# The Index Poisoning Attack in P2P File Sharing Systems

Jian Liang
Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201
Email: jliang@cis.poly.edu

Naoum Naoumov
Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201
Email: naoum@cis.poly.edu

Keith W. Ross
Department of Computer and Information Science
Polytechnic University
Brooklyn, NY 11201
Email: ross@poly.edu
Phone: 1-718-260-3859

***Abstract*— P2P file-sharing systems have indexes, which users search to find locations of desired titles. In the *index poisoning attack*, the attacker inserts massive numbers of bogus records into the index for a set of targeted titles. As a result, when a user searches for a targeted title, the index returns bogus results, such as bogus file identifiers, bogus IP addresses, or bogus port numbers. In this paper we first show that both structured and unstructured P2P file-sharing systems are highly vulnerable to the index poisoning attack. We then develop a novel and efficient methodology for estimating index poisoning levels and pollution levels in file-sharing systems. The methodology is efficient in that involves neither the downloading nor the analysis of binary content files. We deploy data-harvesting platforms for FastTrack, an unstructured file-sharing system, and Overnet, a DHT-based file-sharing system. Applying our methodology to harvested data, we find that index poisoning is pervasive in both systems. We also outline a distributed blacklisting procedure for countering the index poisoning and pollution attacks.**

## I. Introduction

By many measures, P2P file sharing continues to be one of the most important applications in the Internet today. There are currently more than 8 million users simultaneously connected to either FastTrack/Kazaa, eDonkey2000 and eMule, with many additional users sharing files with BitTorrent. The content being shared includes MP3 songs, entire albums, television shows, entire movies, documents, images, software, and games. In 2004, in a tier-1 ISP, P2P file sharing accounted for more than 60% of traffic in the USA and more than 80% of the traffic in Asia [1].

Because of the potential of huge losses, the "copyright industry" (including the music, film, television, gaming, and book publishing industries) is aggressively attempting to curtail the unauthorized distribution of content in P2P file sharing systems. Along with judicial and legislative efforts, the copyright industry is sabotaging many of the more popular file sharing systems. An Infocom 2005 paper [2] exposed and investigated the **pollution attack** in the FastTrack file-sharing network (see also [15] [13]). In the pollution attack, the attacker corrupts the targeted content, rendering it unusable, and then makes this polluted content available for sharing in large volumes. Unable to distinguish polluted files from unpolluted files, unsuspecting users download the polluted files into their own file-sharing folders, from which other users may then later download the polluted files. In this manner, polluted files spread through the file-sharing system. In the Spring of 2004, pollution was highly prevalent in the FastTrack system, with as many as 50% to 80% of the copies of popular titles being polluted [2].

In the current paper we expose and investigate a different – and even more insidious – attack, which we refer to as the **index poisoning attack**. As we shall show in this paper, both structured (that is, DHT-based) and unstructured file-sharing systems are highly vulnerable to, and currently under attack from, index poisoning. To describe this attack, recall that most P2P file sharing systems have indexes, allowing users to discover locations (IP addresses and port numbers) of desired content. Depending on the file sharing system, the index may be centralized (as it was in Napster), distributed over a fraction of the file-sharing nodes (as in FastTrack), or distributed over all (or a large fraction) of the nodes (as in Overnet and other DHT-based systems).

The index poisoning attack is done by inserting massive numbers of bogus records into the index. Records can be poisoned in many ways, but one common method is to use randomly chosen file identifiers (that is, content hashes) which do not correspond to any existing files in the file-sharing system. When a user attempts to download a file with a randomly generated identifier, the file sharing system fails to locate associated actual file and displays in the GUI "more sources needed" or "looking". The file-sharing system typically continues to search for the non-existent file. In response to "more sources needed," the user may attempt to download the title with a different identifier (seemingly from a different location), which may again result in "more sources needed". If the attacker succeeds in massively poisoning the index for the title, the user may try tens of identifiers without locating a copy of the desired title. If the user doesn't locate a file in its attempts, it will typically abandon the search.

A drawback of the pollution attack is that it requires significant bandwidth and server resources to be successful. In particular, in the early stages of the attack, when the title is just released, the attacker needs to make available corrupted copies of the title from many sources, all with high-bandwidth connections (in order to entice the user to download from the

attacker); the attack servers must also respond to a flood of requests and perform expensive uploads. One of the "beauties" of the index poisoning attack is that it requires substantially less bandwidth and server resources. Specifically, with index poisoning, the attacker does not need to transfer files; instead it simply deposits large numbers of bogus text records in the index. Thousands of bogus records per title, all with different identifiers, can be deposited into the index from a single attack node. Currently, the copyright industry is deploying in concert the index poisoning and the pollution attacks.

The paper investigates the poisoning attack in two file sharing systems, Overnet and FastTrack. These two file systems have very different architectural designs. An integral component of the popular eDonkey2000 file sharing system, Overnet is a DHT-based file-sharing system. In Overnet, both keywords and files are hashed, and the resulting records are placed in nodes with ids that are "close" to the hashes. Thus the index is distributed over all of the Overnet nodes. Employed by Kazaa, Grokster and iMesh, FastTrack is a two-tier unstructured file-sharing system whose index is distributed over a relatively small fraction of the nodes, called supernodes.

The contributions of this paper are as follows:

- We show that P2P file-sharing systems are highly vulnerable to the index poisoning attack. We investigate the attack's intricacies for Overnet and FastTrack.
- We develop a novel and efficient methodology for estimating both index poisoning levels and pollution levels in generic file-sharing system. This methodology is efficient in that it involves neither the downloading nor the analysis of binary content files. The methodology instead gathers metadata from the file-sharing indexes. Once the metadata is gathered, we input the metadata into a simple off-line algorithm to estimate the poison and pollution levels.
- We develop and deploy measurement platforms that allow us to gather the requisite metadata in Overnet and FastTrack. To gather the metadata, we use a node-insertion approach for Overnettion approach for Overnet and a crawling approach for FastTrack. We show that index poisoning is currently pervasive in both Overnet and FastTrack.
- We examine methods for countering the index poisoning attack. We outline a promising distributed blacklisting scheme.
- For the index poisoning attack, we compare the required resources for structured and unstructured P2P file-sharing systems. Furthermore, we discuss a powerful distributed denial-of-service attack that can be created by index poisoning a DHT-based file sharing system. In particular, we have discovered that Overnet can be exploited for such an attack.

The contribution of the current paper is significantly different from that of the Infocom 2005 paper [2]. The Infocom 2005 paper measured only the FastTrack network and, in particular, only investigated the pollution attack in the FastTrack network. That measurement study took place in 2004, when index poisoning was not pervasive. Furthermore, the methodology for pollution estimation, presented in [2], was inefficient in that it required a large number of binary files to be downloaded and analyzed for each investigated title. Thus, the current contribution goes well beyond the Infocom 2005 paper in that it investigates index poisoning as well as pollution, for Overnet as well FastTrack, provides a novel and efficient methodology for estimating poisoning and pollution levels, and addresses blacklisting mechanisms to counter the index poisoning attack.

This paper is organized as follows. Section 2 introduces terminology and provides and overview of the index poisoning attack. Section 3 details the index poisoning attack in FastTrack and Overnet. Section 4 presents our methodology for accurately estimating poisoning and pollution levels. Section 5 presents our measurement results. Section 6 describes a distributed blacklisting scheme for countering the index poisoning and pollution attacks. In Section 7, we briefly cover a number or related issues, including a discussion of how index poisoning can make a DHT the source of a massive DDoS attack. We also cover related work in Section 7. We conclude in Section 8.

## II. Overview of Index Poisoning

### A. P2P Terminology

To explain the index poisoning attack, we first need to introduce some terminology. In this paper we investigate attacks against the distribution of songs and videos in P2P file-sharing systems. We shall refer to a specific song or video as a **title**. A given title can have many different **versions**. These versions primarily result from the presence of a large number of rippers/encoders, each of which can produce a slightly different version of the same title. Furthermore, many file types include metadata embedded in the file (such as the ID3 tags in MP3 files), so that additional file versions are created when this metadata is modified. For a popular title, a file-sharing system may contain thousands of different versions. Each version has an **identifier**, which is typically a hash of the version.

