Efficient Tagging of Remote Peers During Child Pornography Investigations

Marc Liberatore, Brian Neil Levine, Clay Shields, and Brian Lynn

Abstract—Measurements of the Internet for law enforcement purposes must be forensically valid. We examine the problems inherent in using various network- and application-level identifiers in the context of forensic measurement, as exemplified in the policing of peer-to-peer file sharing networks for sexually exploitative imagery of children. First, we present a one-year measurement performed in the law enforcement context. Our proposed tagging method offers remote machines application- or system-level data that is valid, but which covertly has meaning to investigators. These tags, when recovered, allow investigators to link network observations with physical evidence in a legal, forensically strong, and valid manner. We present a detailed model and analysis of our method, show how tagging can be used in several specific applications, discuss the general applicability of our method, and detail why the tags are strong evidence of criminal intent and participation in a crime. We then describe the tagging mechanisms that have we implemented using the eMule file sharing client.

Index Terms—Digital forensics, peer-to-peer networking, child pornography, computer crime.

1 Introduction

The most popular resource for the criminal acquisition and distribution of images and video of child pornography is peer-to-peer (p2p) networks, including BitTorrent, eMule, and Gnutella[1]. Law enforcement (LE) have both an easy and difficult time policing these networks. On the one hand, it is easy to identify millions of IP addresses trafficking in known child pornography (CP), as we demonstrate in Section 3. On the other hand, this success falls short in several ways. IP addresses and application identifiers are the foundation of all current criminal network investigations, yet IP addresses do not distinguish multiple physical machines behind a NAT box. Similarly, it is difficult to link the historical activities of a single mobile user moving among many IP addresses. NAT and mobile users represent growing trends.

The value of evidence is the critical difference between forensics and related security research in incident response and privacy; moreover, methods and legal procedures for collecting data differentiate network forensics from simple network measurement [2]. Making guesses or inferences may be suitable for discovering the limits of privacy or advancing incident response, and it may generate an investigative lead, but on its own it will not advance a legal case. Strengthening techniques used in network-based criminal investigations begins to address concerns raise in a report by the National Academy of Sciences [3] calling for a scientific overhaul of forensics, including digital forensics.

In this paper, an expanded version of our preliminary work [4], we introduce new techniques that draw a bright line between the measurement or surveillance of these networks and collection of forensically valid evidence from them. Validating the evidence collected during a network investigation is difficult because remote users do not necessarily maintain a unique and unmodifiable identifier that can be recovered upon seizure of their machine with a warrant. We propose a novel method of subtly tagging a remote computer over the network to create such an identifier. Our approach is an advance over previous methods of gathering information about a remote computer that rely on statistical characterizations, including clock skew [5] or radiometrics [6]. These past characterizations vary with environmental factors such as temperature or attack [7], leading to both false positives and false negatives, and crucially, lack the ability to link together sequential observation by independent observers. Moreover, we detail why our approach, which is akin to recording the serial numbers on bills, is legally sound.

For this work, we built a system to gather evidence of possession of child pornography on a p2p network. It is in use by law enforcement in all 50 U.S. states, specifically trained to use the software, who then provided us with data for almost a full year. To date, the system has been used to obtain thousands of search warrants. We characterize these measurements in order to motivate the future use of our tagging techniques. If tags were found on a machine during a forensic exam, it would be strong evidence that the machine corresponds to observations of a peer made over the network. Unlike methods of statistical characterization, our method has very strong

- *M. Liberatore, B.N. Levine, and B. Lynn are with the School of Computer Science at the University of Massachusetts Amherst.*
  E-mail: liberato,brian,blynn@cs.umass.edu
- *C. Shields is with the Department of Computer Science at Georgetown University.*
  E-mail: clay@cs.georgetown.edu

1. Past studies have found that 28% of possessors of child pornography had images of children younger than 3 years old; and that 16% of investigations of CP possession ended with discovery of persons who directly victimized children [1].

privacy properties: the results can be recovered by investigators only after a search warrant is obtained from a judge. Tags observed by third parties are meaningless. Our careful analysis demonstrates that false positive probabilities can be driven to near zero. The tradeoff is the challenge to make sure tags offered to a target are retained, to be later discovered during an examination.

Specifically, we make several contributions:

- We present nearly one year of investigations into Internet crime, performed with law enforcement. We show that enumerating traffickers of child pornography on p2p networks is simple, and that such trafficking is unfortunately common, with millions of distinct IP addresses participating.
- We analyze the strength of digital evidence relied on by investigators in these crimes, demonstrating that these techniques on their own are insufficient beyond the standard of probable cause for stationary IP addresses. Moreover, such techniques are insufficient for demonstrating intent and do not work well for mobile users.
- Based on the study, we propose a novel method of strengthening network investigations of criminal activity called *tagging*. We analyze its design and demonstrate that the chances of false positives can be made insignificant with relatively low overhead. We also present details of tagging using fountain codes, which have significantly higher recovery rates when deletion of stored tags is higher than 5%.
- Finally, we will show how these tags can be used in several specific applications (including BitTorrent, eMule, and DNS), discuss the general applicability of our method, and detail why the tags are strong evidence of intent and participation in a crime.

Several additions appear in this version of our work: the empirical analysis has been extended from 5 months of data collection on Gnutella to 12 months; all details on tagging based on fountain codes are new; and all details of our eMule implementation are new.

2 Problem and Attacker Model

In this section, we present the motivating problem for our work: network investigations of criminal activity, and forensic validation of the evidence of such crimes. We discuss the investigative process, the legal limitations upon it, and the problem that forensic validation poses. We also present the relevant attacker models.

2.1 Problem Statement

When investigating Internet crimes such as trafficking in child pornography on p2p networks, the general approach of law enforcement is as follows. An investigator issues queries for likely child pornography and gathers results. Some results are chosen for further investigation, often based on likely jurisdiction, and the investigator uses an administrative subpoena to compel an ISP to reveal a physical location that corresponds to the likely source of network traffic that provided the query results. Under a warrant, the location is searched, any computer systems and media are seized, and the media are examined for evidence of the possession or distribution of CP. We describe the various legal restrictions that US investigators operate under in Section 2.2; these restrictions influence our design decisions.

Our interest lies in effectively identifying the correct end system. In particular, can investigators strongly link network measurements with user behavior and intent? Our goals are twofold: First, we evaluate the quality of the procedures currently used to perform these measurements in Section 3.

Thus, our second goal is to improve the quality of evidence and the range of tools available to investigators. In particular, we propose the use of *tagging*. The general mechanism of tagging is to insert bit patterns that are unique to each observation, which we call *tags*, which are then offered to a target during the course of the network-based investigation. These tags can later be recovered from the storage media following a legal seizure, not unlike bills with known serial numbers might be recovered after an undercover transaction involving stolen property or illegal drugs. The tags can then be used to both link the observations with the media, and to show a pattern of behavior, and thus intent, on the part of the suspect.

2.2 Legal and Practical Issues

Data collected during a network investigation can be used for two distinct, dependent purposes. First, measurements can establish the *identity* of a suspect. By identity, we mean a network or application identifier that can ultimately be linked to an individual at a given time and place. Second, measurements can be used to establish *intent* to commit a crime: a user might accidentally download a single CP file, but if they have a large and growing collection over the course of months, it is highly unlikely to be accidental. Discovering *intent* requires consistency of identifiers over time, a property that we observe does not always hold.

There are three key considerations for law enforcement conducting network investigations that relate to, but differ from, those of the typical disinterested researcher:

1. Evidentiary standards: Information that does not meet an evidentiary standard of either *probable cause* for warrants or *beyond a reasonable doubt* for convictions is merely *reasonable suspicion* (a lead), and it is of lesser value. Information collected about a target that comes from a third party (e.g., from one peer about another peer) is not as strong as evidence that was observed directly about a peer (e.g., from the peer by an investigator). This distinction between leads and evidence is roughly analogous to the difference between observation studies used to generate hypotheses, and controlled studies used to test them. Investigators rely upon both evidence that

was publicly shared and evidence acquired through a magistrate-approved search warrant or other valid legal procedures. By design, our technique leaves tags that are recoverable only via a search warrant.

Generally, investigators use network identifiers, such as IP addresses, only for obtaining a search warrant. IP addresses are generally regarded as meeting only the standard of probable cause — good enough for a search, but not convincing enough for prosecution on their own. IP addresses and application-level IDs can vary significantly over time, so skepticism is warranted.

2. Intent: Many crimes include intent as a requirement for conviction. Possession of CP is not illegal when unintentional, e.g., if unknowingly held in a spam folder. Among other indicia [8], multiple attempts to download CP, a growing collection, or the presence of organized archives can demonstrate intent. Our techniques can be used to demonstrate intent in these cases.

