusing on user feedback mediation (section IV-B). The model extends previous analytical models [3–5] and captures the dynamics of the proliferation of the good and bad copies of a single title in the system. Here, the model is briefly described and compared with previous models (the random selection model and the popularity based selection model in [3–5]). Notation used in this analysis is shown in table II.

TABLE II
SUMMARY OF NOTATIONS IN THIS SECTION

M	total # of fresh users joining the network
λ_t	user arrival rate at time t ($M\lambda_t$ users join the system at time t)
G_t	# of good copies currently shared in the network at time t
N_v^G	# of good versions (assume $N_v^G = G_0$)
B_t	# of bad copies currently shared in the network at time t
N_v^B	# of bad versions (assume $N_v^B = B_0$)
Δ_t^G	# of users downloading a good version at time t
Δ_t^B	# of users downloading a bad version at time t
R_t	# of retrials at time t
V_t^G	total amount of voting credits of good versions at time t
V_t^B	total amount of voting credits of bad versions at time t
p_a	probability that a user recognizes the pollution
p_s	probability that a user shares a file after download
p_v	probability that a user casts a vote after download
α	weighting factor([0.1])
L	maximum slackness (time from download to authenticity check)

A. Analytic Models

Initially, there exist G_0 different good versions fed by benign users and B_0 different bad versions introduced by polluters for the target title in the system. Let M be the total number of non-malicious, fresh users joining the system to download a version of the target title (users with intent to download only desired titles). The users arrive the system with the rate of λ_t at time t. When a user queries for the title, the matching keyword key owner returns to the user a list of versions (content keys) available in the system. Then, the user selects one from the list at its own discretion. We will make the following assumptions about user behavior.

- The initial benign users and polluter(s) who have introduced the incipient versions never leave the system and the versions keep being copied by other users.
- Once a user has downloaded a version, the authenticity of the version is checked within j hours, which is an independent and identically distributed (i.i.d.) random variable with an upper bound L.
- If the version is authentic, the user decides whether to share it with probability p_s. If polluted, then the user immediately deletes it and tries another download by issuing a new query until a valid copy is obtained.
- When checking the authenticity of a version, users might fail to detect the pollution of a bad version (false negative) with the probability $1 - p_a$. For the authentic ones, however, no false positive is assumed.
- For attack models, there exists only one intentional polluter for the target title, but the polluter could operate multiple machines (polluting peers) with different IP addresses. Without loss of generality, the initial bad versions (B_0) are introduced by this polluter.

In addition to the above assumptions, for simplicity, we only focus on how efficiently users can find a good version in the system rather than how fast they can download it. So, we assume that it equally takes one time slot (one hour) for all users to download a copy (including the query request and response time). In this sense, for winnowing, only the user feedback mediation (section IV-B) done by (matching) keyword key owners will be considered. Initially, no attacks except the initial bad copies are considered to develop each model, but will be discussed further later in section VII-B.

1) Random Selection Model: When a user sends a keyword search message, the matching keyword key owner returns a list of possible versions of the title currently available in the system. In the random selection model, it is assumed that the user randomly selects one from the list. So, the initial probability that the user selects a good version is $\frac{N_v^G}{N_v^G+N_v^B}$. As a consequence, at time t, the number of users downloading a good version is

$$\Delta_t^G = M\lambda_t \frac{N_v^G}{N_v^G + N_v^B} \tag{1}$$

and the number of users downloading a bad version is

$$\Delta_t^B = M\lambda_t \frac{N_v^B}{N_v^G + N_v^B}. \tag{2}$$

Since the downloaded copies are immediately shared in most file sharing P2P systems including the Kad network, the number of good copies shared at time $t+1$ would be $G_t+\Delta_t^G$. Some of the users, however, might decide either not to share their downloaded good copies or leave the system (with probability $1-p_s$). Assume that such decision happens within j time slots with equal probability $1/L$. Then, the number of good copies shared in the network at time $t+1$ is

$$G_{t+1} = G_t + \Delta_t^G - (1-p_s)\sum_{j=1}^{L} \frac{\Delta_{t+1-j}^G}{L}. \tag{3}$$