Users download different versions of titles from each other, thereby creating multiple **copies** of identical versions in the P2P file sharing system. At any given time, a P2P file-sharing system may make available thousands of copies of the same version of a particular title. Each participating node advertises to the distributed index information about the copies it is sharing. For a given file, this advertised information (which may span multiple messages) includes the version identifier (typically the hash of the file), the location of the file (IP address and port number), and keywords associated with the file.

When a user wants to query for a specific title, the user performs a keyword search, using keywords that relate to the title, such as artist name and song/movie title. The keywords are sent within a query to the distributed index. Depending on the file-sharing system, a subset of the index may be examined. The index responds if it has knowledge about copies

matching the keywords. The responses include metadata for the matching copy, the IP address and port number for where the version resides, and the version identifier. Depending on the P2P system that information might be available at once (e.g. unstructured systems store and retrieve metadata and location information together), or separately. (DHT-based systems store metadata and location information at different nodes and require multiple messages to obtain the two sets of information.)

A P2P node will typically have two port numbers. One port number is used for sending and receiving messages, such as search queries and replies. The other port number is used for uploading files. We refer to these two port numbers as the **messaging port number** and **service port number**, respectively.

### B. The Index Poisoning Attack

In a typical file-sharing system today, when an index receives an advertisement for a copy, the index does not authenticate the advertisement; specifically, it does not verify that the copy is truly available at the advertised location (an IP address and port number). In the index poisoning attack, the attacker falsely advertises copies of the targeted titles. These false advertisements can be:

- random version identifiers that do not correspond to any existing versions;
- IP addresses that do not correspond to any nodes participating in the file-sharing system;
- or unavailable service port numbers at participating nodes.

Today, most of the poisoning is due to advertisements with random identifiers; henceforth, for the sake of clarity, we describe the attack in that context.

When a user attempts to download an advertised copy with a non-existent identifier, the file-sharing system fails to locate the copy and typically responds with "more sources needed" or "looking". For the index poisoning attack, the attacker's goal is to trick the user into repeatedly selecting advertised copies with a non-existent identifiers. To achieve this goal, the attacker advertises a large number of copies with distinct, random identifiers, thereby densely populating the lines in the GUI with poisoned, non-existent copies. Additionally, the attacker may mix in polluted versions, allowing the user from time-to-time to discover and download an existing version, only to find that it is corrupted. Henceforth, we use the terminology **decoy attack** to refer to the index poisoning attack or the pollution attack (or a combination of the two attacks).

### C. NATs and P2P

In order to properly measure the extent of ongoing poisoning and pollution attacks, it is important to understand how NATs impact P2P file sharing. A significant fraction of P2P nodes reside behind NATs. Our measurement work has found that the fraction of nodes behind NATs ranges from 1/3 to 1/2 depending on the file-sharing system and the time of day. One of the problems with NATs is that it is difficult to initiate a TCP connection to a node behind a NAT (unless the NAT for that node has been manually configured or if a workaround scheme is employed [16]). Most file-sharing systems – including Overnet and FastTrack – do not permit file transfers between two NATed peers.

In order to reach both NATed and unNATed users, a pollution attacker will need to be unNATed. For these reasons, pollution attack nodes are typically unNATed. For the poisoning attack, even though the attack nodes do not transfer files, there are advantages for being unNATed in FastTrack. We have observed that poisoning attack nodes are unNATed for both FastTrack and Overnet. It should be noted, however, that throughout this paper we never assume that the decoy attackers are unNATed. Our methodology applies when the decoy attackers have all or some of their servers behind NATs.

### D. Definition of Poisoning and Pollution Levels

One of the contributions of this paper is an efficient methodology for estimating poisoning and pollution levels of titles in file sharing systems. Now we more precisely define the terms "poison level" and "pollution level" for a title. We do this for both versions and copies. In any P2P file-sharing system, as nodes connect and disconnect, copy advertisements are published and de-published (or timed out) in the distributed index. Suppose the investigation period is the time interval $(t, t+\Delta)$ where $\Delta$ is on the order of hours. Fix a particular title, $T$.

Let $\mathcal{V}$ be the set of all versions advertised over this period for the title $T$. Each advertised version is either a poisoned version (that is, non-existent), a polluted version (existent but intentionally corrupted) or a clean version. Denote by $\mathcal{V}_1$, $\mathcal{V}_2$ and $\mathcal{V}_3$ for the sets of poisoned, polluted, and clean advertised versions. We define the **version poison level** to be the fraction of poisoned versions advertised during the investigation interval, that is, $L_1^V = |\mathcal{V}_1|/|\mathcal{V}|$. Similarly, the **version pollution level** and **version clean level** are defined to be $L_2^V = |\mathcal{V}_2|/|\mathcal{V}|$ and $L_3^V = |\mathcal{V}_3|/|\mathcal{V}|$, respectively.

We now define the copy poison level and copy pollution level, which we more simply refer to as the **poison level** and **pollution level**. For each version $v \in \mathcal{V}$, let $\mathcal{C}_v$ be the set of all distinct copies advertised during the investigation interval for the version $v$. Then the poison level $L_1$, pollution level $L_2$, and clean level $L_3$ are defined as:

$$L_i = \frac{\sum_{v \in \mathcal{V}_i} |\mathcal{C}_v|}{\sum_{v \in \mathcal{V}} |\mathcal{C}_v|}, \qquad i = 1, 2, 3$$

## III. The Index Poisoning Attack in FastTrack and Overnet

The paper investigates the poisoning attack in two file sharing systems, FastTrack and Overnet. These two file systems have very different architectural designs: FastTrack uses an unstructured, two-tier architecture; Overnet uses a structured Distributed Hash Table (DHT) architecture.

### A. Overview of FastTrack

With more than two million simultaneous active nodes (in June 2005), FastTrack remains one of the largest P2P file sharing systems. It is used by several FastTrack clients including Kazaa, kazaa-lite, Grokster, and iMesh.

FastTrack is decentralized and unstructured (that is, no DHT). FastTrack has two classes of nodes, Ordinary Nodes (ONs) and SuperNodes (SNs). SNs have greater responsibilities and are typically more powerful than the ONs with respect to availability, Internet connection bandwidth and processing power. When an ON launches the FastTrack application, the ON establishes a TCP connection with a SN, thereby becoming a "child" of that SN. For each file it is sharing, the ON then provides to the SN the file's version identifier and metadata. This allows the SN to maintain a local index which includes identifiers, metadata (including title), and locations for all the files its children are sharing along with locations. Importantly, when a ON disconnects, its SN removes it from its index. Each SN also maintains long-lived TCP connections with other SNs, creating an overlay network among the SNs.

When a user wants to find files, the user's ON sends a query with keywords over the TCP connection to its SN. The SN may forward this query to one or more of its neighbors. If a SN finds a match in its local index, the SN returns a reply, which follows the reverse path in the overlay. A reply may contain multiple advertised copies. For each advertised copy in the reply, the reply contains the version identifier, the location of the copy (IP address and service port number) and metadata. The GUI displays one result per line for each unique identifier received.

### B. Index Poisoning in FastTrack

For FastTrack, the index poisoning attack is to insert bogus records into the indexes of the SNs. To pollute a SN node index, the attack node must create and maintain a TCP connection to the SN, and publish either bogus identifiers (not corresponding to any existent copies), bogus IP addresses or bogus port numbers.

For FastTrack, the index poisoning attack requires ongoing TCP connections to each targeted SN. FastTrack has roughly 20,000 supernodes in operation [3]. The more SNs that an attacker index poisons, the more successful the attack. A single attack node can maintain TCP connections with hundreds of SNs, and over each of these connections it can advertise thousands of bogus versions for one or more titles.

### C. Overview of Overnet

Overnet is part of eDonkey2000, which is one of the more popular file sharing systems today [1]. Overnet is one of the largest deployed DHTs today, with about one million nodes participating in the overlay at any one time, according to statics reported in eDonkey2000 GUI. eMule, an open-source "cousin" of eDonkey2000, also has a DHT-based network with a large number of users.