3. Public-use technology only: *Kyllo v. U.S.*, 533 U.S. 27 (2001) established some parameters regarding the use of technology to collect public information. In *U.S. v. Gabel*, 2010 WL 3927697, it was ruled that software designed to log public sharing of CP on p2p networks does not violate parameters and rules established by the *Kyllo* decision. Similarly, our proposal works with unmodified network protocols that are in general public use. We also note that recording serial numbers of bills stored in a cash register before they are given or taken by others is a technique unchallenged in courts.

Finally, we note that in this paper we define *forensically valid* techniques based on the standards set by *Daubert v. Merrell Dow Pharma.* 509 U.S. 579 (1993): they have a known error rate, are based on testable hypotheses, are based on accepted scientific methods, and are peer reviewed.

2.3 Attacker Models and Assumptions

We have two actors in our scenario. We define the **investigator's attacker model** as follows: An investigator of a given p2p system: *(i)* seeks to identify users of the protocol in possession of, or distributing, child pornography — typically, an IP address within their jurisdiction is the endpoint of the network investigation; *(ii)* must work within the protocol, and cannot rely upon criminal activity or privilege escalation to gather evidence; *(iii)* can consider indirect evidence to generate leads, but must have direct evidence to succeed (i.e., seeks a direct network-level connection to a remote user's system). A **criminal's attacker model** and goals are markedly different. A criminal: *(i)* will actively attempt to acquire new CP; *(ii)* can redistribute and advertise possession of CP; *(iii)* can actively manipulate the protocol, violate laws, or engage in anti-forensics to hide their activities. Clearly, a criminal actively attempting to hide their trail will be harder to catch; we discuss the likelihood and impact of such attempts in Section 4.2.

3 Empirical Study of Crime and Identifiers on P2P Networks

Our goal in this section is to demonstrate the lower evidentiary value of IP addresses and application-level IDs when used on their own in network investigations. This lower value is the result of widespread use of DHCP, mobile networking, and the presence of botnets, and we provide some quantification of this problem.

3.1 Collection Methodology

Our empirical results are based on our one-year measurement study of child pornography (CP) file sharing on p2p networks. Our data was collected using a tool we wrote for monitoring and investigating sharing of child pornography on Gnutella networks [9]. As a consequence of our efforts, our tool *RoundUp* has been adopted as a standard for p2p investigations by the US Internet Crimes Against Children (ICAC) Task Force [10]. ICAC is a collection of law enforcement agencies from all 50 states. Data from almost 600 participating detectives' actual investigations of CP trafficking on Gnutella were stored in a centralized system under police control; investigations were not automated. We analyzed anonymized data.

From 10/5/2009 to 9/30/2010, LE using RoundUp collected measurements of 4.39 million IP addresses using 1.35 million GUIDs publicly sharing CP. A GUID is Gnutella's application-level identifier that is chosen at random during installation, and changeable thereafter. In all, 149,302 distinct CP files were observed at least once on the Gnutella network. These CP files were checked manually at least once by law enforcement, and we identify multiple instances by hash value. The records in the database exist due to the particulars of the Gnutella protocol and the efforts of our LE partners. Specifically, records are either the result of a Gnutella *search* for filenames matching CP-related keywords, a direct TCP connection to a remote peer and a *browse* list of their shared files, or *swarming* information from a remote peer that indicates that a third peer has also been sharing the same file (identified by SHA-1). In all cases, these records included remote IP, GUID, software and version, filename, file size, and SHA-1 hash value, and were stored with a timestamp. We used MaxMind, Inc.'s IP-based geolocation service at the time of measurement to place IPs in a city. The database stored information only about peers that shared known CP.

While Gnutella is not the most popular p2p program, our statistics show it is popular with CP file sharers. We also characterized BitTorrent measurement data collected by Menasche et al. [11]. While our study focused on CP, the data collected by Menasche et al. measured non-contraband content on BitTorrent. Their study measured torrent activity between 8/2008 and 3/2009. Their work includes the details of their measurements, but our focus is on records indicating the IP address and BitTorrent PeerID of participants sharing pieces of torrents. Measurement of these torrents and the peers was performed using

PlanetLab-based measurement proxies that gathered information from trackers. For this dataset, all MaxMind queries were performed by us on one day in spring 2010. Our conclusions about evidence on Gnutella GUIDs and IP addresses are validated by observing the same results for BitTorrent PeerIDs and IP addresses, as we describe below. We begin with a summary of the success and limitations of the current investigative approach.

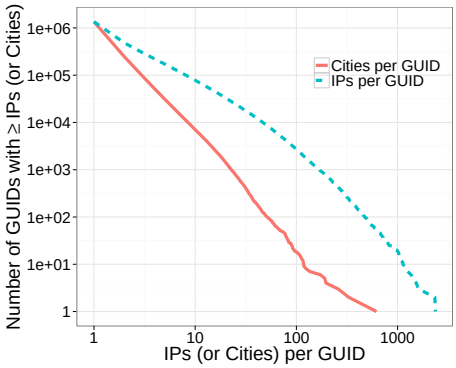
3.2 The Current Investigative Approach

As we stated in Section 2.1, data collected during network investigations are used only as a stepping stone to obtain legal authority to search a physical location for evidence of a crime.

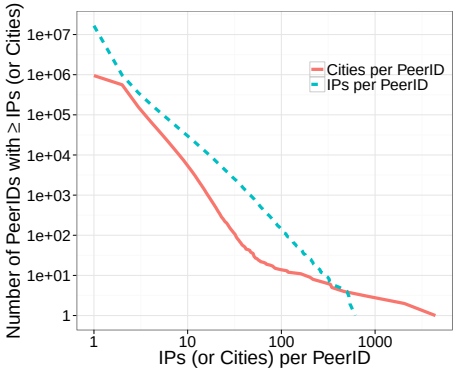
Success of existing approach. In the course of a physical search, storage media are examined for CP and evidence of intent, such as cached search terms. The presence of this evidence is used to create a case for criminal possession of CP. This methodology has been used successfully in thousands of investigations in the U.S.

Limitations of existing approach. There are three key limitations of the current approach. *(i)* When network investigations lead to search warrants, it is evidence found from the search that is often used as the basis for criminal prosecution. In fact, there might be no connection between what is observed on the network and what is found in the search if, among other reasons, users delete or encrypt files or install new client software. *(ii)* Positively identifying a seized machine as the same one that was investigated remotely might be a challenge. Circumstances such as network address translation, DHCP lease times, and mobile interfaces can cause a mismatch. Similarly, many file sharing applications do not provide a stable unique identifier for the user. For example, BitTorrent does not require fixed PeerIDs, and Gnutella does not ensure each client's self-assigned Globally Unique ID is, in fact, globally unique. *(iii)* Intent is a critical part of the definition of criminal CP possession and distribution. Intent can be demonstrated legally in several ways [8]. Unfortunately, one form of evidence of intent on p2p networks — sharing on the network over a long period of time — cannot always be demonstrated easily in court. An even greater challenge is to definitively show that the same person is responsible for using multiple GUIDs or multiple IPs over time, particularly over open wireless access points.

It can be challenging to find the evidence of a crime: The subject of investigation might hide it within the system with encryption or steganography, or might keep the material on a removable storage device that is physically concealed. In cases where the investigator does not locate the material that was seen as being available, it may not be clear whether the wrong system was seized or if the material simply hasn't been discovered. A reliable indication that the correct system was seized, as we propose in the Section 4, can help resolve this dilemma and the others above.



(a) Gnutella



(b) BitTorrent

Fig. 1: *For a given lower bound on the number of IP addresses (or cities), this plot shows the number of application-level identifiers observed to meet that bound. All 1.3 million Gnutella GUIDs were observed at one* or more *IPs. About 78,000 were observed at 10 or more IPs.*

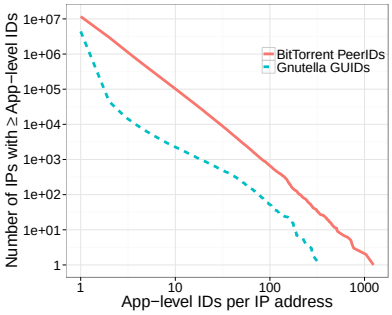
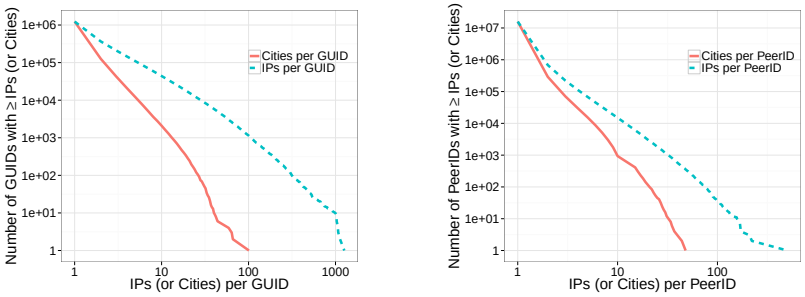


Fig. 4: *For a given lower bound on the number of application-level IDs, this plot shows the number of IPs observed to meet that bound. For example, in the BitTorrent data, all 11.7 million observed IPs were observed using one* or more *PeerIDs, while about 102,000 were observed using 10 or more PeerIDs.*

3.3 Identity and Intent in P2P Networks

Fig. 1 demonstrates how application IDs can fail as a unique identifier. The figure plots the number of IP addresses associated with each ID in the data. For Gnutella, this consists of GUIDs that have been identified as trafficking in child pornography. In our data, about 78,000 GUIDs were each associated with 10 or more IP addresses. A separate line plots the same data by geographic location, with 7,031 GUIDs present in 10 or



(a) Gnutella **(b)** BitTorrent

Fig. 2: *A subset of the data from Fig. 1, this plot shows the minimum number of distinct cities or IPs associated with a given application identifier,* limited to the IDs that were observed only in a single small geographic region.

