Summer November 2014

INFERENCE-BASED FORENSICS FOR EXTRACTING INFORMATION FROM DIVERSE SOURCES

Robert J. Walls
University of Massachusetts Amherst

INFERENCE-BASED FORENSICS FOR EXTRACTING INFORMATION FROM DIVERSE SOURCES

A Dissertation Presented

by

ROBERT J. WALLS

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2014

School of Computer Science

INFERENCE-BASED FORENSICS FOR EXTRACTING INFORMATION FROM DIVERSE SOURCES

A Dissertation Presented

by

ROBERT J. WALLS

Approved as to style and content by:

Brian Neil Levine, Chair

Erik Learned-Miller, Member

Yuriy Brun, Member

Christof Paar, Member

Lori A. Clarke, Chair
School of Computer Science

For Pops

ACKNOWLEDGMENTS

I would first like to thank all of my collaborators, without their help and guidance I would never have been able to get any work done: Marc Liberatore, Brian Lynn, Erik Learned-Miller, Yuriy Brun, Ryan Hurley, Swagatika Prusty, Hamed Soroush, Jeannie Albrecht, Emmanuel Cecchet, Clay Shields, and Janis Wolak. I'd also like to thank Christof Paar for being a part of my committee and Matt Wright for introducing me to research.

Big thanks to Ben Ransford and Shane Clark for their advice and willingness to share their slides with me.

Additionally, I am in debt to Leeanne LeClerc for her unfailing optimism and assistance.

Thanks to my friends in Amherst for their support, late night chats, and midnight food runs: Jimmy, Jason, Tom, Fiona, Matt, Anton, and Monster. I cannot thank Monster enough for all of her care and encouragement. I suspect that she does not like Chipotle as much as she led me to believe.

I owe much gratitude to my family. They raised me and then let me go off and get my degree. All the while they had to put up with the lack of phone calls and visits.

There are far too many people to mention them all here, but I'll keep going anyway: the guys at Tenaska and their willingness to let me pursue my masters while writing code; Sagar for transcribing when I could not write; everyone at the curling club for providing a needed distraction from the stress of graduate school; the Amherst Softball Coyotes for putting up with my competitiveness; and the Crane Hill Cranes for making me go outside in single digit weather. Finally, I'd like to thank Jesse, Yulie, and Chad for sticking with me all of these years.

Most of all, I’d like to thank my advisor, Brian Levine, for putting up with a barrage of fake mustaches and kitten references. His kindness and patience should serve as a guide for any young academic.

This work was supported in part by the National Science Foundation under awards CNS-1018615, CNS-0905349, and DUE-0830876. Additionally this research was supported in part by the Office of Naval Research (N00244-12-1-0057), and the National Institute of Justice (2008-CE-CX-K005). Many thanks to Yahoo for their support.

ABSTRACT

INFERENCE-BASED FORENSICS FOR EXTRACTING INFORMATION FROM DIVERSE SOURCES

SEPTEMBER 2014

ROBERT J. WALLS

B.Sc., UNIVERSITY OF TEXAS AT ARLINGTON

M.Sc., UNIVERSITY OF TEXAS AT ARLINGTON

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Brian Neil Levine

Digital forensics is tasked with the examination and extraction of evidence from a diverse set of devices and information sources. While digital forensics has long been synonymous with file recovery, this label no longer adequately describes the science's role in modern investigations. Spurred by evolving technologies and online crime, law enforcement is shifting the focus of digital forensics from its traditional role in the final stages of an investigation to assisting investigators in the earliest phases — often before a suspect has been identified and a warrant served. Investigators need new forensic techniques to investigate online crimes, such as child pornography trafficking on peer-to-peer networks (p2p), and to extract evidence from new information sources, such as mobile phones.

The traditional approach of developing tools tailored specifically to each source is no longer tenable given the diversity, volume of storage, and introduction rate of new devices and network applications. Instead, we propose the adoption of flexible, inference-based techniques to extract evidence from any format. Such techniques can be readily applied to a wide variety of different evidence sources without requiring significant manual work on the investigator's part. The primary contribution of my dissertation is a set of novel forensic techniques for extracting information from diverse data sources. We frame the evaluation using two different, but increasingly important, forensic scenarios: mobile phone triage and network-based investigations.