With regard to bad copies, a downloaded bad copy will be deleted if the user detects the pollution (p_a) or decides not to share it ($1-p_s$) after pollution detection failure. Thus, the number of bad copies shared in the network at time $t+1$ is

$$B_{t+1} = B_t + \Delta_t^B - (p_a + (1-p_a)(1-p_s))\sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L}. \tag{4}$$

If a user detects that the downloaded copy is polluted (with probability p_a), the user immediately deletes it and issues a new search query. So, the number of retrial at time $t+1$ will be

$$R_{t+1} = p_a \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L} \tag{5}$$

If the retrials are considered, the equation 1 and 2 need to be updated as seen below.

$$\Delta_t^G = (M\lambda_t + R_t)\frac{N_v^G}{N_v^G + N_v^B} \tag{6}$$

$$\Delta_t^B = (M\lambda_t + R_t)\frac{N_v^B}{N_v^G + N_v^B}. \tag{7}$$

Note that under the random selection model, the probability that a user selects a good version remains constant over time. So does the probability that a user selects a bad version.

2) Popularity Based Selection Model: If users base their download selection purely on versions, the download experiences of former users are useless for new arrival users. If the experiences are shared, however, the probability that a new arriving user selects a good version could increase. In fact, in most file sharing P2P, when an inquiry is made, index nodes return a list of versions with information regarding how many copies of each version are available in the network.

In the popularity based selection model, it is assumed that a user selects one version from the list based on its popularity. That is, the probability that a version is selected is proportional to the number of its copies currently shared in the network. So, the probability that a user selects a good version at time t is $\frac{G_t}{G_t+B_t}$. Thus, at time t, the number of users downloading a good version is

$$\Delta_t^G = (M\lambda_t + R_t)\frac{G_t}{G_t + B_t} \tag{8}$$

and the number of users downloading a bad version is

$$\Delta_t^B = (M\lambda_t + R_t)\frac{B_t}{G_t + B_t}. \tag{9}$$

The fact that the popularity of versions of a title follows a Zipf distribution in the Kad network (section VI-A) strongly indicates that Kad users act as those in this model.

3) Voting Credit Based Selection Model (winnowing): In winnowing, each keyword key owner collects user feedback for each version (content key) via voting from users who have downloaded the version. Upon request, the index nodes return a list of versions with the voting credits for each version to the requesting user. Initially each version gets 1 point when published regardless of its authenticity assuming that the original publisher of the version automatically casts a positive vote for the version. Under this model, it is assumed that users select a version from the list based on the voting credits of the version. That is, the probability that a version is selected is proportional to the amount of its voting credits. As a consequence, at time t, the number of users downloading a good version is

$$\Delta_t^G = (M\lambda_t + R_t)\frac{V_t^G}{V_t^G + V_t^B} \tag{10}$$

and the number of users downloading a bad version is

$$\Delta_t^B = (M\lambda_t + R_t)\frac{V_t^B}{V_t^G + V_t^B}. \tag{11}$$

With regard to the voting credit update approaches, AIMD and MIMD are analyzed and evaluated here due to the reasons discussed in section IV-B3.

- AIMD approach: The amount of voting credits of a version increases by one whenever a positive vote is received and conversely decreases by half for a negative vote. Without loss of generality, a user can cast its vote after checking the authenticity of the downloaded version. If the probability that a user casts its vote after download is p_v, then the total amount of voting credits for all good versions at time $t+1$ will be

$$V_{t+1}^G = V_t^G + p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^G}{L} \tag{12}$$