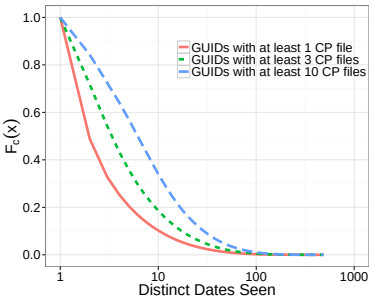


Fig. 3: *For a given lower bound on the number of days a Gnutella peer was seen online, this plot shows the fraction of peers as distinguished by GUID observed to meet that bound (the CCDF).*

more cities. BitTorrent is similar. E.g., there are 5,400 PeerIDs that map to IPs found in more than 10 cities.

One particular GUID was observed in 329 cities around the world using 398 IP addresses. We found this GUID was sharing exactly one file. This file (identified by hash value) was found throughout the network with many different filenames. We assert this GUID is actually a botnet that responds to queries for any term `x` on the network with `x.mpg`, always sharing the same malware content. The existence of this GUID shows a potential difficulty in assuming that GUIDs are unique identifiers for corroborating an investigation with a seized machine, as this GUID appears to be shared by many users. These observations point to a weakness in such IDs that may skew all data points collected, but in a non-obvious way. Within a legal context, as compared to a network measurement study, the implications are more serious: a GUID observed in 329 international cities is not valuable as evidence.

Another problem is posed by mobile users. Fig. 2 isolates IDs that report from 2 or more IP addresses all located one state or region of a country. Since these IDs have IP addresses that map to only one geographic area, we assert that it is most likely one real user, as botnets and misconfigurations are unlikely to be contained to a geographical area. It is unclear to us if these users are sharing their ID with friends, or are accessing diverse open WiFi [12], [13]. This data suggests either that these identifiers are weak, or that users actively move around to avoid detection; either motivates our tagging solution.

Fig. 4 demonstrates the related problem of relying on a peer's IP address as a unique identifier of a specific computer. The figure plots the number of IDs observed per IP address. For example, 6,239 IP addresses were each linked to at least 5 different GUIDs in our database. It is not clear to us whether the GUIDs represent five or more different users behind one NAT box or if one user is responsible for all activity from that IP using five or more GUIDs. In BitTorrent, this problem is worse, with 426,425 IPs using at least five PeerIDs in our study. Since PeerIDs can be generated per torrent, it is almost impossible to

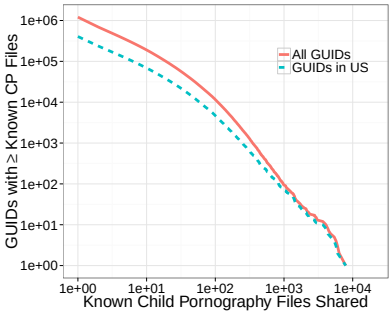


Fig. 5: *For a given lower bound on the number of known CP files shared by a peer, the number of GUIDs observed to meet that bound. Across all observations, all 1.2 million GUIDs were observed sharing at least one files, while about 530,000 were seen sharing 10 or more files. All peers observed were sharing at least one such item due to LE methodology.*

link the download of a torrent with a specific installation on a computer, or even a specific user on a computer.[2]

Fig. 5 shows that GUIDs originating in the US are only about 33.5% of all traffickers. Over 530,000 GUIDs worldwide had 10 or more CP files, and most shared a single file; the metric underscores the scope of the problem as it represents only *known* CP. The users corresponding to these GUIDs often share as-yet-unknown CP or files that have yet to be manually checked by LE; have archived large collections that aren't being shared but found upon execution of a search warrant; or turn out to be contact offenders. Fig. 3 demonstrates that many users are observed repeatedly over long periods of time. The figure also shows that users with larger collections (some have thousands of files) tend to stay on the network longer, an indicia of intent. Unfortunately, with current techniques it is unknown if some of the GUIDs observed once in these two figures are actually the same user.

2. These data were collected before μTorrent, a popular BitTorrent client, began randomizing PeerIDs per-download by default.

4 Remote Device Tagging

Forensic measurements that attempt to tie observations together using information provided by an application-level protocol face a challenging problem. As we demonstrated in the previous section, it is already the case that not all connections are one-to-one between peer and IP address. Cellular providers make use of NATs and a small pool of shared address that are re-assigned frequently, and they are an increasingly common, high-bandwidth choice for network access. Furthermore, users are often in possession of many devices in one home, and investigation of every device during execution of search warrant is sometimes impractical. In short, remote network identifiers increasingly lack uniqueness and consistency.

In this section, we propose a novel mechanism to offer a tag to a remote device that is under investigation. We begin by presenting the tagging process and follow with a discussion and analytical model of the process. We separately discuss tagging for systems where deletion is unlikely and systems where deletion is likely. In the next section, we show several tagging opportunities that exist in eMule, BitTorrent, and DNS.

4.1 The Tagging Process

We propose the tagging of remote machines by investigators, to leave a record of an observation on the remote machine for later recovery during warranted search. We envision the general process in three steps as follows.

First, investigators discover a *vector* for tags: we define a vector as a set of bits embedded in a protocol that can be set within the bounds of the protocol by the investigator and offered to a remote machine under investigation. Further, these bits or some function of them must be stored by the remote machine on non-volatile media. For example, as detailed in Section 5, BitTorrent peers will ask each other their application name and peer ID, and there are minimal restrictions on these values. These values may be stored in a file at the target and can function as tags to uniquely identify the remote machine. No unauthorized access to the target's machine is required; tags are offered in the normal function of a system.

Second, when directly connecting to a remote machine during an investigation, investigators use an appropriate vector to tag the machine. Tags are selected in such a way that their meaning is not obvious and to minimize the likelihood of collision. The investigator records the tags used to so that they can be validated when recovered. One method of selecting tags is to take a hash value of text representing specific details of the investigation. This hash can be provided to commit the investigator to one or more values ahead of the search. The root of a Merkle tree of all hash values used for a specific time period is an efficient method of committing to a series of values.

Finally, upon issuance of a warrant, investigators seize a machine and look for known tags on it. These tags may be found in the expected place, or recovery may require more advanced forensic techniques such as file carving. Tags that are recovered from a seized machine validate that it is a specific system that was investigated over the network. Because recovery requires a judicially approved warrant, and because the meaning of the tags is hidden, our approach has robust privacy properties.

There are two ways in which retrieving tags from a machine can fail. False negatives occur when tags that were offered by an investigator are unrecoverable, due to deliberate user action, log rotation, cache eviction, and so on. In these cases, the tags will not be available as evidence. False positives occur when investigators recover tags that they did not actually offer. We examine these problems below.

Tagging methods. Our results are based on two classes of tagging. In both, we address the problem of tagging when space is minimal by offering a large tag of n bits as k smaller subtags. The size of each subtag is n/k bits. The two classes differ as follows.

- **Standard Tagging:** In our standard approach, we offer k subtags to a remote machine, and all k subtags must be recovered to reconstruct a larger tag. We present several methods, and each represents a trade-off between the subtag size and the false negative rate as well as the number of sessions that can be tagged globally. Storing multiple copies of each subtag will reduce the false negative rate (FNR), while increasing the length of each subtag will reduce the false positive rate (FPR).
- **Fountain Code Tagging:** In this approach, we store multiple subtags of size n/k each, and any $\lceil k * 1.05 \rceil$ are sufficient for recovering the larger tag. This approach survives subtag deletion more robustly. The tradeoffs are more complicated because subtag size is a largely separate parameter from the full tag size. As above, storing more subtags on a remote target will reduce the FNR, while increasing the length of each subtag will reduce the FPR. Subtag length has no effect on the number of sessions that can be tagged globally. Instead, increasing the *tag* length increases the total number of sessions that can be tagged globally, but doing so also increases the FPR of individual machines.