Overnet is based on the Kademlia DHT [14]. All participating nodes have equal roles and no hierarchy exists. When a client joins Overnet, it joins with a 128-bit ID. Presented with any 128-bit key, the Kademlia DHT finds the nodes that have the closest IDs to the key. To locate the closest nodes, Overnet uses UDP messages and iterative searches. Thus, to find the closest node to a key, the querying client will send UDP messages to a sequence of nodes, with each node in the sequence having an ID that is closer to the key.

When a node joins Overnet, it advertises information about the files it is sharing. Specifically, for each file it is sharing, after hashing the file to obtain the version identifier, the joining node sends into the DHT a message with the file location (IP and port) with the identifier as the key. This message is routed iteratively to the nodes that are close to the identifier in the ID space. When these nodes receive the message, they update their local indexes with a record <key, value>, where key is the version identifier and value is the joining node's identifier, IP address, and port number. Furthermore, for each file it is sharing, the joining node extracts keywords from the file's metadata and hashes each keyword into a 128-bit key. For each such keyword, the joining node advertises a record <key, value>, where key is the hash of the keyword and value is the version identifier. That record also contains the metadata information for the file, such as artist, title, album, file size, etc. Thus, the records in the Overnet index are distributed across all of the Overnet nodes with a two-phase publishing mechanism.

When a user queries for a title, the following steps are taken:

1) The client hashes each keyword (with more than a few characters) provided by the user.
2) For each hashed keyword, the client iteratively searches the DHT, with the key being the hashed keyword. When a query reaches a node that has records for the keyword, the node sends to the client all of the version identifiers for which it has corresponding records (along with the associated title names and other metadata). The requesting node thus receives several sets of identifiers, one set for each keyword. The client filters the responses, keeping only those title names that match all the keywords. The GUI then displays all of the filtered responses, with each version on a different line.
3) The user selects a version for downloading. The client iteratively queries the DHT, with the key being the version identifier. When a query reaches a node that has records for the version identifier, the node sends to the client the the locations of the copies of the versions.
4) The client then attempts to transfer the title from those locations.

### D. Index Poisoning in Overnet

The attacker first determines keywords from the title and hashes the keywords. The attacker then poisons with one of two approaches:

1) The attacker generates a random identifier, not derived from the hash of some file. The attacker publishes <key, value> pairs, where key is the hash of a keyword and value is the random identifier. When a user searches

for the keyword, the user's client receives the bogus identifier. If the user selects the bogus identifier, the DHT displays "looking" and searches indefinitely for the identifier.

2) The attacker first publishes <key, value> pairs, where the key is a keyword hash and value is a version identifier. The attacker then sends a second publish message for <key, location> where the key is the same version identifier inserted in the previous step, but the location is bogus (for example, non-present IP address).

Although both of these approaches are feasible, the first approach is clearly simpler. Our measurement work (see Section 5) has determined that attackers are currently using the first approach.

Because Overnet uses UDP for messaging, including for the publish messages, the attacking node does not need to maintain any TCP connections to index pollute a particular title. The attacker node simply needs to determine the DHT nodes responsible for the keywords. Once having determined those nodes, the attacker can send publish messages directly to those nodes. For each keyword in the title, the attacker can advertise hundreds or even thousand of identifiers. To maintain this bogus information in the index, the attacker must periodically refresh the poisoned information. DHTs, in general, are highly vulnerable to the index poisoning attack for targeted titles, unless appropriate counter measures are taken. There is also some "natural" poisoning in a DHT because the indexes contain "soft state" and are maintained with refresh messages.

## IV. Methodology

To estimate poisoning and pollution levels, the "straightforward" approach is to query the file-sharing system, sample copy advertisements, attempt to download versions from those advertisements, and then attempt to determine if the downloaded versions are clean or polluted. This approach has several flaws. First, it is difficult to develop a reliable automated procedure to determine if a version is polluted or not. The paper [2] presented an automated procedure that had relatively small false positive and false negative probabilities. Since that study, attackers have begun to corrupt files in a variety of different ways, rendering automated detection more difficult if not impossible. The alternative to an automated procedure is to manually listen to (or watch) the versions. Such a manual procedure consumes hundreds of hours of human resources per title. Second, in addition to the human resources, this approach requires the downloading and storage of a large number of versions, consuming large amounts of bandwidth and storage resources. In particular, hours or even days can be required per title to download a sufficient number of versions to get an accurate estimate of pollution levels. Third, if a download attempt fails, the approach may falsely declare the copy advertisement as poisoned, when in actuality the sources with the version may have full request queues.

Our methodology, which does not involve the downloading of any files, has the following steps:

1) Over the measurement period, track the advertised copies. For each advertised copy, we record the version identifier along with the node that published the message. We refer to this step as **harvesting**. From the harvested data, create a list of the advertised versions, and for each advertised version a list of the distinct advertised copies. This step needs to be customized for each file-sharing system.
2) Determine from the harvested data which advertised versions are poisoned, which are polluted, and which are clean. This step is generic to many P2P file sharing systems and involves a crucial substep in determining the sources that are responsible for poisoning and polluting the system.
3) Determine the poison and pollution levels, for both versions and copies, based on the equations given in Section 2.

We now elaborate on each of these three steps. Let $\mathcal{T}$ denote the set titles under investigation.

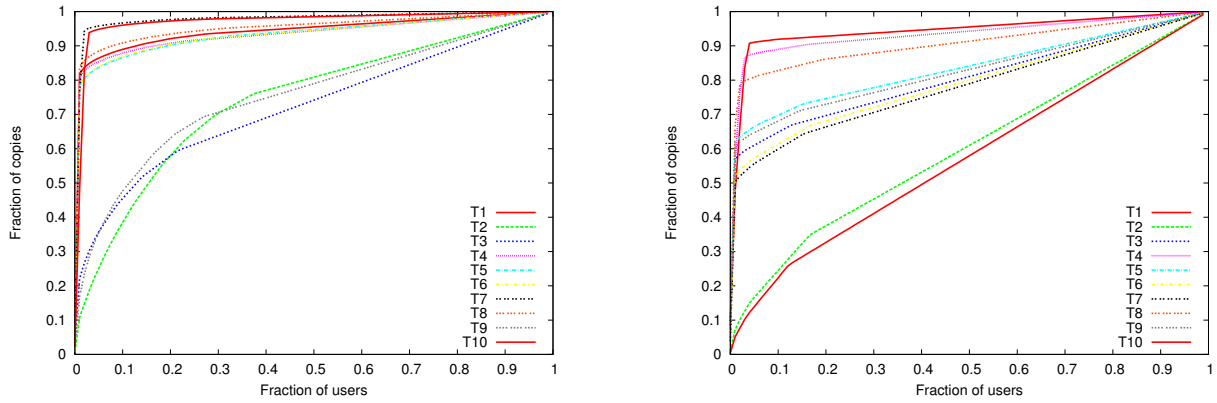
### A. Harvesting in FastTrack

In the first step we need to determine the distinct advertised copies for each of the investigated titles. In [2] we described our FastTrack Network crawling platform. In this paper, we use that crawler to obtain the following information for each title investigated over the measurement period: the set of version identifiers; for each identifier, a list of advertised copies, and for each copy on the list, the source IP address and the service port number. If the source is behind a NAT, then this harvested source IP address will be the node's private IP address. Because many NATed nodes have the same IP address, we have to take special care to distinguish the different nodes and properly count the distinct advertised copies. Very few FastTrack nodes use the default service port [3] and the majority choose service port numbers at random. Thus, we can distinguish between distinct copy advertisements by using the tuple (IP address, service port number), even when nodes are behind NATs.

For the investigated titles, we selected 10 songs from the iTunes top-100 list in April of 2005 [6]. We harvested the copy information for these 10 titles for a 1-hour period in April 2005.