Next, how much will voting credits be reduced if there is one negative vote? Since there are N_v^B bad versions and the total amount of voting credits for all bad versions at time t is V_t^B, one negative vote reduces the credits by $\frac{V_t^B}{N_v^B} * \frac{1}{2}$ on average. So, the amount of voting credits after reflecting the first negative vote is $V_t^B - \frac{V_t^B}{2N_v^B} = V_t^B \frac{2N_v^B-1}{2N_v^B}$. If one more negative vote is reflected, then the amount of voting credits will be $V_t^B \frac{2N_v^B-1}{2N_v^B} - V_t^B \frac{2N_v^B-1}{(2N_v^B)^2} = V_t^B \frac{(2N_v^B-1)^2}{(2N_v^B)^2}$. In this way, the total amount of final voting credits for all bad versions after n negative votes will be $V_t^B \frac{(2N_v^B-1)^n}{(2N_v^B)^n} = V_t^B(1 - \frac{1}{2N_v^B})^n$. Since there are

$$n_t^{EN} = p_a p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L}$$

possible effective negative votes (n_t^{EN}) at time t and a portion $(1-p_a)$ of users who fail to detect the pollution will cast a positive vote with the probability p_v. Thus, we have

$$V_{t+1}^B = V_t^B(1 - \frac{1}{2N_v^B})^{n_t^{EN}} + (1-p_a)p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L}. \tag{13}$$

- MIMD approach : The amount of voting credits for a version doubles for a positive vote but decreases by half for a negative vote. We can simply calculate the amount of voting credits that are increased by one positive vote. Since there are N_v^G good versions and the total amount of voting credits for all good versions at time t is V_t^G, one positive vote increases the credits by $\frac{V_t^G}{N_v^G}$ on average. So, the total amount of voting credits for all good versions after reflecting the first positive vote is $V_t^G + \frac{V_t^G}{N_v^G} = V_t^G \frac{N_v^B+1}{N_v^G}$. After similar iteration, the final voting credits with n positive votes will be $V_t^G(1+\frac{1}{N_v^G})^n$. Since there are

$$n_t^{EP} = p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^G}{L}$$

possible effective positive votes (n_t^{EP}) at time t, we get

$$V_{t+1}^G = V_t^G(1 + \frac{1}{N_v^G})^{n_t^{EP}}. \tag{14}$$

By the way, with bad versions, one positive vote and one negative vote offset each other with regard to V_{t+1}^B, and the number of possible effective negative votes is

$$\begin{aligned} n_t^{EN} &= p_a p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L} - (1-p_a)p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L} \\ &= (2p_a - 1)p_v \sum_{j=1}^{L} \frac{\Delta_{t+1-j}^B}{L}. \end{aligned}$$

In this case, if $n_t^{EN} \geq 0$, then V_{t+1}^B decreases but increases otherwise. So, we have

$$V_{t+1}^B = \begin{cases} V_t^B(1 - \frac{1}{2N_v^B})^{n_t^{EN}} & \text{when } n_t^{EN} \geq 0 \\ V_t^B(1 + \frac{1}{N_v^B})^{n_t^{EN}} & \text{otherwise.} \end{cases} \tag{15}$$

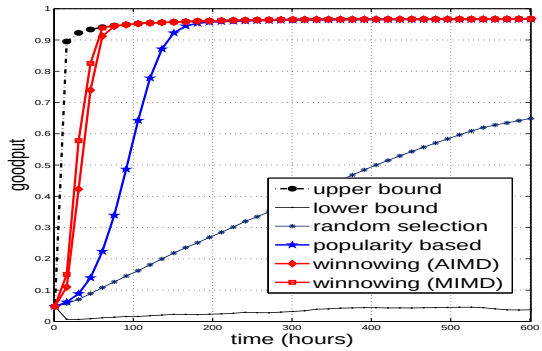
Using these results, the effectiveness of user-mediated feedback is now examined for a specific scenario.