4.2 False Negative Rates

Given that in our model we allow the criminal to erase evidence from their own machine, why do we expect our techniques to work at all? There are many reasons. First, unlike most mechanisms in security, most forensic mechanisms are not subject to *catastrophic failure*: even if one person can and does erase evidence of their actions, that does not imply that everyone else will do likewise, nor does it mean that one person can erase evidence for everyone else. Further, it is still worth investigating those that do not erase evidence. In contrast, if there exists a security exploit in Windows, then one user can comprise every Internet-accessible Windows machine.

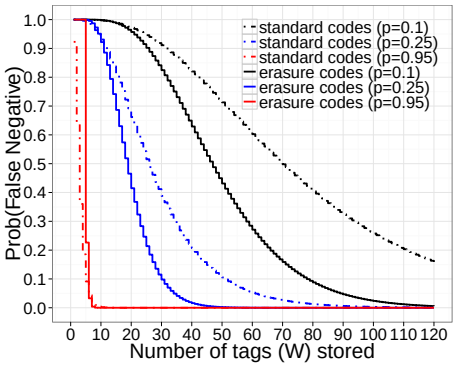


Fig. 6: *A comparison of the recovery probability of standard codes (Eq. 1) analyzed in Section 4.3 to fountain codes (Eq. 2) analyzed in Section 4.4. $k = 4$ in all cases.*

Secondly, these crimes are not always committed by persons with great savvy — the quantitative proof is the measurement we present in Section 3: we identified 4.39 million IPs (using 1.35 million GUIDs) sharing known images of confirmed child pornography. These observations are based on a database of hash values. Anyone can trivially circumvent a hash match for a particular file, *yet millions did not*.

Thirdly, we expect that generally application developers will not help unsavvy criminals nor aim to thwart tagging mechanisms. Since the tags are impossible to trace back to investigators, developers will have no sense of whether they are being used. To be sure they are not open to any tagging and not just the mechanisms we propose here, developers would need to perform a covert channel analysis [14] on their program as well as all OS libraries in use, likely blocking all caching, logging, and similar output from both. Moreover, our methods are designed to tag system mechanisms that improve performance when left enabled (e.g., the DNS cache); we assert that developers are in general more interested in improving performance than protecting traders of child sexual exploitation imagery. Of course, the copyright enforcement actions of trade groups such as the RIAA and MPAA are thwarted actively by some p2p developers. However, the civil torts these groups pursue require only *relevance* for subpoena of a target machine (the 4th Amendment does not apply) and the much lower standard of a *preponderance of evidence* at trial. In short, our tagging techniques would be overkill for supporting civil torts. Again, there is no incentive to remove tagging vectors.

Finally, we show that tagging can leverage fountain codes [15] to achieve successful recovery even the probability of deletion is high. We can encode a full tag as many smaller subtags such that recovery requires only relatively few subtags are found. Because any ordering or subset of subtags are sufficient, fountain codes are a versatile solution for systems where the exact deletion characteristics are unknown ahead of time.

False Negative Model. Let p be the probability that a single subtag is not deleted, and assume that deletion is an i.i.d. process. Assume that we have a full tag that is stored as k subtags, and that we offer W subtags to a remote peer. Each of the k subtags will be stored W/k times, and we require at least one copy of each subtag to recover the full tag. The probability that the full tag cannot be recovered is

$$Pr\{\text{False Neg.}\} = 1 - (1 - (1-p)^{W/k})^k \quad (1)$$

On the other hand, by applying fountain codes [16], we must recover any $\lceil k * 1.05 \rceil$ of the subtags (we explain this value in detail below). The chances that recovery is not possible is

$$Pr\{\text{False Neg.}\} = 1 - \sum_{i=\lceil k*1.05 \rceil}^{W} \binom{W}{i} p^i (1-p)^{W-i} \quad (2)$$

Eqs. 1 and 2 are plotted in Figure 6 for $k = 4$ and several values of p. The fountain code approach has extra overhead but it allows for greater flexibility during recovery. When the chance of recovery of a subtag is lower than about 95%, the overhead of fountain codes begins to pay off. For environments with a high deletion rate, fountain codes are an obvious choice in terms of efficiently lowering false negatives.

4.3 Modeling False Positives of Standard Tags

In this subsection, we analyze the false positive rates of tags offered to and stored by systems that delete data with a low probability. In Section 4.4, we perform the same analysis for a tag scheme that leverages fountain codes to mitigate high deletion rates.

Tags are most useful as evidence after a search warrant has been executed. Therefore, how certain are tags as evidence? In other words, what is their false positive rate? How do different tag sizes, numbers of tags, and tagging schemes interact with the FPR? We probe these questions here.

We define false positives as when a machine that was never tagged appears to be tagged. The analysis we carry out to determine false positives applies equally to the scenario where an adversary places tags on a third-party victim's machine in an attempt to frame the victim, as we assume the attacker doesn't know which tags are used by investigators.

Model assumptions. Assume investigators tag target machines with an n-bit tag each time they are observed on the network (called a *session*), and they keep a database of T entries. The number of entries is exactly the space of all tags that have ever been or will ever be assigned for a distinct taggable event. Each entry will include other essential information about the investigation: the name of the investigator, the date, the tagged IP address, etc. Here we set $T = 2^{\frac{n}{f}}$, and therefore the chance that a recovered tag (that was not offered by investigators) is a false positive is $T/2^n$. We assume $f > 1$, where $1/f$ is the fraction of table space used for a tag, since when $f = 1$ the chance of a false positive is 1. We discuss how f affects performance below, and in fact it is one of two

variables that must be decided ahead of time. We let L be the number of candidate tags that are discovered (i.e., those subtags that are not deleted).

In the analysis below, we assume a log file is recovered from a seized machine and that, unbeknownst to investigators, the machine has never been tagged. In other words, any bit fields that contain apparently valid tags contain bit strings drawn from some unknown distribution, which we assume is independent of the tagging database. For simplicity, we assume that distribution to be uniform.

Large tags. The simple case for tagging is when n is very large; in that case, it is easy to make it improbable that a tag found on seized machine falsely matches a tag in the database. The chances that one or more of L candidate tags match stored values in the database is

$$\begin{aligned} Pr\{\text{False positive}\} &= 1 - Pr\{\text{no matches}\} \\ &= 1 - (1 - \frac{2^{n/f}}{2^n})^L \end{aligned} \quad (3)$$

However, the maximum value of n is not chosen by the investigator; it is a constraint of the tagging vector, as discussed in Section 5. To illustrate, consider when $L = 2000$ and $f = 2$, then the chances of a false positive are less than 10^{-6} when $n = 64$ bits. However, if $n \leq 32$, the chances of a false positive is greater than 3%, which is most likely too high.

Small tags. In some situations, the tag size n is limited and we require a low false positive rate. To overcome this limitation, we have the investigator generate and use many *subtags* per session. Subtags are generated by splitting n-bit tags into k equal-length parts. An investigator then offers subtags to a remote machine k times in a session.

There are two scenarios that we must consider.

- **Case A.** The subtags are stored by the target machine in a *preserved order* that can be recovered.
- **Case B.** The subtags are stored by the target machine in an *unordered set* that prevents ordered recovery. For the unordered case, we offer two solutions: **(B1)** tagging the target k times each session; and **(B2)** allocating space in the subtags to denote the order for recovery of the full tag, which we show below is a better solution.

We derive the false positive rates of the three approaches below and then compare their performance.

Case A: Order Preserved: Concatenated subtags. In this case, we assume subtags are written to a sequential log file, and that investigators can reconstruct the original tag by assembling subtags in the order they are recovered from the log file. It may be that other data is inserted into the target's log between subtags, which can result in false positives. Here, we model the most conservative case: we show the number of false positives given that none of the L candidate subtags were placed by investigators.

When the machine is recovered, the investigator will accept the machine as tagged only if k of the L subtags, when concatenated, appear in the database of T assigned tags. The problem is that investigators must try every combination of $\binom{L}{k}$ found, which creates a large number of potential false positives. Here, $T = 2^{\frac{n}{f}}$ as before (and subtags are n/k-bits long). The false positive rate of the concatenated tags is

$$\begin{aligned} Pr\{\text{False positive}\} &= 1 - Pr\{\text{no full tag matches}\} \\ &\leq 1 - \left(1 - \binom{L}{k}\frac{1}{2^n}\right)^{2^{\frac{n}{f}}} \end{aligned} \quad (4)$$

Note that this is a conservative upper bound, not an equality, as we have elided the inclusion-exclusion terms.

Case B1: Unpreserved Order: Multiple subtags. In this case, the target machine does not store tags in a preserved order. In this solution to the problem, we have investigators offer the machine k subtags that share the same database. Therefore, there is a limit of $T = 2^{\frac{n}{fk}}$ tags that can be assigned.