Via probabilistic descriptions of typical data structures, and using a classic dynamic programming algorithm, our phone triage techniques are able to identify user information in phones across varied models and manufacturers. We also show how to incorporate feedback from the investigator to improve the usability of extracted information.

For network-based investigations, we quantify and characterize the extent of contraband trafficking on peer-to-peer networks. We suggest various techniques for prioritizing law enforcement's limited resources. We finally investigate techniques that use system logs to generate and then analyze a finite state model of a protocol's implementation. The objective is to infer behavior that an investigator can leverage to further law enforcement objectives.

We evaluate all of our techniques using the real-world legal constraints and restrictions of investigators.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Digital Forensics in Modern Investigations

While digital forensics has long been synonymous with file recovery, this label no longer adequately describes the science's role in modern investigations. Spurred by evolving technologies and online crime, law enforcement is shifting the focus of digital forensics from its traditional role in the final stages of an investigation to assisting investigators even in the earliest phases — often before a suspect has been identified and a warrant served. However, forensic techniques are lagging behind investigation needs. Thousands of people around the world are arrested each year for trafficking child pornography (CP) online, but most remain free because investigators do not have the tools or resources necessary to pursue all cases (see Chapter 5). Further, the continued popularity of mobile devices require investigators to be able to extract information and evidence from impractically many software platforms and devices. In the first six months of 2014, over 70 new smart phones were released[1]. The traditional approach of developing tools tailored specifically to each device and environment is no longer sustainable given the challenges presented by new technologies.

Adding to forensics' woes, the Bureau of Justice Statistics reports that the number of digital evidence requests at publicly-funded forensics labs increased from around $2,800$ in 2002 to an estimated $31,000$ in 2009 [21,116].

Driving these changes is, in large part, the *diversity* of digital information and devices. Law enforcement must examine a wide variety of *data sources* ranging from online services,

[1]Data calculated from `http://www.phonearena.com/new-phones/`.

such as email, to personal devices, such as mobile phones and cameras. Many of these sources store information in formats that must be manually reverse engineered before the investigator can extract the information — a time-consuming and difficult process. Even among mobile phones there are thousands of different models with dozens of proprietary operating systems.

Solving the diversity problem alone is not sufficient, as even a single device can contain an astounding *volume* of information, the vast majority of which is often superfluous to investigators. Personal devices commonly dedicate a significant portion of their storage to operating system data and other artifacts that are immaterial to a criminal investigation. For example, the typical feature phone has hundreds of megabytes of storage of which only a few megabytes are used for user-generated records, such as text messages and address book entries; this problem is even more pronounced for smart phones. Forensic practitioners must quickly identify potential areas of interest while ignoring unneeded data.

The investigation *environment* poses yet another challenge for investigators. Investigations are most successful when the investigator can gather detailed information about a target. Crimes such as CP trafficking force law enforcement to operate in an environment with few physical-world analogs. A suspect may reside anywhere in the world, and the investigator's only link to the target is a myriad of network protocols and servers.

Finally, forensics cannot be separated from the law. All techniques must operate within, and be evaluated under, the real constraints and goals of investigations. This task is difficult to accomplish as jurisprudence often reacts slowly and many issues arise when adapting old laws to new technologies. While the judicial system has had decades to consider the legal implications associated with traditional, single-machine forensics, it struggles to handle new sources of information. Investigators are frequently forced to try and anticipate the potential legal issues of using novel forensic techniques.

1.2 Moving Forensics Forward

Given these challenges, forensic practitioners can no longer rely solely on highly specialized tools built for each potential evidence source. There are simply too many sources, changing too rapidly. Instead, we argue for the adoption of flexible, inference-based techniques that can be readily applied to a wide variety of different evidence sources without requiring significant manual work on the investigator's part. In essence:

> *The traditional forensics approach of developing tools tailored specifically to each new digital evidence source, is no longer tenable. Instead, inference-based techniques offer investigators a scalable means to quickly, accurately, and soundly extract information from diverse data sources, even if the exact underlying format is unknown.*

We develop and evaluate this idea in the context of two diverse, and increasingly important, forensic scenarios: mobile phone triage and network-based investigations. Phone triage is important for quickly extracting details about a suspect's activities and connections to assist an ongoing investigation. Network-based investigations are important for investigating online crimes such as child pornography trafficking on p2p networks. Both of these scenarios embody the challenges described above. Mobile phones, for example, feature a diverse set of hardware and software platforms that often store information in undocumented formats. Network-based investigations involve the interaction of complex and largely hidden processes, some of which are designed to thwart investigation.