B. Analytical Results

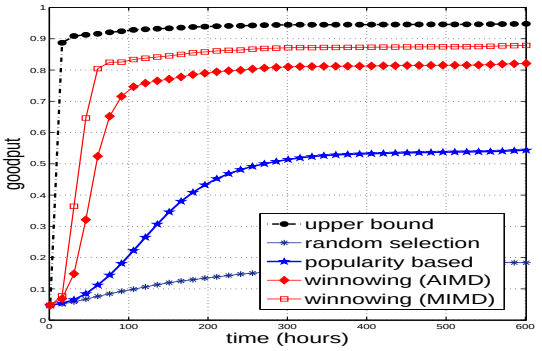
1) The scenario: For numerical comparison, values of key parameters were set as follows: $M = 20,000$, $G_0 = 25$, $B_0 = 500$, and $L = 48$. These values were adapted from previous studies [3–5]. With regard to the user arrival rate (λ_t), RedHat 9 Torrent tracker trace [24] is used to reflect the realistic user interest and arrival rate. The metric for comparison is $goodput_t$, which is defined as the ratio of the number of good copies (G_t) to the total number of copies ($G_t + B_t$) shared in the system at time t (i.e. $goodput_t = \frac{G_t}{G_t+B_t}$). The *final goodput* refers to $goodput_t$ where $t = 600$ [5]. Unless otherwise specified, these basic settings were used to generate all results.

2) Without other attacks except B_0: Figure 5(a) shows goodputs of the different models, as a function of time, under perfect conditions ($p_a = 1,\ p_s = 1,\ p_v = 1$). The upper bound that is shown would be achieved if each user always selected a good (non-polluted) file version; the lower bound is if each user always selected a bad (polluted) file version. This is possible if there exists an omnipotent big brother in the system who lets users know what versions are authentic whenever asked, which is the upper bound of any reputation system. The lower bound is the reverse case, so that the probability that a user selects a good copy is always 0, which is the ultimate objective of polluters. As seen in this graph, if no pollution attack except B_0 is performed, winnowing with either AIMD or MIMD reaches optimal performance much faster than either the popularity based approach or the random selection approach. The random selection approach does not reach the optimal performance level even under perfect conditions.

Perfect conditions are obviously unrealistic in real systems. Therefore, the relative performance of each approach is investigated under more realistic conditions ($p_a = 0.8,\ p_s = 0.6,\ p_v = 0.6$). In this case, p_a and p_s are adapted from the previous work [3–5] whereas p_s is set under the assumption that users who decide to share the



(a) Goodputs under perfect conditions ($p_a = 1, p_s = 1, p_v = 1$)



(b) Goodputs under realistic conditions ($p_a = 0.8, p_s = 0.6, p_v = 0.6$)

Fig. 5. Results without attacks except the initial pollution (B_0)

downloaded file will willingly cast their votes (i.e. $p_s = p_v$). Figure 5(b) demonstrates goodputs of the different models, as a function of time, under the realistic conditions. With 60% user feedback, winnowing with AIMD or MIMD outperforms the other two approaches by an even larger margin. In addition, since MIMD approach increases the probability that a good version is selected more rapidly than MIMD approach under low user feedback, MIMD approach slightly beats AIMD approach if no other pollution attack is considered. In fact, winnowing beats the other two with as low as 20% user feedback (graphs are not shown).

3) Effect of the reverse voting attack: No other pollution attack except the initial bad versions (B_0) has been considered thus far. A polluter, however, could contribute some other actions such as the reverse voting attack explained in section V-A to decrease the goodput in the system. Let *pollution rate* be the ratio of the number of reverse votes casted by the polluter over the total number of votes casted in the system over time in winnowing [7]. The attacker may prefer to cast positive votes for bad versions rather than negative votes for good versions due to the inequality of size in voting messages. For the purposes of investigation, therefore, the ratio of the number of negative votes to the number of positive votes is set to $1/2$.

[7]Similarly, in the popularity based approach, pollution rate is defined as the number of fake keyword publish messages cast by the polluter to increase the popularity of polluted versions over the total number of keyword publish messages generated in the system over time.