### B. Harvesting in Overnet

Whereas in FastTrack we harvest by crawling the network, in Overnet we harvest by inserting our own Overnet nodes into the DHT. For each title, we extract one of the keywords from the title's name and hash it. For each hashed keyword, we insert our own Overnet node with node ID configured to be the 128-bit value of the hashed keyword. In this manner, users that advertise a copy for the title will send a publish message to our inserted node. The inserted node thus collects advertisements for the title over the measurement period. Each advertisement is encapsulated in a UDP datagram and includes the the name of the title, the identifier (content hash) for the version, as well as other information such as duration, size, etc.




(a) FastTrack

(b) Overnet

Fig. 1. Cumulative fraction of copies from ranked users.

The UDP datagram header provides the source IP address and the source (messaging) port number. If the source is behind a NAT, then we still receive the public IP address and public port number. We then keep a record of all the distinct copies that are advertised to the inserted node. In Overnet, we consider two advertised copies to be the same if they have been advertised with the same source IP addresses and source port numbers; otherwise, we consider the advertised copies as distinct. Note that if multiple users are advertising distinct copies of the same version from behind a NAT, our procedure will correctly identify the advertisements for distinct copies, as the users will have different public port numbers. (We could have also identified unique Overnet nodes with Overnet node IDs as in [17]. However, because it is easy to generate arbitrary Overnet IDs, we elected to use IP addresses and port numbers instead.)

For the investigated titles, we selected 10 songs from the iTunes top-100 list in June of 2005 [6]. These selected titles are distinct from those selected for the FastTrack experiments because of the dynamic nature of popular music charts. We harvested the copy information for these 10 titles for a 1.5-hour period in June 2005. We chose the time period to reflect the rate that Overnet clients use to refresh their shared files.

### C. Classifying the Versions

After harvesting the requisite data, the next step is to determine which versions are poisoned, which are polluted, and which are clean. We now present a novel methodology that can be applied to any P2P file-sharing system. Throughout the remainder of the paper, we define a **user** to be the tuple (IP address, service port number) for FastTrack and (IP address, messaging port number) for Overnet.

We begin with the following observation: For many of the investigated titles, among the users that have at least one version of the title, *the large majority of users advertise at most a few versions of the title and a relatively small number of users advertise a large number of versions of the title.* Figures 1(a) and 1(b) support this claim for FastTrack and Overnet, respectively. In these figures, for each title, we re-order the users according to the number of copies of the title they advertise. The user with the most copies is ordered first. With this ordering (which is different for each title), Figure 1 plots the CDF of the fraction of copies with respect to the fraction of users. If all users were sharing the same number of copies, then this curve would be a straight line running from point (0,0) to (1,1). For FastTrack, we see that many curves quickly rise, with 5% of the users advertising more than 85% of the copies. Similarly, for Overnet, for many of the titles, the large majority of the users advertise few versions whereas a small minority advertise a large number of versions.

Having made this observation, we now state our methodology, which can be broadly applied to most P2P file-sharing systems. The first step is to determine the users that are advertising relatively large numbers of versions of the same title. To this end, let $\mathcal{U}$ be the set of user that have advertised at least one version of one of the investigated titles. For each $u \in \mathcal{U}$, let $V_u^T$ denote the number versions of title $T$ that user $u$ is advertising. Let

$$V_u^{\max} = \max_{T \in \mathcal{T}} V_u^T$$

be the maximum number versions across all titles that user $u$ is advertising.

Let $m^{\max}$ be the mean of the $V_u^{\max}$ values across all users, that is,

$$m^{\max} = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} V_u^{\max}$$

Let $K$ be a fixed constant. We then say that user $u \in \mathcal{U}$ is a **heavy user** if $V_u^{\max} \geq K m^{\max}$; otherwise, we say $u$ is a **light user**. Let $\mathcal{H}$ be the set of heavy users and $\mathcal{L} = \mathcal{U} - \mathcal{H}$ be the set of light users. Because the users in $\mathcal{H}$ advertise a relatively large numbers of copies for at least one of the investigated titles, we naturally suspect them of being the decoyers. However, we do not make this assumption in the following analysis.

Now recall that $\mathcal{V}$ is the set of all advertised versions. Denote $\mathcal{V}_H$ for the set of versions that have been advertised by users in $\mathcal{H}$ (advertised by the heavy users). Denote $\mathcal{V}_L$ for the versions that have been advertised by the users in $\mathcal{L}$ (advertised

the light users). Some versions may be advertised by both heavy and light users. Let $\mathcal{V}_X = \mathcal{V}_H \cap \mathcal{V}_L$, $\mathcal{V}_H^* = \mathcal{V}_H - \mathcal{V}_X$, and $\mathcal{V}_L^* = \mathcal{V}_L - \mathcal{V}_X$. Observe that $\mathcal{V}_X$ is the set of versions that have been advertised by both the heavy and light users; $\mathcal{V}_H^*$ is the set of versions that have been advertised by the heavy users but not by the light users; and $\mathcal{V}_L^*$ is the set of versions that have been advertised by the light users but not by the heavy users. Note that the sets $\mathcal{V}_H^*$, $\mathcal{V}_X$, $\mathcal{V}_L^*$ form a partition of the set all advertised versions $\mathcal{V}$.

Our claim is that the vast majority of the advertised versions in $\mathcal{V}_H^*$ are poisoned versions, the vast majority of the advertised versions in $\mathcal{V}_X$ are polluted versions, and the vast majority of the advertised versions in $\mathcal{V}_L^*$ are clean versions. The intuition behind this claim is as follows. We suspect the users in $\mathcal{H}$ to be decoyers and the users in $\mathcal{L}$ to be regular users. If a version has been advertised by a regular user but not by a decoyer, then the version is likely to be clean; if a version has been advertised by both a decoyer and a regular user, the version is likely polluted (as it has spread from the decoyers to the set of regular users); finally, if the version is advertised by a decoyer but not be an regular user, the version is almost certainly poisoned.

To evaluate this heuristic, we carry out the following experiment:

1) Query for the title and obtain an advertised identifier $v$ for the title.
2) For Overnet, we query again with the identifier $v$. If no reply is returned, we conclude that the version is poisoned.
3) For FastTrack, from the first query we have obtained at least one advertised location (IP address and port number) for $v$. We then attempt to download from that location. If we fail to establish a connection and get a TCP RST segment, then we conclude that $v$ is a poisoned version. Otherwise, we download a copy of $v$
4) If we have downloaded a copy of $v$, we check to see if it is polluted, as described below.
5) We also check to see which of the sets $\mathcal{V}_H^*$, $\mathcal{V}_X$, $\mathcal{V}_L^*$ the version belongs to.

Table I provides the results for FastTrack. ¿From the 44,929 versions we obtained via the queries, 8,450 belong to $\mathcal{V}_X$, 33,186 belong to $\mathcal{V}_H^*$, and 3,293 belong to $\mathcal{V}_L^*$. For each available version, we examined the binary content to determine whether the the version is polluted or not. This can be a time-consuming task, requiring exorbitant human resources. To reduce the human intervention, we did the following preprocessing to reduce the number of versions to be listened to manually:

1) Recalculated the hash value of the complete file to check that it matched its declared identifier. If not, we concluded that the version was polluted.
2) Compressed the file using the gzip tool. Since mp3 and wma files are already highly compressed, if the compression ratio was high, we concluded that the file was polluted.
3) Attempted to decode the file to check if the file is decodable (ffmpeg project [7]). If not, we concluded that the file is polluted.

| | # of Versions | # of Success | Accuracy |
|---|---|---|---|
| $\mathcal{V}_X$ | 8,450 | 8,400 (polluted) | 99.4% |
| $\mathcal{V}_H^*$ | 33,186 | 32,021 (unavailable) | 96.5% |
| $\mathcal{V}_L^*$ | 3,293 | 3,097 (clean) | 94.1% |

TABLE I
EVALUATION OF THE VERSION CLASSIFICATION SCHEME.

A relatively small number of versions passed all three of these tests. We manually listened to each of the remaining versions to determine whether it was polluted or not.