The primary contributions of this dissertation are novel forensic techniques that employ inference-based algorithms to extract information from diverse data sources. This dissertation focuses on applying these techniques to the scenarios mentioned above: mobile phone triage and network-based investigations. At a high-level, this dissertation covers the following topics.

1.3 Grounding Forensics Research in the Legal Context

Unlike many other scientific disciplines, digital forensics has largely advanced through the efforts of practitioners. Academic researchers are just now beginning to enter the fray;

however, their contributions are often limited because of their lack of understanding of the legal context in which law enforcement operates. To provide a strong legal framework for this dissertation, we use the lessons detailed in Chapter 2 to ground our research in a detailed and accurate investigator model, thereby, ensuring that each of our proposed techniques is evaluated under the real-world constraints and restrictions of investigators.

1.4 Extracting Information from Mobile Phones

To address the problems of diversity and unknown data formats in mobile phones, we describe and evaluate DEC0DE, a system for inference-based triage (Chapter 3). *Triage* in this context refers to the process of quickly and accurately acquiring evidence, often on-scene. This work includes a method of *block hash filtering* for revealing the most interesting portions of memory within a large store on a phone — efficiently reducing the volume of data that needs to be analyzed. To recover information from the remaining data, DEC0DE employs an efficient and flexible use of probabilistic finite state machines to create a maximum likelihood parse of the phone's memory. We provide extensive empirical evaluation of the system and its ability to perform well on a large variety of phone models.

To address the challenges presented by smart phones, we discuss a post-inference technique, LIFTR, that incorporates investigator feedback to adaptively filter false positives (Chapter 4). Our intuition suggests that a practitioner can quickly verify a result as a true or false positive allowing LIFTR to use the input to more effectively cull the useful results.

1.5 Enhancing Network-based Investigations

Law enforcement has limited resources to investigate and prosecute crimes. Unfortunately, the extent of criminal activity often exceeds police capabilities. In the case of CP trafficking on p2p networks, hundreds of thousands of peers share contraband, but U.S. investigators only have the resources to arrest a few thousand suspects each year. In

Chapter 5, we quantify the extent of the problem on two p2p networks, Gnutella and eMule, and evaluate different techniques for prioritizing targets.

These results are based on data gathered using tools specifically tailored to each network. In Chapter 6, we describe how investigators can wield inference-based techniques to investigate different networks and protocols without needing to create a new tool for each. Our work includes techniques for inferring a model of a protocol's implementation, analyzing the model for investigative opportunities, and subsequently generating concrete message sequences to utilize those opportunities.

1.6 Collaborators

All research activities were conducted under the supervision of Brian Levine. The preliminary work for Chapter 2 was completed in collaboration with Marc Liberatore, Clay Shields, and Brian Levine. Chapter 3 contains research done with Erik Learned-Miller and Brian Levine. Chapter 4 was done in concert with Saksham Varma, and Brian Lynn. Chapter 5 details collaborative efforts with Ryan Hurley, Swagatika Prusty, Hamed Soroush, Jeannie Albrecht, Emmanuel Cecchet, Brian Levine, Marc Liberatore, Brian Lynn, and Janis Wolak. Chapter 6 details efforts in conjunction with Yuriy Brun, Marc Liberatore, and Brian Levine.

CHAPTER 2

EFFECTIVE DIGITAL FORENSICS

Many technical mechanisms across computer security for attribution, identification, and classification are neither sufficient nor necessary for forensically valid digital investigations; yet they are often claimed as useful or necessary. Similarly, when forensic research is evaluated using the viewpoints held by computer security venues, the challenges, constraints, and usefulness of the work is often misjudged. In this chapter, we point out many key aspects of digital forensics with the goal of ensuring that research seeking to advance the discipline will have the highest possible adoption rate by practitioners. We enumerate general legal and practical constraints placed on forensic investigators that set the field apart. We point out the assumptions, often limited or incorrect, made about forensics in past work, and discuss how these assumptions limit the impact of contributions.

The lessons detailed in this chapter form the foundation of the investigator models described in later chapters.