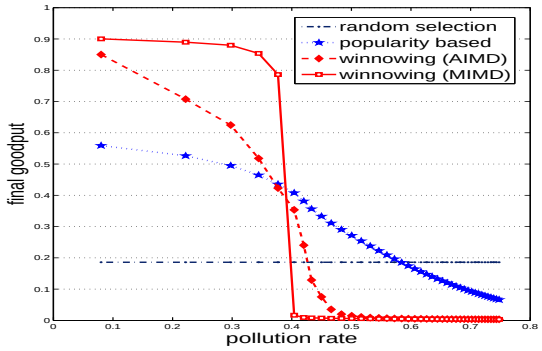


Fig. 6. The effects of the reverse voting attack in winnowing and the fake keyword publish attack in the popularity based approach.

Figure 6 describes the effectiveness of the reverse voting attack done by the polluter. As seen in the graph, if not properly addressed, one polluter could sharply decrease the final goodputs of all approaches with the exception of the random selection approach. This result substantiates the previous study [4], showing that the random selection approach outperforms most imperfect reputation systems under severe (around 60% in this case) pollution attacks. In fact, the index pollution attack is more specific to winnowing than the popularity based approach. This is because, if the number of reverse votes is greater than that from normal compliant users, the voting credits will contribute negatively to the goodputs in winnowing.

4) Effect of the IP binning strategy: To remedy the problem, the IP/24 prefix based binning strategy with weighted voting explained in section V-A is applied. Let's consider the impact of a reverse vote on V^G_{t+1} and V^B_{t+1} with the weighting factor (α) at time t in winnowing. If one reverse vote is received for a good version at time t,

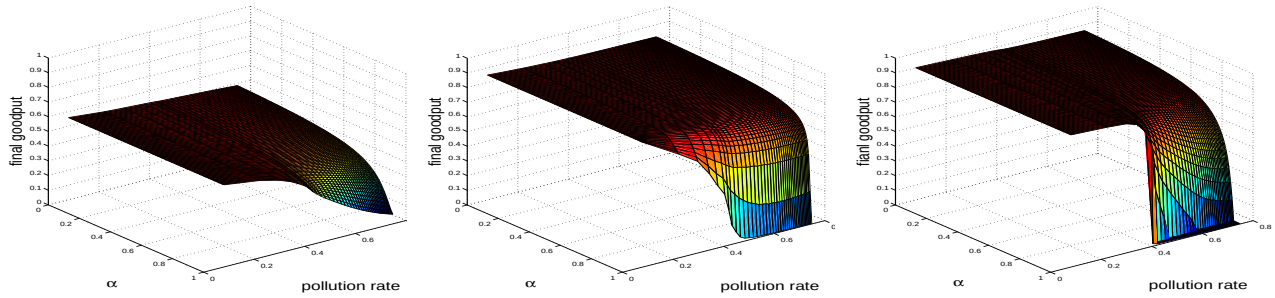
$$V^G_{t+1} = V^G_t(1 - \frac{\alpha^k}{2N^G_v})$$

and, one reverse vote is received for a bad version,

$$V^B_{t+1} = \begin{cases} V^B_t + \alpha^k & \text{AIMD} \\ V^B_t(1 + \frac{\alpha^k}{N^B_v}) & \text{MIMD} \end{cases}$$

where k is the order of the vote in the IP/24 bin to which it belongs.