Table I shows that 99.4% of the versions in $\mathcal{V}_X$ were actually polluted, 96.5% of the versions in $\mathcal{V}_H^*$ were actually poisoned, and 94.1% of the versions in $\mathcal{V}_L^*$ were actually clean. Overall, the scheme correctly classifies more than 96% of the versions, validating our claim. Because this experiment is very intensive in human resources, it was only done for FastTrack. Henceforth, we refer to heavy users as decoyers and light users as regular users.

### D. Poisoning and Pollution Levels

¿From our validated claim, it follows that version poison, version pollution, and version clean levels can be accurately approximated by $L_1^V = |\mathcal{V}_H^*|/|\mathcal{V}|$, $L_2^V = |\mathcal{V}_X|/|\mathcal{V}|$, and $L_3^V = |\mathcal{V}_L^*|/|\mathcal{V}|$, respectively. Similarly, the copy poison level $L_1$, copy pollution level $L_2$, and copy clean level $L_3$ can be accurately approximated by:

$$L_1 = \frac{\sum_{v \in \mathcal{V}_H^*} |\mathcal{C}_v|}{\sum_{v \in \mathcal{V}} |\mathcal{C}_v|},$$

$$L_2 = \frac{\sum_{v \in \mathcal{V}_X} |\mathcal{C}_v|}{\sum_{v \in \mathcal{V}} |\mathcal{C}_v|},$$

and

$$L_3 = \frac{\sum_{v \in \mathcal{V}_L^*} |\mathcal{C}_v|}{\sum_{v \in \mathcal{V}} |\mathcal{C}_v|}.$$

Note that all the sets $\mathcal{V}$, $\mathcal{V}_H^*$, $\mathcal{V}_X$, $\mathcal{V}_L^*$, and $\mathcal{C}_v$, $v \in \mathcal{V}$ employed in these estimates are obtained directly from harvested metadata; binary files are not used to create the estimates.

## V. MEASUREMENT RESULTS

### A. Data Set

Table II summarizes a one-hour raw data set collected by the FastTrack crawler in April 2005. Depending on the title, the number of users sharing the title ranged from about 5,000 to 38,000, the number of advertised copies ranged from about 55,000 to 300,000, and the number of advertised versions ranged from about 15,000 to 160,000. The number of copies per user ranged between 1.93 for title T3 to 38.97 for T7. A large number of average copies per user suggests some level of decoying, since under normal conditions a user would only have few versions of the same title (maybe different recordings), and in most cases just one.

| Titles | # of users | # of copy | # of versions | # of public IP |
|---|---|---|---|---|
| T1 | 11,151 | 12,4349 | 44,317 | 7,725 |
| T2 | 38,189 | 99,771 | 22,290 | 20,997 |
| T3 | 28,366 | 54,903 | 23,080 | 19,749 |
| T4 | 13,827 | 123,925 | 42,771 | 9,756 |
| T5 | 20,557 | 195,967 | 66,628 | 14,178 |
| T6 | 21,529 | 197,613 | 67,095 | 14,511 |
| T7 | 7,938 | 309,409 | 162,999 | 5,601 |
| T8 | 16,247 | 231,051 | 102,933 | 11,140 |
| T9 | 10,878 | 25,881 | 14,738 | 7,785 |
| T10 | 5,083 | 168,714 | 55,967 | 3533 |

TABLE II

RAW DATA FOR TEN TITLES FROM FASTTRACK

| Titles | # of users | # of copy | # of versions | # of public IP |
|---|---|---|---|---|
| T1 | 378 | 4,159 | 3,901 | 375 |
| T2 | 2,010 | 2,531 | 730 | 1,979 |
| T3 | 1,184 | 3,084 | 1,115 | 1,172 |
| T4 | 535 | 4,633 | 4,290 | 531 |
| T5 | 1,372 | 4,280 | 2,891 | 1,359 |
| T6 | 2,133 | 5,265 | 3,171 | 2,100 |
| T7 | 2,213 | 5,159 | 3,085 | 2,191 |
| T8 | 884 | 5,024 | 4,219 | 870 |
| T9 | 1,485 | 4,342 | 2,907 | 1,469 |
| T10 | 2,038 | 2,398 | 337 | 2,020 |

TABLE III

RAW DATA FOR TEN TITLES FROM OVERNET

Table III presents summarizes our Overnet raw data. Depending on the title, the number of users sharing the title ranged from about 400 to 2,200, the number of advertised copies that we identified range from about 2,400 to 5,300, and the number of versions ranged from about 300 to 4,300. The almost 10-fold difference is due to $(i)$ a smaller number of users in Overnet; and $(ii)$ different design architecture. In an ideal DHT each node is responsible for part of the key space. A massive DHT based system, however, requires more relaxed rules. As described in [14] a node is to publish information for a key at a number of other nodes that are "close" to the keyword hash. However, the closeness criterion is determined by the implementation. While crawling Overnet we have found that as many as 1,600 nodes reply with metadata information for the same keyword. Moreover, when publishing the information about its content, the Overnet client only sends publish messages to a few nodes. Thus, the index for a single keyword is distributed around a large part of the DHT and not just around few nodes. As described in Section IV, to harvest data in Overnet, we inserted Overnet nodes. In an ideal DHT our inserted nodes would have captured all the publish messages for titles containing the targeted keywords, since their IDs were exactly equal to the hash values of those keys. However, because the Overnet implementation spreads the advertisements over many nodes, our inserted nodes capture only a fraction of the advertisements. Thus the poisoning and pollution level results presented for Overnet only provide rough estimates.

### *B. Decoyer Detection*

Applying the methodology of Section IV to the ten titles under investigation, we identified 8,683 decoy users (that is, heavy users) from 624 IPs in FastTrack and 27 decoy users from 26 IPs in Overnet. We found no NATed decoy users in this measurement. The FastTrack decoyers simultaneously emulate 13.9 client instances per IP on average, compared to 1.43 instances per IP for regular users. The explanation of these client-to-IP ratios is that to reach more users, the decoyer needs to create more clients from every IP source to compete with the massive regular user space. In FastTrack, the average number of copies per user for a decoyer is 136.3, much higher than 2.9 for regular users. Decoyers are less than 7% of all users but they provide over 77% of all copies and 72% of all versions. Although on a smaller scale, our Overnet results show a similar trend with about 23,771 copies and 22,678 versions provided by decoyers and only 17,104 copies and 3,907 versions provided by regular users.

| | # of IPs | # of users | # of copies | # of versions |
|---|---|---|---|---|
| Decoyer | 624 | 8,683 | 1,183,622 | 443,102 |
| Ordinary | 82,015 | 117,673 | 347,939 | 167,103 |

TABLE IV

THE CHARACTERISTICS OF DECOYERS AND REGULAR USERS OVER THE TEN TITLES FROM FASTTRACK NETWORK

| | # of IPs | # of users | # of copies | # of versions |
|---|---|---|---|---|
| Decoyer | 26 | 27 | 23,771 | 22,678 |
| Ordinary | 12,135 | 12,545 | 17,104 | 3,907 |

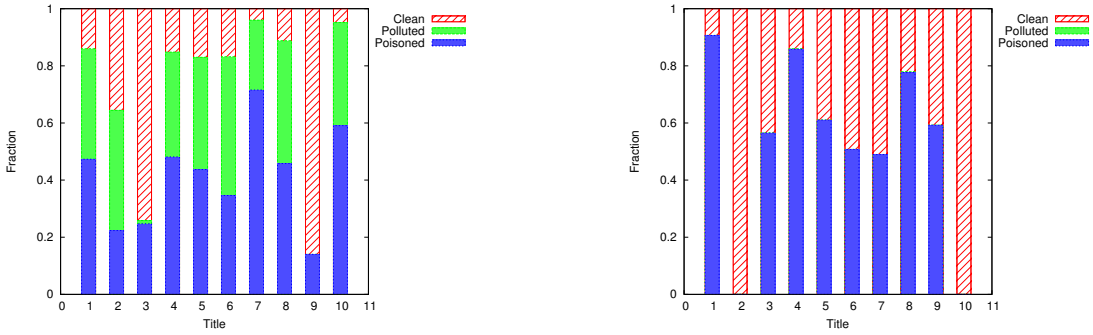
TABLE V

THE CHARACTERISTICS OF DECOYERS AND REGULAR USERS OVER THE TEN TITLES FROM OVERNET NETWORK

Figure 2 presents the copy poison, pollution, and clean levels, based on our methodology, for the ten titles for both FastTrack and Overnet. First consider the FastTrack network. Two titles, T3 and T9, are relatively clean, with practically no pollution and less than 25% poisoning. Because there are several independent companies (working for the copyright industry), each with its own decoy technique and strategy, it is not surprising that some titles suffer only from poisoning and not from pollution. The remaining eight titles under investigation for FastTrack are under heavy poisoning and pollution attacks. The total decoy percentage (poison percentage plus pollution percentage) ranges from 65% to as high as 95%. Our results in Overnet indicate extensive poisoning and little pollution. Two of the investigated titles (T2 and T10) are found to be clean, whereas the remaining titles have poisoning levels ranging from 50% to as much as 90%.