2.1 Principles of Digital Forensics Research

Our observations are based largely on our experience working directly with practitioners [89,146] and advancing work [91,122,152] that operates within forensic principles that we detail within this chapter. At a high level, these principles are as follows.

- Digital forensics is investigator-centric, and unless developed with an understanding of the restrictions that investigators are under, most novel results cannot and will not be adopted. For example, prior to the issuance of a warrant, techniques for criminal investigators cannot conduct a search in violation of a person's *reasonable expectation of privacy*, and evidence gathered otherwise would be suppressed in court.

- The value of a new technique depends in part on its complexity and therefore it must be judged against simpler options available to investigators. Similarly, defenses against investigation should not be evaluated with an assumption that high precision is always needed. For example, civil investigations are often based on simple subpoena and mere demonstration of relevance; sophisticated investigative techniques may be needlessly restrictive or indirect compared to capabilities and information available after subpoena.
- Forensic techniques are most valuable when addressing the most common adversary, not the strongest; there is no correlation between technical savvy and dangerousness to society. It is not possible for one savvy criminal to destroy, hide, or obfuscate the evidence of everyone else. In contrast, security work must consider that one person can leverage a vulnerability to attack every computer that uses the flawed system.
- Finally, forensic investigations seek to find the person responsible rather than stopping at a machine or line of code. Consequently, the scope of forensics is often more broad than that of the traditional security domain. For example, most any policy or law requires consideration of a person's intent, something that is often demonstrated indirectly through an amalgamation of facts and evidence.

Unfortunately, experiences like ours in deploying well-used tools based on novel forensic research are rare. More typically, computer security research aimed towards forensic applications has little or no impact — often because the researchers are poorly acquainted with the real-world problems faced by forensic investigators and the constraints placed on solving them. Similarly, when forensic research is evaluated using the viewpoints held by computer security venues, the challenges, constraints, and usefulness of the work can be misjudged.

Our position is that *security venues should publish forensics research, but these works should be evaluated in the proper context.* When selecting reviewers, ensure they can examine papers using the goals and principles of digital forensics and not just those of computer security. The reviewer should question if the authors are actually looking outside of the computer security problem when claiming an approach is applicable to forensics. Otherwise, the security community risks encouraging low-impact work while rejecting worthwhile solutions to forensic problems.

Adding to the confusion, the technical overlap between security and forensics can falsely color one's view of the latter. For example, packet attribution techniques proposed by security researchers can be useful for determining the source IP address for network-level attacks, but as Clark and Landau [33] point out, such mechanisms are "neither as useful nor as necessary as it would appear" for investigations that require identification of a person rather than a machine. We generalize that statement further: many technical mechanisms across computer security for attribution, identification, and classification are neither sufficient nor necessary for forensically valid digital investigations. Developing a security mechanism for, say, remote identification of a device, and claiming it works for forensics is akin to developing a new cryptographic hash function and claiming it can be applied to many security problems: the claim is easy to make, but the impact is negligible.

In general, digital forensics is concerned with techniques (1) that support or refute a hypothesis that explains a person's violation of law or organizational policy; (2) such that the investigator is limited by a defined set of procedural restrictions for gathering evidence; (3) where the value of evidence is defined by a qualitative context and not only quantitative measure; (4) and where the error rates and procedures of techniques are known and testable.

Researchers have an enormous opportunity for impact by transforming novel research results into techniques that can be used by investigators. The need is great: the National Academy of Sciences (NAS) recently published a report calling for a scientific overhaul of digital forensics [108]. Further, prevalent forensic techniques do not scale and already the demand for forensic examination is much greater than current capacity [54].

In Section 2.2 we begin by enumerating general legal and practical constraints placed on forensic investigators in the context of the U.S. legal system. We then move to briefly describing the wider scope of investigations forensics examiners face. In Section 2.3, we review a set of useful lessons for researchers regarding the applicability of techniques to digital forensics, and highlight recent work in this context. While we focus primarily on investigations in the context of the U.S. legal system, our conclusions are applicable to most other forensic contexts.

2.2 Forensic Investigations

In this section, we detail general models for criminal and civil scenarios, and we describe how investigations are focused on people rather than systems.