Figure 7(a), (b), and (c) show the final goodputs as a function of pollution rate and the weighting factor α. For fair comparison, this strategy is also applied to the popularity based approach. As seen in the graphs, all approaches provide considerable immunity to the attack in terms of the final goodput with a low α value. Winnowing with AIMD and MIMD approaches, however, outperform the popularity based approach. Since measurements in the KAD system have

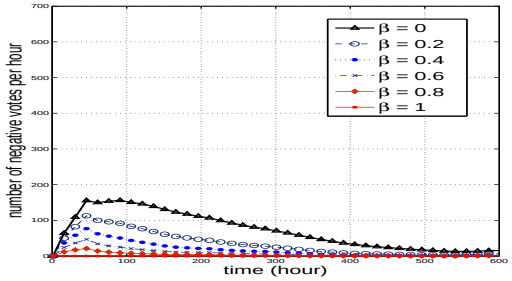


(a) Popularity based approach (b) winnowing with AIMD (c) winnowing with MIMD

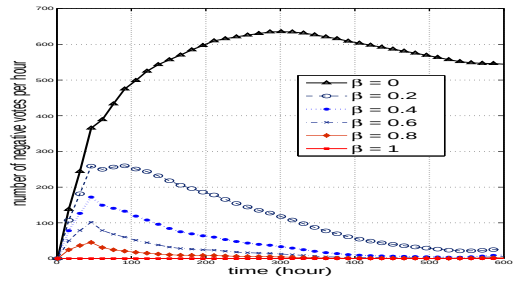
Fig. 7. Effect of weighting factor (α) under the reverse voting attack with the IP binning strategy with ($p_a = 0.8, p_s = 0.6, p_v = 0.6$)

shown that there are only 1.1 users in the same IP/24 address range who have downloaded the same file version (see section VI-B), winnowing (even with a low α value) correctly reflects overall user feedback. In addition, sybil attacks in the same IP/24 range will be highly restricted under this strategy.

5) Under the index node insertion attack: The results above show that winnowing is robust to the reverse voting attack. Consider instead that the attacker takes advantage of the index node insertion attack (section V-B). Most DHT-based P2P systems are susceptible to this attack because it is difficult to detect, and no penalty can be given to the malicious index node under current implementation. In winnowing, however, the imbalanced feedback mechanism can penalize the malicious index node with negative votes.



(a) $p_a = 0.8, p_v = 0.6$



(b) $p_a = 1, p_v = 1$

Fig. 8. # of negative votes received by the attacker under different β

Let β be the ratio of good versions returned by the polluter. Since the attacker can easily manipulate the voting credits for each version, it can regulate β so that only a portion of good versions can be downloaded by users. Figure 8 demonstrates the number of negative votes received by the malicious index node (polluter) under different β as a function of time. As seen in the graph, if the polluter tries to decreases β, it will be confronted with the higher number of negative votes. Clearly, the higher the user awareness (p_a) and user feedback (p_v), the more penalties are imposed on the attacker under the same β. Thus, by properly tuning the size of the negative vote, compliant users could exhaust the resources of the malicious node, which, to the best of our knowledge, is a unique contribution of this approach. Moreover, the number of negative votes will further increase if the attacker tries to pollute multiple titles or want to run many Sybil index nodes. As indicated, if index nodes keep only clean index records (i.e. $\beta = 1$), the overhead incurred by the imbalanced voting mechanism will be negligible, which is a strong incentive for index nodes to voluntarily remove decoy (i.e. bogus or corrupted) index records.

VIII. Discussion

In this section, we analyze the overhead for implementing winnowing on the eMule system. We also discuss the limitations of our approach and possible solutions (to those limitations).

A. Message Overhead

As seen in the figure 3, the message overhead of winnowing includes the messages for the publisher verification, the message content verification, and the user feedback reports. These messages are newly added in winnowing along with the basic messages of eMule. Among these, the message overheads for publisher verification, the location verification, and the positive votes are quite negligible in that they are small and used only for a single round-trip communication between two peers.