The false positive rate for the case of k subtags of $\lfloor n/k \rfloor$-bits each is given by a Binomial distribution.

$$\begin{aligned} Pr\{\text{F.P.}\} &= Pr\{k \text{ or more of } L \text{ subtags match}\} \\ &= 1 - \sum_{i=0}^{k-1} \binom{L}{i} (2^{\frac{n}{fk} - \frac{n}{k}})^i (1 - (2^{\frac{n}{fk} - \frac{n}{k}}))^{L-i} \end{aligned} \quad (5)$$

Eq. 5 quantifies the tradeoff between using one tag of n bits and k subtags of one bit each, and all cases in between.

Case B2: Unpreserved Order: Labeled subtags. When the tagged machine stores the subtags in an unordered set, a better solution is to give each subtag its own database, or to otherwise label each subtag to constrain the set of possible databases it could be in. For example, we can reserve $\log_2 k$ bits in each subtag to denote which database it is in. In that case, each subtag has length $r = \lfloor n/k \rfloor - \lceil \log_2 k \rceil$ bits, and we have $T = 2^{rk/f}$ tags possible. To determine the false positive probability, we assume that the L candidate tags are equally divided among the k subtag databases. Therefore, there are $(L/k)^k$ candidate full tags to evaluate; each must not match an assigned value.

The false positive probability is modification of Eq. 3:

$$\begin{aligned} Pr\{\text{F.P.}\} &= 1 - Pr\{\text{none of } (\tfrac{L}{k})^k \text{ subtags match}\} \\ &= 1 - \left(1 - \frac{2^{rk/f}}{2^{rk}}\right)^{(\frac{L}{k})^k} \end{aligned} \quad (6)$$

As we discuss in Section 5.2, specific ranges of IP addresses such as CIDR blocks can be used as tags. In this case, the fixed prefix of the CIDR block serves in place of the $\log_2 k$ bits that would otherwise be reserved to denote the sub-database. The no-cost nature of these prefix bits explains the improved performance of this method in the comparison below.

4.3.1 Sessions Taggable by Each Method

To compare these three methods, we assume the investigator knows the vector-specific length of each subtag and value of L, and has a desired FPR. Her job is to select f and k such that the false positive rate is achieved and the number of sessions that can be tagged is maximized.

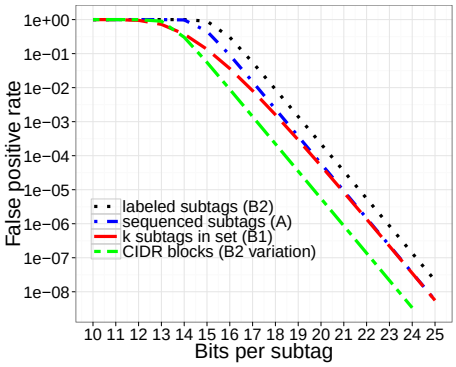


Fig. 7: *The false positive rate as a function of the number of bits per subtag, corresponding to $k = 4$, $f = 3$, and $L = 2000$. The exact values plotted are from Eqs. 4, 5, and 6.*

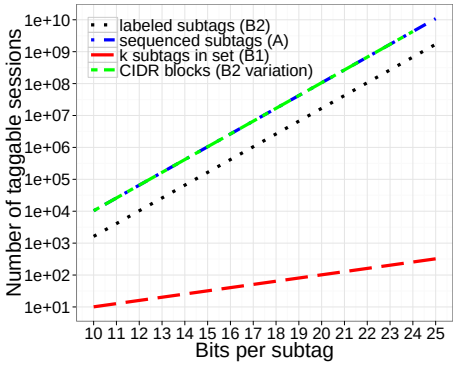


Fig. 8: *The number of sessions that can be tagged as a function of the number of bits per subtag corresponding to $k = 4$, $f = 3$, and $L = 2000$. Each plot is based on the case's formula for T.*

In general, larger values of f and k lower the FPR but reduce the number of sessions that can be tagged. We assume the goal is to minimize k, since if more tags are required to be stored, it is more likely that in general that the tags may be removed by normal operation of the machine.

Accordingly, we evaluate three questions: *(i)* For a desired false positive rate ρ, what is the minimal value of f? *(ii)* How does the number of sessions, T, that can be tagged vary with f? *(iii)* What is an acceptable FPR? We explore these questions in several ways.

First, as quantitative examples, we compute the FPR ρ and number of sessions for each method when $k = 4 and f = 3$, and when $L = 2000$ candidate tags are found based on Eqs. 4, 5, and 6. The false positive rates for the three techniques is shown in Fig. 7. Note the x-axis is the subtag size and not n. In all cases, the probability of a false positive decreases exponentially with the subtag size, and it is less than 10^{-6} when subtags are at least 22 bits. Similarly, Fig. 8 compares the number of sessions that can be tagged by each method, including the CIDR variant of Case B2, based on the definitions of T for each. The number of sessions offered by Case B1 is many orders of magnitude lower than the other solutions.

Second, we address the broader question of how to choose f and T for a given FPR ρ. While Figs. 7 and 8 used the subtag size as the independent variable, here we assume the subtag size is fixed and that L is given. For Case A, a minimal value of f from Eq. 4 is

$$f = n/\log_2\left(\log(1-\rho)/\log\left(1-\frac{\binom{L}{k}}{2^n}\right)\right) \tag{7}$$

Since $T = 2^{n/f}$, we can state T in terms of ρ as

$$T = \log(1-\rho)/\log\left(1-\frac{\binom{L}{k}}{2^n}\right) \tag{8}$$

For Case B2, we have from Eq. 6

$$f = rk/\left(\log_2(1-(1-\rho)^{1/(\frac{L}{k})^k}) + rk\right) \tag{9}$$

and since $T = 2^{rk/f}$, we can state T in terms of ρ as

$$T = (1-(1-\rho)^{1/(\frac{L}{k})^k})2^{rk} \tag{10}$$

Second, we offer the following simple algorithm that allows an investigator to set f and k. (1) Select a desired false positive rate ρ, and set $k = 1$, which determines the subtag length. (2) Calculate f and the maximum number of taggable sessions, T, using either Eqs. 7 and 8 (Case A) or Eqs. 9 and 10 (Case B2). (3) If T is too small, increase k (which implies an increase to n), and goto Step 2.

The question remains as to what false positive rate is appropriate. Historically, law enforcement around the US have made about $A = 2000$ arrests per year. We use this number as an example, however in practice, we can keep one set of tables per available tagging channel. To set ρ, we assume that all the arrests are mistakes, and let $\rho = 0.1A$ such that the expected number of arrests that have a false positive tag is 0.1. Chernoff bounds could be used to ensure that values above the expected mean occur with very low probability.

4.4 Case C: Fountain Codes

In this section, we show how fountain code codes can be used to improve tag recovery while maintaining a low false positive rate for small subtags.

Like the above strategies, a single session's tag is encoded over many entries that are later gathered for reconstruction. Using fountain codes [15]–[17], we write many subtags and *any* k subtags in any order can be used to recover the original n-bit tag. Each subtag is b-bits long and is composed of v bits that encode the original tag and c bits that are necessary for decoding and checksumming. Fountain codes require $k > (\frac{n}{v} + \epsilon)$ entries to recover the original tag. In practice, $k \approx 0.05\frac{n}{v}$ [16]. The investigator therefore aims to offer many more than k entries to the target to ensure identification.

Rateless encoding schemes are more often applied to the transfer of large amounts of data, and the additional metadata needed for decoding is amortized over large packets. Our challenge is that we desire to encode a string of about 32–64 bits and have no ability to amortize overhead costs in each entry. Another challenge introduced by our use of fountain codes is we wish to offer a single machine tags over many sessions, and the entries for each distinct tag must not collide during reconstruction. For simplicity, our scheme is based on LT codes [15].

Encoding. The scheme works as follows. Each time an investigator wishes to offer tags to a remote machine at a specific remote IP, he uses a different session key. The key for session i is S_i. Each key is derived from the a master key S as $S_i = \text{hmac}_S(i)$. The value of S_i is the same for all remote IP addresses but the key used for the session is randomly selected without replacement. The maximum value of i is a parameter that we discuss below, but we expect i to be typically less than 1,000.

Once the key is selected, an n-bit tag is selected. No combination of tag and S_i is repeated, and unlike the schemes above, no part of the tag space is reserved to prevent false positives: the space of all tags is $2^n * max(i)$.