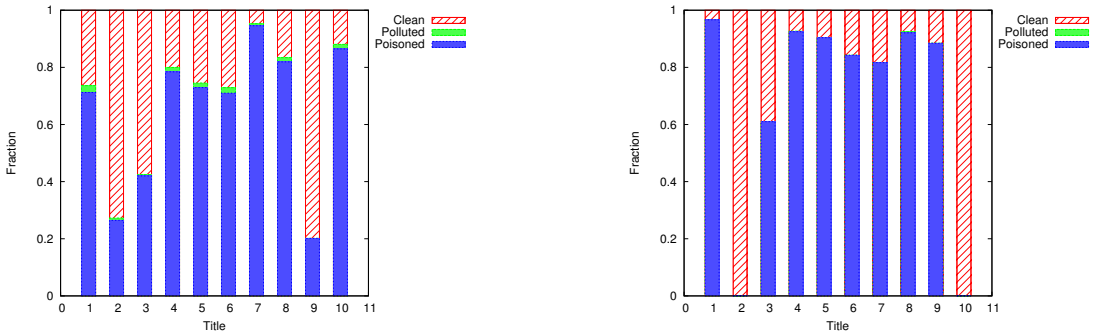
Figure 3 presents the version poisoning, pollution, and clean levels. We see that, in both networks, for most of the investigated titles, the majority of the versions are poisoned. An exception again are the relatively clean titles T2 and T9 in FastTrack and T2 and T10 in Overnet. The poisoning levels in both networks are comparable with about 70% to 90% of the targeted titles being poisoned. Why are the version poison levels generally much higher than the copy poison levels for the targeted titles? The explanation is twofold:

- The decoyers create massive numbers of polluted versions, which explains the large fraction of poisoned versions in these systems. But copies of these poisoned




(a) FastTrack (b) Overnet

Fig. 2. Estimates of poison/pollution/clean levels by copies for 10 titles




(a) FastTrack (b) Overnet

Fig. 3. Estimates of poison/pollution/clean levels by versions for 10 titles

| Title | Poisoned | Polluted | Clean |
|---|---|---|---|
| T1 | 1.86 | 45.38 | 107.4 |
| T2 | 3.77 | 228.25 | 42.23 |
| T3 | 1.40 | 6.67 | 106.12 |
| T4 | 1.78 | 68.53 | 50.16 |
| T5 | 1.76 | 71.16 | 229.45 |
| T6 | 1.43 | 74.90 | 222.7 |
| T7 | 1.44 | 65.95 | 69 |
| T8 | 1.26 | 64.84 | 204.1 |
| T9 | 1.22 | n/a | 20.39 |
| T10 | 2.05 | 67.73 | n/a |

TABLE VI
NUMBER OF COPIES PER VERSION FOR TOP 1000 FASTTRACK VERSIONS

versions do not circulate and multiply as do the clean and polluted versions.

- The decoyers often make available a large number of copies of the same polluted version, thereby increasing the availability of the (very frustrating) polluted files in the users' GUIs.

Our findings shown in Table VI confirm our expectation of the uniqueness of poisoned identifiers. The table shows the average number of copies per version for the top 1000 versions in FastTrack. Poison versions on average have between 1 and 2 copies per version, compared to a higher number for clean and polluted versions.

## VI. DEFENDING AGAINST THE POISONING ATTACK

We have shown that existing P2P file-sharing systems are highly vulnerable to index poisoning attacks. In this section we address possible directions for countering the attack. As we shall see, it is a challenging problem to defend against the poisoning attack in a P2P file-sharing system.

We first observe that this vulnerability is not a consequence of having an index system distributed over peer nodes. The original Napster design would have also been vulnerable to the poisoning attack. In fact, centralized systems are even more vulnerable than decentralized systems since bogus records only need to be inserted in one node.

### A. Rating Versions and Advertisements

One possible direction is to authenticate versions to defend against the pollution attack and to authenticate advertisements to defend against the index poisoning attack. An example of a content-rating system is Amazon's review system, which helps prospective customers anticipate the quality of the goods being sold. For file-sharing systems, some content rating websites and forums have been initiated. Separate from the file sharing systems, these forums include verified file hashes or download links to counter the pollution attack [20] [21]. These hash-checking systems are enabled by volunteer postings, typically with a moderator to manage disputes. These centralized hash-checking systems are subject to legal attack: The popular eDonkey link forum was forced to shut down; Supernova, the Bitorrent resource link website [19] [22], had a similar fate.

Although version rating and verification schemes can in some cases be used to authenticate content [25], it is more

difficult to authenticate advertisements, that is, verify that a source that advertises a hash actually has a version that corresponds to the hash. This is a challenging problem, particularly in a distributed system. But even if it were possible to authenticate advertisements, the attackers could simply acquire or create versions, advertise their hashes, have their hashes authenticated, and then not upload files in response to requests. Note that this attack works even if the hashes are assigned meaningful reputations.

### *B. Rating Sources*

The eBay feedback system enables a buyer/seller to rate its counterpart in terms of transaction performance. The rating history helps build the reputations of individual buyers and sellers. Similarly, a promising approach for file sharing is to recognize that there are good sources, which advertise and upload files they actually have, and bad sources, which index poison and pollute the system. If the bad sources can be identified, they can then be blacklisted. A well-known centralized forum was introduced to aggregate suspicious sources reported by users [24]. Ideally, one would like to have a distributed scheme that responds to attempts to evade detection by changing identity. (Here, identity can be the source IP address, a password-authenticated username, or a certified public-key.)

We now outline a distributed approach that could be applied to two-tier unstructured systems, such as FastTrack. This approach is based on our decoyer detection methodology discussed in Section 4. For the sake of discussion, we simply use the IP address of the source for source identification. (Decoyers sit in front of NATs; but to handle the remote possibility that a decoyer is NATed, we can augment the definition of a source with the service port number.) Since in FastTrack, advertisements are maintained with ongoing TCP connections, the source IP address of an advertisement is authenticated by the establishment of a TCP connection. In devising this scheme, we again make use of the observation that decoyers advertise a large number of versions of the same title.

Decoyers typically control narrow blocks of IP addresses and can easily move their nodes from one IP address to another within the block. Thus, rather than assigning reputations to individual IP addresses, we assign reputations to narrow ranges of IP addresses. For the IP range, we use /n subnets. One needs to choose n small enough so that both decoyers and ordinary users do not operate from within the same prefix; and large enough to cover multiple decoying servers in the same subnet. For correctness, we use /n = /24 for the remainder of this discussion.

In this scheme, each supernode creates a local reputation for each /24 prefix that is advertising directly to the supernode. The reputation is a function of the number of copies per title being advertised by the prefix; if a prefix advertises a large number of copies of any one title, it will likely have a low reputation. Specifically, for each such advertising prefix $i$, let $W_i^T$ denote the number of copies of title $T$ that are advertised to the supernode (including replica copies across nodes in prefix $i$). Also, for each prefix $i$, the supernode determines $y_i^T$, the number of nodes from prefix $i$ that advertise title $T$. We then define the local reputation (local to the supernode) of prefix $i$ as

$$r_i = \frac{1}{\max_T W_i^T / y_i^T}, \tag{1}$$

where the max is taken over all titles advertised by subnet $i$. Note that reputations range from 0 to 1, with higher values $r_i$ corresponding to better reputations. In this manner, each SN can create a local reputation list, which contains all the prefixes directly advertising to the SN along with the reputations of those prefixes.