The messages for the content key verification and negative votes, however, could result in heavier overhead. First, the content key verification accompanies a relatively high number of subordinate messages to find the matching content key owners through the iterative routing. Based on our measurements, one content key verification yields 21 KADEMLIA_REQ messages on average. This overhead could be well absorbed if the

messages are used for the routing table maintenance. In fact, each peer in the Kad network periodically checks every contact in its routing table through the KADEMLIA_HELLO_REQ message in order to detect and delete stale routing entries, which is normally done once every two hours [20, 22]. If a KADEMLIA_REQ message is sent to one of its contacts for the content key verification, then the peer does not need to send a KADEMLIA_HELLO_REQ message again to check the liveness of the contact. Therefore, the overhead for the content key verification could compensated by the reduced overhead for the routing table maintenance. Second, since the size of a negative vote is larger than that of a positive vote, sending a negative vote could burden the voter and the system. Since the number of negative votes sent by non-malicious users is proportional to the number of decoy index records in the system, the overhead caused by negative votes on benign users decreases as decoy index records are removed.

B. Space Complexity

The user feedback mediation of winnowing (section IV-B) depends on the use of lists to maintain user information, which includes KRL, CKVL, CRL, and LVL. If the number of users in a list is too high, these lists increase the space complexity of the system. The KRL may require the most space, since every downloader needs to send keyword search messages to download a file.

To reduce such overhead, we propose the use of Bloom filters [25, 26]. Bloom filters have a strong space advantage over other data structures for representing sets, while incurring the risk of false positives. The risks, however, are manageable by tuning the number of bits of the hash function and the number of hash functions used over the possible number of users in the set. Under reasonable assumptions, a Bloom filter with an estimated 1% false positive rate will require 89% less storage than that of a complete list.

C. Incentive Mechanisms

The success of winnowing depends on the voluntary participation of index nodes and downloaders; index nodes need to remove decoy index records and downloaders must, to some degree, share their download experiences with the index nodes.

In winnowing, the imbalanced feedback provides sufficient incentives for index nodes to purge decoy index records from their index records. As seen in figure 8, the more decoy index records the index node holds, the greater the volume of negative voting messages it will face in winnowing. Thus, it is in the best interest of index nodes to aggressively remove decoy index records they hold.

Even though their voting efforts promote their own public good, we have not addressed the issue of additional incentives for downloaders to provide feedback. Several promotion methods can be applied. First, automation techniques such as the automated failure report (section IV-B2) are useful because they can lower the efforts of participants. Second, notifications could be used. Pop-up windows could be created whenever a file download is completed, the downloader deletes the downloaded file, or the file is first viewed or accessed. This will remind the downloader to cast their vote.

IX. Conclusion

This paper proposes a novel P2P anti-pollution scheme called *winnowing*. In winnowing, index nodes make every effort to maintain their index records as clean as possible, through publish message verification and user feedback mediation.

Publish message verification of winnowing have been implemented on top of the up-to-date eMule client. The winnowing clients (i.e. index nodes) efficiently block bogus publish messages with these modifications. Measurements in the Kad network show: (1) up to 35% of index records of keyword key owners are bogus where no publisher information is located from the index records; (2) a significant amount of bogus keyword publish messages continue to be issued by polluters; (3) keyword publish (both clean and bogus) messages arrive in a pattern strongly influenced by the time of day; (4) version popularity of a title in the Kad network follows a Zipf distribution; and, (5) the average number of distinct users per IP/24 address range who have downloaded the same version of a title is only 1.1.

An analytical model of winnowing has been developed. The model demonstrates the effectiveness of user feedback mediation. The major findings are: (1) the winnowing approach converges much faster than random selection and popularity based selection under a reasonably assumed user feedback (e.g. 60%) level; (2) most reputation systems are weak to the reverse voting attack, if not well designed; (3) With the IP/24 prefix based binning strategy with weighted voting, winnowing provides considerable immunity to the reverse voting attack; and, (4) winnowing will impose a heavy penalty on attackers even under the index node insertion attacks, with the use of a novel technique called imbalanced feedback.

References

[1] J. Liang, R. Kumar, Y. Xi, and K. W. Ross, "Pollution in p2p file sharing systems," in *IEEE INFOCOM'05*, Miami, FL, March 2005.

[2] N. Christin, A. S. Weigend, and J. Chuang, "Content availability, pollution and poisoning in file sharing peer-to-peer networks," in *ACM EC'05*.