Following the LT code encoding algorithm, an n-bit tag is divided into d blocks of n/d bits each. Each block is indexed from 1 to d. We don't describe the algorithm here, but to encode n, a random selection of between 1 and d blocks are xor'ed together. To enable decoding, each entry should hold the resulting n/d bits and the indices used. For example, if $n = 24$, then with $d = 4$ blocks, the largest entry would store $24/4 + \log_2(4) = 14$ bits. However, because the xor order doesn't matter, there are only 56 combinations of 1 to 4 blocks. Therefore, we store only a 6-bit entry that indexes a lookup table stored by the investigator describing which blocks where xored. (We require only one such table globally.) In general, we require $\lceil n/d \rceil + \lceil \log_2(\sum_{i=1}^{d} \binom{d}{i}) \rceil$ bits. It is easy to show that the optimal choice for $n = 24$ and $n = 32$ is $d = 4$, and for $n = 48$ it is $d = 6$; the resulting total encoding sizes per entry are $10, 12$, and 14 bits, respectively. For the rest of this section, we always use optimal choices of d for any particular value of n.

Raptor codes (a variation on fountain codes) are more resilient to noise channels [18] in general. However, we desire strong, parameterized guarantees for a reconstruction process using very few bits at a time. In our scenario, a noisy entry is one not placed by the investigator. Using a noisy entry to decode the original tag can result in an incorrect result. We ensure that the investigator selects only from true entries by using a checksum for each entry (a well-known solution).

We break each b-bit entry up into two parts:

- **v-bit encoding.** The encoding for an n-bit tag is $v = \lceil n/d \rceil + \lceil \log_2(\sum_{i=1}^{d} \binom{d}{i}) \rceil$
- A **c-bit checksum.** We let $C = checksum(V, S_i)$ be the value of the checksum. We set the length to $c = (b - v)$ bits long. For example, we can set C to be the c least-significant bits of a cryptographic $hmac_{S_i}(V)$ function, keyed with session key S_i.

Decoding. The tag recovery process is as follows. Once a machine is searched, all L recovered entries are candidates for identifying the original tag. For each key ever used, each of the checksums of the L entries are validated. We create a set l_i consisting of the entries that were valid for key S_i. From the set, k are randomly selected and the tag is recovered. If k entries cannot be found, the set is discarded.

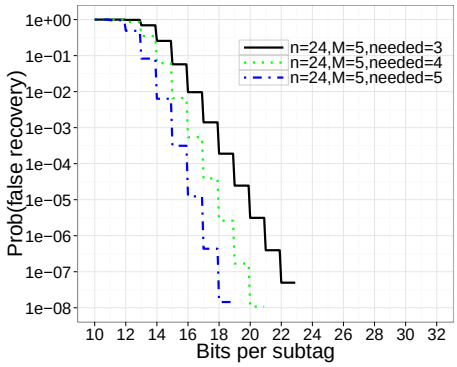


Fig. 9: *For a fixed number of trials, $M = 5$, the false positive rate greatly decreases as the number of trials that need to agree increases. The plot assumes a sufficient number of subtags are present and that $n = 24$ is fixed. Based on Eq. 13 with* $\max(i) = 1$.

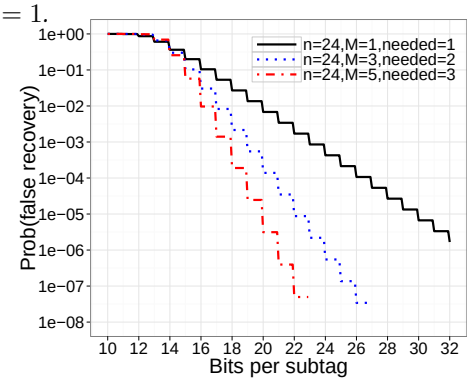


Fig. 10: *As the number of trials increases from $M = 1$ to $M = 5$, the FPR greatly decreases, such that a majority of the trials need to agree upon the same answer. The plot assumes a sufficient number of subtags are present and that $n = 24$ is fixed. Based on Eq. 13 with* $\max(i) = 1$.

False Positives. The derivation of the FPR of fountain code tags must include the probability of a subtag falsely passing a checksum and our use $max(i)$ sessions keys.

The probability of a false checksum match for one subtag (for a given session key i) is

$$Pr\{\text{false pos. subtag}\} = 1/2^c \tag{11}$$

Some subset of the subtags on the machine will have a correct checksum. Given that we need k entries to recreate a tag, if we select $k = \lceil n/d + \epsilon \rceil$ subtags, the chances of falsely reconstructing a full tag is equal to the complement of selecting k true entries.

$$\begin{aligned} \alpha &= Pr\{\text{false pos. tag}\} \\ &= 1 - (1 - 1/2^c)^{\lceil \frac{n}{d} + \epsilon \rceil} \end{aligned} \tag{12}$$

Eq. 12 is an over-estimate: the fountain code algorithm itself is unlikely to converge falsely for random values. However, in the analysis in this section, we conservatively assuming the code will converge for any input.

The FPRs described by Eq. 12 are not very low. However, there are a few methods for improving upon these false positive rates further. Assume that the set of entries with correct checksums is larger than k. To recover the tag, we select k entries uniformly at random M times, and we let the recovered tag be equal to the

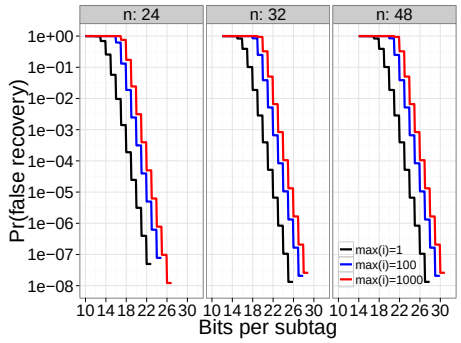


Fig. 11: *When a single machine is tagged multiple times, there are greater chances for a false positive. The chances of one or more false positives when a machine is found with 300 candidate tags, plotted for various values of n (24, 32, and 48), with a majority vote among $M = 5$ trials for each session. These plots are based on Eq. 14.*

majority vote of the M trials. p equals Eq. 12. We seek the probability that M/2 or more trials are falsely recovered.

$$Pr\{\text{false pos. tag}|\ max(i) = 1,\ M \text{ trials}\} = \sum_{j=\lceil M/2 \rceil}^{M} \binom{M}{j} \alpha^j (1-\alpha)^{M-j} \quad (13)$$

Fig. 9 shows the reduced FPR for recovering a tag when $M = 5$ trials are used in the case that $n = 24$ and 3, 4, or 5 trials must agree on the result (Eq. 13). Similarly, Fig. 10 shows the FPR when M is varied from 1, 3, and 5, and the majority vote is scaled accordingly (from 1, 2, and 3, respectively). These results can be compared against the FPR for standard codes, shown in Fig. 7. In short, fountain codes are more flexible than standard codes in terms of testing for a true result.

Lastly, we compute the error introduced by using $max(i)$ session keys: each session key introduces its own chance for false recovery. The probability of one or more false recoveries given $|K|$ sessions keys is equal to the complement of no key resulting in a false recovery:

$$Pr\{\text{false pos. recovery}|\ max(i) \text{ sessions},\ M \text{ trials each}\} = 1 - \left(1 - \sum_{j=\lceil M/2 \rceil}^{M} \binom{M}{j} \alpha^j (1-\alpha)^{M-j}\right)^{max(i)} \quad (14)$$

Fortunately, as Fig. 11 illustrates, although the FPR rate drops linearly with the magnitude of $max(i)$, the values are still small.

4.4.1 Number of taggable sessions

The number of taggable sessions for fountain codes has no trade-off against the subtag length: $T = 2^n i$. It is a straight advantage over standard tagging.

5 Available Tagging Vectors

For tagging to be practicable and of maximum value, several conditions must hold. First, a machine under investigation must be able to receive data from a remote source. Second, that data must be stored in a fashion that can be retrieved by investigators if the machine is physically seized. Third, investigators must be able to work with this data in such a way that it is specific to a single investigation — re-using tags dilutes or nullifies their evidentiary value, violating the assumptions we make in our security analysis. The information will only be recovered when legal authorization by means of a search warrant so allows. Here, we discuss the general manner by which such opportunities can be found and present several specific tagging vectors.

Though several of the approaches that we describe are available in the field, and our eMule tagging implementation has been made available to law enforcement, we do not present specific results in this paper. We, of course, cannot issue search warrants to acquiring the data, and collecting the data would involve an IRB-approved process for obtaining informed consent.

5.1 Discovering Available Vectors

In our experience, offered tags that are stored ultimately reside in one of two places: log (or audit) files, and cache files. Across many systems, log files record both regular and exceptional events. Log files exist for audit and debugging reasons, and typically include many details of triggering events. Cache files exist to improve performance or reliability of systems. For example, most p2p applications store data about remote peers for several purposes: to allow for decentralized operation and bootstrapping; to enable efficient load distribution; or to enable optimizations such as tit-for-tat. The removal of the taggable files would be detrimental to the performance of the system that creates them, and therefore these are the best candidates.

There are several ways in which tagging opportunities can be discovered. We used a manual, ad hoc process to discover the tagging opportunities we describe below. We conjecture that both static and dynamic analysis techniques can be applied to applications to find tagging vectors. It is an interesting problem for future work.