Each SN would also maintains its own global reputation list, which includes all IP prefixes advertised to all the SNs. SNs would exchange with each other their global reputation lists. When a SN receives global lists from other SNs, it combines the received lists with its own local list to create a new global reputation list at that node. For this purpose, a reputation algorithm such as Eigentrust [18] could be used. (Eigentrust also includes a distributed implementation.) In this manner, by continually exchanging and combining, each SN obtains reputation values for prefixes, reflecting the prefixes global behavior throughout the file-sharing system. Each SN periodically refreshes its local reputation lists, recalculates its global view, and exchanges its global view with other SNs. The lower the reputation, the more likely the prefix is a decoyer. Each SN could periodically send its global reputation list to its children nodes, and the children nodes could use a peer-dependent threshold for blacklisting peers with reputations lower than the threshold.

Figure 4 shows the reputation values of all /24 subnets that participated in FastTrack during our measurement interval. The maximum in (1) is taken over the ten investigated titles. The x-axis of this figure is the IP space, ranging from 0.0.0/24 to 255.255.255/24; the y-axis is the reputation values of the discovered subnets. The reputations in Figure 4 are obtained in a centralized manner by merging the reputations from the individual supernodes. Although this figure results from a "centralized" algorithm, it illustrates the potential of a distributed algorithm. The blacklisting threshold could be .005, thereby blacklisting the subnets with reputations at the bottom of the figure. This threshold would result in blacklisting 53 of the advertised /24 networks (which includes 624 observed IP addresses) and not blacklisting the remaining 62,403 of the remaining advertised /24 networks (which includes 82,015 advertised IP addresses).

One problem with such a scheme is that some of the SNs actually may be operated by attackers. But if the number of compromised SNs is small, then a reputation scheme such as Eigentrust should give meaningful results [18]. Another issue is the scheme as described requires each SN to classify each received advertisement into a title. To this end, automated classification mechanisms could be employed. Alternatively, we could define reputation simply in terms of total number of copies advertised (rather than per title), and then employ a

voting mechanism to identify the subnets that advertise many copies to many SNs.

A similar approach can be taken in a DHT based system such as Overnet. Each node can maintain a local reputation list for each /24 prefix that publishes metadata to it. In Overnet it is easy to keep track of the number of versions for every keyword since it is used as a key in the first round of file information publishing. Thus, nodes can calculate their local reputation lists in a manner similar to the approach proposed above for FastTrack. In order to take into account the breadth of metadata for every keyword discussed in Section V-A, nodes with close IDs can exchange their reputation lists and each of them can construct its view on the part of the network responsible for a set of keywords.

We have just sketched one possible approach for creating a blacklist in a distributed manner. The central assumption in this scheme is that each decoyers advertise a relatively large number of copies from /n networks. To circumvent blacklisting, the decoyers could deploy a massive number of hosts, each in a different /n network, and each advertising a relatively small number of copies. However, such a deployment would be unlikely due to exorbitant cost. Many variations of this reputation/blacklisting scheme are possible. In future research, it is of interest to analyze such variations in greater detail.

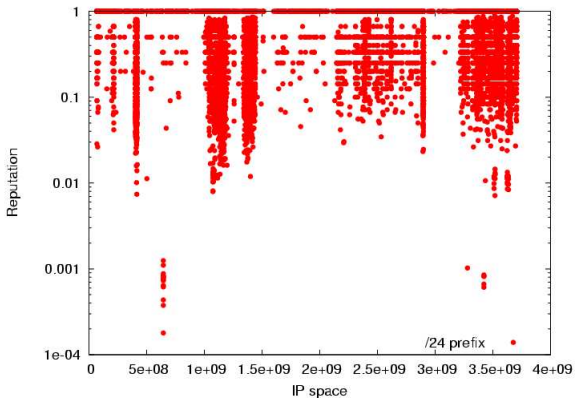



Fig. 4. Reputations of /24 prefixes

## VII. DHT Vulnerabilities to Poisoning

### A. Node Insertion Attack

Another form of the index poisoning attack is for the attacker to insert nodes into the file-sharing and manipulate the indexes of the inserted nodes. For example, in a two-tier unstructured system, such as FastTrack, the attacker could insert powerful nodes with the expectation that those nodes will be promoted to supernodes. Once the inserted nodes become supernodes, the attackers could manipulate the indexes in those nodes as they so desire. To date, we have not observed the presence of such a "node insertion" attack in FastTrack. This is probably because the FastTrack protocol is proprietary, uses encryption, and has not been fully cracked.

On the other hand, we have observed in our measurements the presence of the node insertion attack in Overnet. Specifically, in addition to actively advertising bogus versions of targeted titles, the decoyers have inserted their own nodes into the overlay network. Those nodes have identifiers that are close to targeted keyword hashes. When those nodes are queried, they respond with randomized identifiers; in fact, each query results in a fresh set of randomized identifiers! Figure 5 plots the number of unique identifiers returned as a function of the number of queries for four nodes. Two of these nodes are normal nodes and two these nodes we suspect to be malicious inserted nodes. As shown in the figure, the normal nodes provide no more than 100 identifiers since those nodes are making available a bounded number of files. The inserted nodes, however, continuously provide new identifiers, without bound. The node insertion attack for targeted titles is particularly pernicious in a DHT.

We remark that Figures 2 and 3 do not include the copies that are advertised by the malicious inserted nodes, since these malicious nodes do not publish advertisements. When a user queries with keywords for a title that has been targeted by the index poisoning attack, it may receive poisoned identifiers from the inserted nodes (not reflected in Figures 2 and 3) as well as from the regular nodes (reflected in Figures 2 and 3). Thus, for Overnet, the poisoning level is actually worse than that depicted in Figures 2 and 3.

This attack is very damaging to the P2P system in that it not only affects the targeted content, like the active poisoning attack does, but it can also prevent users from finding other items in the network. For example, if a title "word1 word2" is targeted by an attacker who's strategically inserted nodes reply bogus results to queries for either word1 or word2, then searches of any titles that include word1 or word2 will result in fewer results than in a clean system. If the title is just word1 or word2, then a user might only get bogus replies because of the intentional poisoning of "word1 word2".

Without observing the presence of this attack, Dumitriu et al [15] discuss a related attack and conclude that unstructured systems are more vulnerable than DHTs. Even though those conclusions appear to be contradictory to the ones provided here, this is not the case since they study a routing attack where malicious nodes attack the whole network and try to bring down its goodput. The attack that we observed is different in that it only targets a limited number of titles.

### B. Poisoning: DHT versus Unstructured

Because of the differences in their architectures, structured and unstructured systems require different amounts of effort to poison. In Section III we described how index poisoning can be done in both types of networks. In an unstructured P2P system, such as FastTrack, a successful attack will require that every emulated client maintains active TCP connections to all, or at least a great number, of the supernodes in the network. In Overnet, on the other hand, publishing of metadata is done by low-overhead UDP messages that need to be refreshed every hour or so. Furthermore, only the nodes responsible

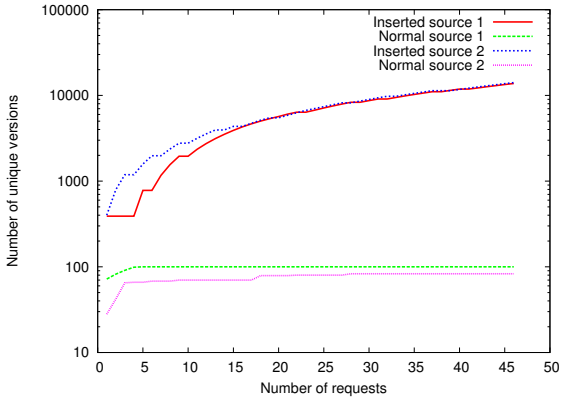



Fig. 5. Advertised versions from malicious inserted nodes and regular nodes

for the keyspace of the targeted titles need to be poisoned for successful attack. For this reason, when poisoning a small number titles, an attack on a DHT requires significantly less resources than an attack on a distributed unstructured system. For a large number of titles, however, the number of nodes that need to be contacted in a DHT will grow and eventually the resource requirements can become higher than those in an unstructured system.