[3] U. Lee, M. Choi, J. Cho, M. Y. Sanadidi, and M. Gerla, "Understanding pollution dynamics in p2p file sharing," in *IPTPS'06*, Santa Babara, USA, February 2006.

[4] 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'05*, Banff, Alberta, Canada, 2005, pp. 38 – 49.

[5] R. Kumar, D. D. Yao, A. Bagchi, K. W. Ross, and D. Rubenstein, "Fluid modeling of pollution proliferation in p2p networks," *ACM SIGMETRICS Performance Evaluation Review*, pp. 335 – 346, 2006.

[6] J. Liang, N. Naoumov, and K. W. Ross, "Efficient blacklisting and pollution-level estimation in p2p file-sharing systems," in *The ASIAN INTERNET ENGINEERING CONFERENCE*, 2005, pp. 1–21.

[7] P. Dhungel, X. Hei, K. W. Ross, and N. Saxena, "The pollution attack in p2p live video streaming: measurement results and defenses," in *the 2007 workshop on Peer-to-peer streaming and IP-TV*, Japan, 2007.

[8] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *WWW'03*, Budapest, Hungary, 2003, pp. 640 – 651.

[9] C. Costa, V. Soares, J. Almeida, and V. Almeida, "Fighting pollution dissemination in peer-to-peer networks," in *ACM SAC'07*, Seoul, Korea, 2007, pp. 1586 – 1590.

[10] K. Walsh and E. G. Sirer, "Experience with an object reputation system for peer-to-peer filesharing," in *USENIX NSDI'06*, San Jose, CA, May 2006.
[11] M. Iguchi, M. Terada, and K. Fujimura, "Managing resource and servent reputation in p2p networks," in *the 37th Hawaii International Conference on System Sciences*, 2004.
[12] J. R. Douceur, "The sybil attack," in *IPTPS'02*, Cambridge, MA, March 2002.
[13] C. Costa and J. Almeida, "Reputation systems for fighting pollution in peer-to-peer file sharing systems," in *IEEE P2P'07*, 2007.
[14] R. Brunner, "A performance evaluation of the kad-protocol," Master's thesis, University of Mannheim, Sophia-Antipolis, France, 2006.
[15] M. Steiner, T. En-Najjary, and E. W. Biersack, "Exploiting kad: Possible uses and misuses," *ACM SIGCOMM Computer Communication Review*, vol. 37, pp. 65 – 70, 2007.
[16] MD4. [Online]. Available: http://en.wikipedia.org/wiki/MD4
[17] J. Liang, N. Naoumov, and K. W. Ross, "The index poisoning attack in p2p file sharing systems," in *IEEE INFOCOM'06*, April 2006.
[18] Q. Feng and Y. Dai, "Lip: A lifetime and popularity based ranking approach to filter out fake files in p2p file sharing systems," in *IPTPS'07*, February 2007.
[19] 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'06*, Dijon, France, 2006, pp. 1875 – 1879.
[20] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. F. Kune, N. Hopper, and Y. Kim, "Attacking the kad network," in *SecureComm'08*, Istanbul, Turkey, September 2008.
[21] D. Stutzbach and R. Rejaie, "Improving lookup performance over a widely-deployed dht," in *IEEE INFOCOM'06*, April 2006.
[22] "emule morph," May 2008. [Online]. Available: http://emulemorph.sourceforge.net/
[23] "Top 10 songs," June 2008. [Online]. Available: http://top10songs.com/
[24] "Redhat 9 torrent tracker trace." [Online]. Available: http://mikel.tlm.unavarra.es/~mikel/bt_pam2004/
[25] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, pp. 422 – 426, 1970. [Online]. Available: http://portal.acm.org/citation.cfm?id=362692
[26] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," in *the 40th Annual Allerton Conference on Communication, Control, and Computing*, 2002, pp. 636–646.