In both civil and criminal contexts, digital forensics is concerned with techniques that address the four points stated in the introduction: (1) hypothesis testing, (2) procedural constraints, (3) evidentiary value, and (4) error rates. Practice provides another set of constraints. For example, all methods rely on acquisition of evidence, yet data is commonly destroyed, lost, stolen, or encrypted. The capacity of investigators to take on new cases is limited, and selection is based on many external factors[1]. Even if a case is accepted, there is often too much data to image, store, and process, and an imperfect triage process is frequently necessary [54,104][2].

2.2.1 Criminal Investigations

Criminal investigations take place along two phases, a *pre-warrant* phase, occurring prior to the issuance of a search warrant, and a *post-warrant phase*.

[1]We explore mechanisms for resource allocation in Chapter 5, specifically, with regard to peer selection when investigating contraband trafficking on p2p networks.

[2]Chapters 3 and 4 describe our contributions to mobile phone triage: flexible probabilistic techniques for inferring information.

2.2.1.1 Pre-warrant phase

In the pre-warrant phase, investigators are limited by U.S. law, stemming largely from the Fourth Amendment, that dictates what can acquired before a warrant is in place. The main goal of this phase is to subsequently meet the *probable cause* standard for obtaining a magistrate-issued warrant. This standard is is a qualitative measure often defined as meeting a "fair probability" that further evidence will be found in the location to be searched; see *U.S. v. Sokolow*, 490 U.S. 1 (1989). Probable cause does not require that the evidence is strong enough for conviction, merely that the evidence support a reasonable belief that the suspect committed a crime. The caveat is that the collected evidence must not violate a person's expectation of privacy[3]. During this phase, publicly-observable criminal activity is logged, and then specific targets are selected among available suspects. The choice of target is a balance between *dangerousness* to society and efficiency of case execution.

2.2.1.2 Post-warrant phase

In the post-warrant phase, investigators are limited only by a court-issued warrant. Warrants require *particularity*, which limits the place to be searched; for example, its unlikely to get a warrant for all apartments in a building, nor does having a warrant for one allow investigators to enter another. Warrants also require *specificity* which defines the type of item to be found. In digital contexts, it's hard to violate specificity since any computing device is typically a viable target.

After obtaining a warrant, there are fewer legal restrictions on law enforcement's technical approach[4]. However, in order to obtain a conviction in court, the investigator must

[3]A person's *expectation of privacy* is established using the two-pronged *Katz* test. The first prong asks whether the person subjectively demonstrated an expectation of privacy, and the second prong asks if that expectation is objectively reasonable from the standpoint of society; see *Katz v. U.S.*, 389 U.S. 347 (1967).

[4]In *U.S. v. Comprehensive Drug Testing*, 579 F.3d 989 (9th Cir. 2009) (en banc), the Ninth Circuit imposed *ex ante* restrictions on warrants related to computer searches; however, recent legal scholarship suggests these restrictions are "both constitutionally unauthorized and unwise" [79].

collect a higher standard of evidence. This evidence must help prove *beyond a reasonable doubt* that a suspect committed a crime.

Peer review of digital forensics research must include an analysis of the legal justification required to employ proposed techniques. Techniques that don't require special privileges are applicable to the broadest number of settings and therefore are the most desirable. But such papers should provide the justification that a warrant or other restrictions, such as *Kyllo v. U.S.*, do not apply; see our discussion of *Admissibility* below. If post-warrant capabilities are indeed required, the paper should detail why easier technical solutions aren't available to obtain the same results. For example, it's likely that watermarking the outgoing traffic of a user requires a wiretap; at that point, the criminal investigator will find it just as easy to get a warrant to install a covert key-logger or other device on the target's computer. We return to this point in Section 2.3.

2.2.1.3 Exceptions to the Fourth Amendment

The law underlying realistic investigative models shifts frequently. For example, courts have long held that there are exceptions to the Fourth Amendment's warrant requirement, but these exceptions are not always clear in the case of digital evidence. One exception to the warrant requirement is based on consent by a person to proceed without a warrant. Often the limit of that consent can be unclear, especially when the objects to be searched are digital devices. Events involving the ACLU and Michigan State Police [62] motivate the following example.