### C. Leveraging a DHT for a DDoS Attack

Another possible attack, although currently not employed, is to cause a DHT to be the source of a powerful DDoS attack against any arbitrary host. Recall that in a DHT, sources sharing content advertise to the DHT information about the content as well as the source's IP address and port number. Thus, a malicious user can use the poisoning attack in a completely different context - not to attack a specific title, but instead attack a targeted host. If for a number of popular titles the attacker inserts into the DHT numerous poisoned records pointing to a targeted host, then the users that want to download those titles will be told that the content resides in the targeted host. The users will then repeatedly send requests to the targeted host. In a large scale P2P system, these requests may generate a successful DDoS attack. As of June 2005, we have experimented with Overnet and have found that it can easily be made the source of a DDoS attack. In fact during some of our experiments we advertised our own nodes so heavily that the responses that we received brought our university network down! We have alerted the Overnet team about the problem.

### D. Related Work

Although a relatively new Internet application, there are many measurement studies on P2P file-sharing systems. Bandwidth, availability, and TCP connection duration for popular filesharing systems such as Gnutella and Napster are examined in [8] [9] [11] [4]. P2P user behavior and content characteristics are studied in [10] [12].

Several independent research groups have developed crawlers for P2P file sharing systems. A crawler for the original single-tier Gnutella system is described in [4]. A crawler for the current two-tier Gnutella system (with "ultrapeers") is described in [5]. A crawler for the FastTrack P2P file sharing is described in [2].

In [2] it was established that pollution is widespread for popular titles. The methodology in [2] to determine pollution levels is inefficient in that it requires downloading and binary content analysis of hundreds of versions for every investigated title. In [13] the authors examine intentional and unintentional pollution and conduct a measurement study of content availability in the four popular P2P file-sharing networks. In [15], the authors develop a deterministic model for the pollution evolution in a file-sharing system. They also examine a "takeover attack" when the attackers either insert or takeover a limited number of P2P nodes.

A user-supported antip2p banlist is available in [24]. Walsh and Sirer propose an object rating scheme that is based on peer endorsements of objects [25]. As described above, object rating is unlikely to be successful for abating the index poisoning attack.

## VIII. Conclusion and Future Work

A critical component of a P2P file sharing system, the index allows users to discover and locate content in other peers. Unfortunately, arbitrary users can easily poison the index by advertising to it bogus records. It is difficult for a P2P file-sharing to authenticate the advertisements, that is, to verify that the advertised content is not only present but will also be uploaded upon request. Attackers have discovered this vulnerability and are now aggressively index poisoning popular file-sharing systems.

P2P file sharing is a natural and power application, allowing communities of users to pool files, locate pooled files, and download pooled files. The P2P file-sharing sharing paradigm has broad applications in business, educational, and community arenas. Unfortunately, without special care, the indexes in these systems can easily be compromised. When designing P2P file-sharing systems in the future, it is crucial that designers bear the poisoning attack in mind. We believe that counter measures based on content and advertisement authentication will be largely unsuccessful. A more promising direction employs user authentication and, in open communities, distributed reputations and blacklisting. Although we outlined a distributed blacklisting scheme in Section 6, much more work remains to be done in the counter-measure area.

DHTs are particularly vulnerable to poisoning attacks against specific titles. Indeed, in a DHT, a small set of nodes is typically responsible for responding to queries about any specific title. Thus, to poison the distributed index for a specific title, the attacker only needs poison a relatively small number of nodes. Furthermore, the attacker can insert its own node into the DHT, fixing the ID of the node so that queries for the targeted title are directed to it. Finally, if special care is not taken, index poisoning can be used to to turn a DHT into an engine for a massive DDoS attack.

**Acknowledgment:** We would like to thank Andromahe Zografos and Rakesh Kumar for their useful comments throughout this research. We also acknowledge NSF grant NSF 0325777 and the Polytechnic University Center for Advanced Technologies in Telecommunications.

## REFERENCES

[1] CacheLogic Research: The True Picture of P2P File Sharing, http://www.cachelogic.com/research/
[2] J. Liang, R. Kumar, Y. Xi, K.W. Ross. Pollution in P2P File Sharing Systems, *IEEE INFOCOM 2005*, Miami, March 2005
[3] J. Liang, R. Kumar, K.W. Ross, The FastTrack Overlay: A Measurement Study, to appear in *Computer Networks*
[4] M. Ripeanu, I. Foster, and A. Iamnitchi,"Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design," *IEEE Internet Computing Journal*, vol. 6, no. 1, 2002.
[5] D. Stutzbach, R. Rejaie. Characterizating Today's Gnutella Topology, *submitted*.
[6] Apple iTunes Top 100, http://www.apple.com/itunes/
[7] FFmpeg project, http://sourceforge.net/projects/ffmpeg/
[8] S. Sen, J. Wang. Analyzing Peer-to-Peer Traffic Across Large Networks, *ACM/IEEE Transactions on Networking, Vol. 12, No. 2, April 2004*
[9] S. Saroiu, P. K. Gummadi, S. D. Gribble, A Measurement Study of Peer-to-Peer File Sharing Systems, *Multimedia Computing and Networking (MMCN'02)*, San Jose, January 2002
[10] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, J. Zahorjan. Measurement, Modeling, and Analysis of a Peer-to-Peer File-Sharing Workload. *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003*
[11] V. Aggarwal, S. Bender, A. Feldmann, A. Wichmann. Methodology for Estimating Network Distances of Gnutella Neighbors. *Proceedings of the Workshop on Algorithms and Protocols for Efficient Peer-to-Peer Applications at Informatik, 2004*
[12] F. Le Fessant, S. Handurukande, A.-M. Kermarrec, L. Massouli. Clustering in Peer-to-Peer File Sharing Workloads. *IPTPS'04*
[13] N. Christin, A. S. Weigend, and J. Chuang. Content Availability, Pollution and Poisoning in Peer-to-Peer File Sharing Networks. *Proceedings of the Sixth ACM Conference on Electronic Commerce (EC'05)* Vancouver, BC, Canada. June 2005.
[14] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer Information System Based on the XOR Metric, *IPTPS'02*, Cambridge, MA, USA, March 2002.
[15] D. Dumitriu, E. Knightly, A. Kuzmanovic, I. Stoica, and W. Zwaenepoel. Denial-of-Service Resilience in Peer-to-Peer File-Sharing Systems, *Proceedings of ACM SIGMETRICS 2005*, Banff, Canada, June 2005.
[16] S. Guha and P. Francis. Characterization and Measurement of TCP Traversal through NATs and Firewalls, *submitted*
[17] R. Bhagwan, S. Savage, G. M. Voelker. Understanding Availability *2nd International Workshop on Peer-to-peer systems, February 2003*
[18] Sepandar D. Kamvar, The EigenTrust Algorithm for Reputation Management in P2P Networks, *WWW-2003*.
[19] Slyck news, http://www.slyck.com/news.php?story=623, 2004.
[20] Edonkey link list, http://www.freereactor.com/list.php, 2005.
[21] Magnet Project, http://magnet-uri.sourceforge.net/, 2005.
[22] Slashdot news, http://yro.slashdot.org/yro/04/12/19/1712258.shtml ?tid=95&tid=123, 2004.
[23] IP to orgnization mapping, www.banlist.org
[24] Bluetack website, http://www.bluetack.co.uk/index.php.
[25] K. Walsh and E. Sirer, Fighting Peer-to-Peer SPAM and Decoys with Object Reputation, *P2PECON Workshop*, Philadelphia, Pennsylvania, August 2005.