5.2 Specific Tagging Vectors

Here we give several specific examples of vectors that currently exist in the regular functioning of p2p file sharing software. For the eMule file sharing network, we describe our implementation. We present the remaining opportunities as proofs-of-concept, and not as fully developed tagging systems. A limitation of our work is that we do not evaluate the churn of these systems.

5.2.1 Tagging in eMule

eMule utilizes a credit system between pairs of clients. When a client A is able to download portions of a file from another client B, the receiving client A credits the sending client B for the downloaded bytes. If the roles are later reversed and client B attempts to download a

portion of a file from client A, A will give download priority to B over other clients.

eMule clients are uniquely identified by their user hash, a 128-bit value that consists of 14 random bytes that are generated when an eMule client is run for the first time. The user hash is exchanged between clients whenever they establish communication. Because user hashes are openly exchanged, it is easy for a knowledgeable user to impersonate another client by presenting a falsified user hash. To guard against clients impersonating others to improperly receive download priority, secure user identification may be exchanged between clients. The exchange is initiated at the request of either client based on a RSA key-pair.

The secure identity exchange, if used, proceeds as follows. If client A were to request a secure identity exchange of client B, A would include a 32-bit random challenge value in its request to B. For B to prove its identity to A, B must respond by signing with its private key a string consisting of the challenge value and A's public key. A validates the signature using B's public key. The exchange also allows for public keys to be communicated. The algorithm is described in [19], and we have verified it by examining the eMule 0.50c source.

Since credits are persistent between instances of an eMule client, eMule persistently stores a record for each user hash it has verified. The record also includes the remote client's public key, along with the number of bytes transferred. This data is stored in the file `clients.met` on the user's computer.

Our modified eMule client that supports tagging separately logs its own user hash and public key each time it is used by an investigator. When the investigator attempts to make a case for obtaining a warrant, our modified eMule client will always offer a secure identity exchange to the suspect's client. The remote client default to storing the investigator's user hash and public key in its `clients.met` file. When agents seize a system, the `clients.met` file can be recovered. Using tools that we have developed, agents can output `clients.met` into a human-readable format, demonstrating that the investigator's user hash and public key have been stored.

In practice an investigator attempts to show that a single user was able to download an entire file. Therefore only two known values are stored on the remote system: one user hash and one public key. The possibility of a false positive due to collisions remains very small. The user hash consists of 14 random bytes, or 2^{112} possibilities. eMule evicts entries older than 150 days. Even with 24K distinct user hashes added per day, the odds of a collision are approximately 1 in 2^{90}. Earlier versions of eMule used a time-based seed to generate the user hash, but that problem was corrected. The public key size is 2^{384}, making key collisions even more unlikely than user hashes. Furthermore, the user hash and public key are stored as a pair, meaning both values would need to be duplicated to generate a false positive, or 1 in 2^{474}.

Because eMule clients expose their user hash and public key, a suspect might claim that the tags were received from another client impersonating the investigator. However, eMule will not store the user hash and public key unless the source client has demonstrated that it can sign a challenge with its corresponding private key. The investigator can also provide the private key associated with the public key, if necessary.

5.2.2 BitTorrent Peer Caches

In BitTorrent, peers may actively download and upload a torrent for long periods of time. Files are large, and the culture of BitTorrent users is such that continuing to provide upload bandwidth for a torrent after the download is complete is encouraged. To maintain state for these active torrents across application restarts and machine power-offs or suspensions, most BitTorrent clients write relevant information to a cache file. Were this caching disabled the performance of BitTorrent would be worse, as clients would have to re-discover all peers after application restarts. Removing this functionality from the program is a poor defense (see Section 2.3).

The μTorrent client stores, per user-account and per torrent, the IP addresses and ports of remote peers sharing that torrent. (The same client, rebranded, is also the BitTorrent client distributed by BitTorrent, Inc.) These IPs and ports are stored in a file named `resume.dat`, which is a *bencoded* dictionary (associative array). This dictionary is keyed by the each active torrent's infohash. An infohash uniquely identifies the content of a torrent. Each value associated with a torrent's infohash is another dictionary. In this dictionary, the key "peers6" encodes 128-bit IPv6 addresses and 16-bit ports of peers. IPv4 addresses are encoded as backwards-compatible IPv6 addresses. Crucially, these addresses and ports can be provided not just from the tracker, but from other peers through the peer-exchange extensions to the BitTorrent protocol. In our observations, these values need not represent reachable or even valid peers to be entered into the peer cache, presumably because well-behaved BitTorrent clients may ignore incoming connections.

In a similar fashion, Vuze stores, per user-account, a cache file for each active torrent, named `<infohash>.dat`. These bencoded dictionaries contain a key explicitly describing the "tracker_cache", including entries for "tracker_peers" formatted as follows:

```
[  {  'ip': '83.253.52.14',
      'port': 6886,
      'prot': 1,
      'src': 'Tracker'},
   {  'ip': '87.7.101.196',
      'port': 54650,
      'prot': 1,
      'src': 'PeerExchange'}, ... ]
```

Here, even the source of the remote peer is listed, so we can exclude all non-"PeerExchanged" addresses from consideration when recovering tags, eliminating a potential source of false positives.

In both cases, the address and port serve as a tag, though we have observed that the order of the entries

in the peer cache is not preserved in either case. Investigators can use CIDR blocks of address space as tags for IPv4 (e.g., leaving 24-bit subtags for /8 blocks; see Figs. 7 and 8) and equivalent mechanisms in IPv6. Investigators can rent small blocks of address space from many different ISPs to prevent any particular address range from appearing as obviously enforcement-related.

As currently implemented, neither peer caching mechanism requires the investigator to answer BitTorrent protocol messages sent to the addresses that may be stored, so the investigator can offer tags chosen from the IPv6 or v4 address space as appropriate through the peer exchange mechanism. If the implementation were changed to require these remote tags to be valid, the investigator could limit the tags to addresses under their control, running appropriately modified p2p software.

5.2.3 DNS Cache Entries

By default, μTorrent performs reverse DNS lookups on peer IPs once it has connected to them, and Vuze can be configured to do likewise. Many p2p applications include this feature. By performing this lookup, a p2p application likely causes the host OS to cache the returned DNS entry (supplied by LE with unique values). The existence of this entry, and possibly the textual value of the DNS entry itself, serve as a tag.

5.2.4 Vuze log files

Vuze (formerly Azureus) is among the most popular of BitTorrent clients. It creates user-account-specific log files for several purposes, including debugging. The log file named `debug_1.log` (or `debug_2.log`, when rotated), contains at least two obvious candidates for tagging.

The first arises from the evolving nature of the BitTorrent protocol specification. In particular, the format by which BitTorrent peers are identified, the *peerID*, is not fully specified. To aid developers in discovering and naming new BitTorrent implementations, peerIDs in unrecognized formats are saved to the log. As these peerIDs can be arbitrarily chosen 120-bit strings, they present an ideal tagging target. Similarly, when peers give longer-form identifiers, as permitted in both the LibTorrent Extension Protocol and the Azureus Messaging Protocol, unknown or mismatched identifiers are written to the log file. Below is an example real entry, demonstrating a large number of available tagging bits:

```
- [2009] Log File Opened for Vuze 4.2.0.2
- [0406 09:16:22] unknown_client [LTEP]:
 "Unknown KG/2.2.2.0" / "KGet/2.2.2"
 [4B4765742F322E322E32],
 Peer ID: 2D4B47323232302D494775533761494E45425245
- [0406 09:22:14] mismatch_id [LTEP]:
 "BitTorrent SDK 2.0.0.0" / "BitTorrent SDK 2.0"
 [426974546F7272656E742053444B20322E30],
 Peer ID:
 2D4245323030302D27595147314159502764626 2
```

The second tagging opportunity arises from a more subtle side channel present in the log. In particular events for protocol errors can be triggered at timed intervals, such that the inter-event timing forms a side channel. As a simple example, one second between log entries could represent an encoded `0`, and two second delays a `1`. Our past work has shown that channels of this kind are not difficult to implement and that data can be encoded and sent reliably even in the absence of feedback about the time on the receiving system [20]. This is made easier by the fact that BitTorrent files are often large and clients tend to stay on the network for a long period of time [21].

In both cases, these events are logged, but no information is given to the user through the GUI. Both cases clearly allow for tagging, as the log includes information chosen by the remote peer. Further, these tags preserve sequencing information, due to their explicit timestamps.

By default, these log files are rotated when they exceed 256 KB. We performed a two-week measurement experiment using the current version of Vuze (5.0.0). Across five separate instances each downloading ten active torrents, we saw exactly three instances of `unknown_client` messages, and an average of 63KB written to the each instance's log. I.e., the logs do not fill up quickly and false positives are likely to be more rare than the conservative experimental values selected for L in Figs. 7, 8, and 11.