Imagine a person is stopped by police for a traffic violation. While a recent supreme court decision holds that the police may not generally, without a warrant, search the contents of a cell phone[5], assume the person gave consent thereby obviating this requirement. Does this consent allow an officer to open the phone and browse through the contents of the address book or call log? Does it also extend as far as to allow the officer to use a special

[5]See Riley v. California 13-132 (2014).

tool to extract and save all of the phone's information, including deleted data? The answers to these questions depend highly on the specific circumstances and the courts are inconsistent on this issue [80].

Search at the U.S. customs border is another exception to the warrant requirement. In 2011, this directly impacted the security community with several high profile border searches of Bradley Manning associates [61]; see also *U.S. v. Howard Cotterman*, 09-10139 (2011).

There are investigators and models that have implicit exceptions to the Fourth Amendment. For example, investigators working under the U.S. national security (FISA) rules may defer a request for a warrant until after a search, and there are analogous positions in other countries. Similarly, rogue investigators can elect to not follow any restrictions, risking that collected evidence is thrown out during trial. We assert that developing new techniques to function under such models is largely wasted effort due to their limited applicability in a civil society.

2.2.1.4 Admissibility and Validity

The exclusionary rule, set forth in *Weeks v. U.S.*, 232 U.S. 383 (1914), in concert with the fruit of the poisonous tree doctrine, set forth in *Silverthorne Lumber Co. v. U.S.*, 251 U.S. 385 (1920), dictates that illegally obtained evidence cannot be used in court, nor can any evidence further found from this illegal evidence be used. This rule places great pressure on law enforcement to follow the standards and procedures put forth by the Fourth Amendment and other legal directives. These rules are intended to ensure that techniques not valid within the rules will not be commonly used by law enforcement in the U.S.

Further, we note that in the pre-warrant stage, law enforcement must take care in acquiring evidence from third parties. In addition to the problems of hearsay, even well-meaning third parties cannot repeatedly gather information for law enforcement. In doing so, the third party is acting *under the color of law*, and any evidence they collect is governed by the same rules as apply to law enforcement, including the need for warrants .

Another relevant ruling for computer scientists is *Kyllo v. U.S.* 533 U.S. 27 (2001), where the court ruled that using a technology that is not in “general public use” to gather evidence pre-warrant is a violation of a person’s expectation of privacy. This exact phrasing is important: source code available publicly on a researcher’s web site might not be considered general public use. With regard to digital forensics, this has sometimes been interpreted by investigators to mean that tools can only use information provided by normal operation of the system being investigated. Recent cases have supported this view, including *U.S. v. Borowy*, 595 F.3d 1045 (9th Cir. 2010) and *U.S. v. Gabel, 2010 WL 3927697*, but the exact extent to which can investigators can exploit a network protocol to gather information remotely is unsettled law.

In order for a forensic investigator’s testimony to be admissible in court it must follow the *Daubert* standard; see *Daubert v. Merrell Dow Pharmaceuticals,* 509 U.S. 579 (1993). According to this standard, the investigator’s conclusions must be based on *scientifically valid* methodology. This means the methods are peer reviewed, based on testable hypotheses, have a known error rate, follow an existing set of standards, and are generally accepted within the scientific community.

2.2.2 Civil Investigations

Civil forensics cases tend to be very different than criminal cases in that the government is not using state force to prove guilt for a crime. Instead, citizens turn to the courts to resolve a dispute. Because of this, the methods for gathering evidence are very different than criminal cases and are often limited by mutual agreement between the litigating parties or by the cost of the investigation. Processing of the acquired evidence is similar to a criminal investigation in that examiners are obtaining evidence that supports or refutes a hypothesis about what users did, saw, or knew in the context of the overall dispute. The Daubert standards apply as well.

In the United States the acquisition of evidence is generally governed by the Federal Rules of Evidence (FRE). Individual states are allowed to develop their own rules of evidence, but most states use the Universal Rules of Evidence which are based on the FRE [86]. The Federal Rules of Evidence were amended in 2006 to recognize the importance of electronic information, and rules were adopted to ensure its availability [130].

Rule 26(a)(1)(A)(ii) [48] requires disclosure of sources of electronically stored evidence without any request from the other party. This implies a duty to preserve potentially useful electronic evidence if it likely to be useful in litigation, and the law allows a judge to sanction parties who fail to do so. Rule 37(f), however, does limit these sanctions if the data was destroyed as part of normal operation of the system. As a result of these rules and to avoid having to produce data if sued, many businesses will limit the data collected or the time they keep it as part of their usual practice.