5.2.5 OneSwarm

Prusty et al. [22] describe how OneSwarn peers store the cryptographic keys of neighbors. There we claim a trivial case of tagging: by recovering the keys when a warrant is served, it can be confirmed that a system transferred the data even if the data is no longer on the system.

5.2.6 Other Tagging Opportunities

Depending upon the p2p system and implementation, there are other targets of opportunity. An obvious potential target is the payload data being transmitted by the p2p users. Most systems use some sort of hashing scheme to prevent the deliberate poisoning of exchanges with bad data. Still, if this data is ever written to disk, for example as either as a temporary file or through the VM system, traces of it may persist and be recoverable through standard forensic means. Relative to the tag sizes derived earlier, the immense size of even a relatively small file system allocation unit could provide a definitive tagging opportunity.

6 Related Work

Existing methods of matching traffic to end systems rely heavily on mutable identifiers such as IP addresses. Alternatives use statistical properties that also are both protean and easily falsified. For example, a remote peer's clock skew can be measured based on TCP headers [5], [23], but this value is both affected by temperature and easily falsified by modifying the TCP header. Even radiometric identifiers [6] can be attacked [7]. These statistical measures of a remote peer are not used by practitioners because of these problems. Xie et al. [24] also note that the ephemeral assignment of IP addresses presents accountability problems in network security

and incident response. In contrast, our focus is on law enforcement, who would be unable to subpoena evidence from a myriad ISPs and web sites to deploy that solution. Our approach has a tunable error rate. Their approach has the advantage of distinguishing many hosts at once.

Independently, Shebaro et al. [25] have proposed leaving timing channel fingerprints in the log files of services running on anonymized networks, such as Tor. Their approach makes no assumption about what is logged, other than the existence of timestamps that can later be correlated to known events. In contrast, our approach is to offer the remote service a chance to persistently store known, pre-determined values that can be retrieved when a warrant is issued.

Our user-centric approach differs from past measurement and characterization of deployed p2p networks, e.g., [11], [26]. These past works have focused on performance-related metrics. In contrast, our focus is on forensic investigation and criminal conduct. Finally, our work employs well-known steganographic, watermarking techniques, though we apply them to a novel scenario.

7 Conclusion

Remote, network-based investigation of crimes against children is now the standard for law enforcement [10]. As we demonstrated using one year's worth of data on Gnutella, not all peer-to-Internet connections are one-to-one. As cellular access grows in popularity as a high-bandwidth medium, it will be more common to see IPs re-assigned frequently. Moreover, users are often in possession of many devices in one home, which poses an important triage problem for investigators.

We have also presented a mechanism for more reliable, verifiable identifiers that are stronger evidence in remote investigations, easily meeting the standard of beyond a reasonable doubt. We have provided a strong analytical model demonstrating the applicability of tagging to this problem. We have presented several distinct avenues for tagging across applications and the OS.

Acknowledgements. We thank George Bissias and Ramesh Sitaraman for insightful discussions of the tagging FPR analysis. This work was supported in part by NSF awards CNS-0905349, CNS-1018615 and DUE-0830876 and NIJ award r2008-CE-CX-K005.

References

[1] J. Wolak, D. Finkelhor, and K. J. Mitchell, "Child-Pornography Possessors Arrested in Internet-Related Crimes NJOV Study," Nat. Center for Missing & Exploited Children, Tech. Rep., 2005.

[2] R. Walls, B. N. Levine, M. Liberatore, and C. Shields, "Effective digital forensics research is investigator-centric," in *Proc. USENIX Workshop on Hot Topics in Security (HotSec)*, August 2011.

[3] National Research Council, *Strengthening Forensic Science in the United States*. The National Academies Press, Feb 2009.

[4] M. Liberatore, B. N. Levine, and C. Shields, "Strengthening Forensic Investigations of Child Pornography on P2P Networks," in *Proc. ACM CoNEXT*, November 2010.

[5] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, April-June 2005.

[6] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless Device Identification with Radiometric Signatures," in *Proc. ACM MobiCom*, October 2008, pp. 116–127.

[7] B. Danev, H. Luecken, S. Capkun, and K. El Defrawy, "Attacks on physical-layer identification," in *Proc. ACM Conf on Wireless network security*, 2010, pp. 89–98.

[8] T. Howard, "Don't Cache Out Your Case," *Berkeley Technology Law Journal*, vol. 19, pp. 1157–1575, Fall 2004.

[9] M. Liberatore, R. Erdely, T. Kerle, B. N. Levine, and C. Shields, "Forensic Investigation of Peer-to-Peer File Sharing Networks," in *Proc. DFRWS Annual Digital Forensics Research Conf*, Aug 2010.

[10] U.S. Dept. of Justice, "The National Strategy for Child Exploitation Prevention and Interdiction: A Report to Congress," http://www.justice.gov/psc/docs/natstrategyreport.pdf pages 19–22, Aug 2010.

[11] D. Menasche, A. Rocha, B. Li, D. Towsley, and A. Venkataramani, "Content Availability and Bundling in Swarming Systems," in *ACM CoNext*, 2009, pp. 121–132.

[12] J. Stockwell, "WiFi Turns Internet Into Hideout for Criminals," Wash. Post http://www.washingtonpost.com/wp-dyn/content/article/2007/02/10/AR2007021001457.html, Feb 11 2007.

[13] U.S. Dept. of Justice, "Magic Valley Man Gets 12+ Years For Child Pornography," http://saltlakecity.fbi.gov/dojpressrel/pressrel08/childporn011108.htm, Jan 2008.

[14] V. Gligor, "A guide to understanding covert channel analysis of trusted systems," NCSC, Tech. Rep. NCSC-TG-030, Nov 1993.

[15] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," in *ACM SIGCOMM*, 1998, pp. 56–67.

[16] D. MacKay, "Fountain codes," *IEE Proc.-Commun.*, vol. 152, no. 6, pp. 1062–1068; see also http://www.inference.phy.cam.ac.uk/mackay/DFountain.html, December 2005.

[17] A. Shokrollahi, "Raptor codes," *IEEE/ACM Trans. Netw.*, vol. 14, no. SI, pp. 2551–2567, Jun. 2006.

[18] R. Palanki and J. Yedidia, "Rateless Codes on Noisy Channels," in *Proc. IEEE Intl Symp on Information Theory*, 2004, p. 37.

[19] Monk, "Secure User Identification," http://www.emule-project.net/home/perl/help.cgi?l=1&rm=show_topic&topic_id=135.

[20] S. Cabuk, C. Brodley, and C. Shields, "IP Covert Channel Detection," *ACM Trans. on Info. Sys. Sec.*, vol. 12, no. 4, pp. 1–29, 2009.

[21] M. Izal, G. Urvoy-Keller, E. W. Biersack, P. Felber, A. A. Hamra, and L. Garcés-Erice, "Dissecting BitTorrent: Five Months in a Torrent's Lifetime," in *Proc. ACM PAM*, 2004.

[22] S. Prusty, B. N. Levine, and M. Liberatore, "Forensic Investigation of the OneSwarm Anonymous Filesharing System," in *Proc. ACM CCS*, Oct 2011, pp. 201–214.

[23] S. Murdoch, "Hot or Not: Revealing Hidden Services by their Clock Skew," in *Proc. ACM CCS*, Oct 2006, pp. 27–36.

[24] Y. Xie, F. Yu, and M. Abadi, "De-Anonymizing the Internet Using Unreliable IDs," in *Proc. ACM SIGCOMM*, August, 2009, pp. 75–86.

[25] B. Shebaro, F. Perez-Gonzalez, and J. R. Crandall, "Leaving timing-channel fingerprints in hidden service log files," *Digital Investigation*, vol. 7, Supplement, no. 0, pp. S104 – S113, 2010.

[26] A. Klemm, C. Lindemann, M. K. Vernon, and O. P. Waldhorst, "Characterizing the query behavior in peer-to-peer file sharing systems," in *Proc. ACM IMC*, 2004, pp. 55–67.

Marc Liberatore is a Research Scientist in the School of Computer Science at the University of Massachusetts Amherst. His interests include digital forensics, anonymous communication systems, and p2p systems.

Brian Neil Levine is a Professor in the School of Computer Science at the University of Massachusetts Amherst. His research focuses on digital forensics, mobile networks, privacy, and the Internet.

Clay Shields is a Professor of Computer Science and Director of the Georgetown Institute of Information Assurance at Georgetown University. He conducts research in digital forensics and in network security, and teaches classes in computer and network security, programming, operating systems, and networks.

Brian Lynn joined the School of Computer Science at the University of Massachusetts Amherst in 2007. Prior to that he was a Research Manager at Hewlett-Packard Laboratories in Palo Alto, CA. His interests include digital forensics and computer networks.