Rule 26(f) requires parties to meet early on in the litigation process to discuss discovery, including issues surrounding electronic information and the format of production for data. Parties also discuss how to handle privileged documents, such as attorney-client communications, as broad delivery of electronic records can include those as well.

The FRE do recognize that the cost of providing electronic discovery can be high. Parties are therefore only required to produce data that is "reasonably accessible" by Rule(26)(b)(2)(B). Should the other party object and request further discovery, the court will consider what is accessible and what is not. The court can grant access to the additional material but place the costs of production on the objecting party if it chooses.

While this is a brief overview of only some rules of evidence, these provisions lead to differences from criminal investigations. Many times forensic investigators are not brought onto the case until after the early discovery meeting has taken place, resulting in an agreement that doesn't accurately account for sources of evidence that can be produced nor for an efficient production format. While it is possible to revisit production, doing so can be complicated by the need for agreement from all parties; this is difficult in an adversarial proceedings. The investigator may also be limited in what they are allowed to report should they find evidence of additional malfeasance outside the original case. The limitations are placed to protect privileged material or to help parties reach an agreement on what data sources can be examined.

Access to equipment is also different than criminal cases. Rather than seizing equipment immediately, a significant amount of time might lapse between when user activity occurred and evidence is preserved. This often means that evidence can be harder to recover, as normal system activity might have caused older information to be removed and overwritten. Many civil cases involve businesses whose equipment is in regular use and parties are loathe to provide access as they incur a cost for it being down. This is true even when the investigator is examining equipment owned by the hiring party, which often happens in cases of computer misuse, internal fraud, or theft of intellectual property. Finally, investigators may be limited in the depth of the examination performed simply due to a limited budget for discovery of electronic materials.

2.2.3 Investigating People as a Goal

Computer security is centrally concerned with the enforcement or defeat of technically based and clearly delineated computer security policies [15]. Response to computer security failures is typically designed to identify the cause of the failure, which might provide clues to the identity of an attacker. However, intrusion response, which is familiar to security investigators, is but a small subset of digital forensics.

Forensic investigators instead often investigate suspected breaches of organizational policies or laws not reflected in any computer security policy. Indeed, such rules may be impossible to implement in a computer system. A user's intent is often relevant but impossible to determine directly. For example, a user may be allowed to copy a file for work use, but not to sell or otherwise release it. Bradley Manning's alleged theft of documents falls in this category. No existing security mechanism can directly determine intent, but an investigator may be asked to gather evidence about how and why copied files were used. Similarly, possessing photos of children likely does not violate a security policy, but having pictures of children being sexually exploited is illegal. Systems that can generalizably differentiate between the two do not exist. Furthermore, possession requires a demonstration of knowing intent: unexamined images that are unknowingly and unintentional stored in a user's spam folder are not illegal. Howard [69] provides a cogent discussion of indirect evidence of knowing possession.

Researchers must be aware of the focus on people, not just computers. Systems that answer questions about user behavior can be beneficial to forensics investigators, even if no computer security problem is being addressed.

2.2.4 Applicability and Impact

Many proposed forensic techniques are easily thwarted with only limited technical knowledge, but that doesn't lessen their practical effectiveness. While security mechanisms have impact because they can address the worst case, forensic mechanisms have impact because they can address the common case. For example, the most realistic forensic model allows for any individual to erase information from storage; in this scenario, why should we expect new techniques to work at all? Surely criminals will seek to cover their tracks.

For example, investigators commonly identify images of child pornography on p2p networks by hash value. Criminals could easily change just one bit in shared images to escape detection; yet millions do not [89] for several reasons. First, they may not think that they need to make changes to evade investigators, or they may lack the skill to do so. Second, they may not think they'll be caught, and percentage-wise that is largely true. Third, people are interested in sharing content, and in order to do so they give files very descriptive names including "illegal child pornography"; at that point there is no legal reason to flip a bit.

Given that anyone could flip a bit but that millions do not, it is high impact to develop techniques that succeed in the common, rather than worst case. This fact is anathema to computer security researchers, even though, analogously, security systems with known flaws remain useful.

For example, despite power monitoring attacks [46,81,109], most people do not use tamper-proof hardware. Various forms of the Sybil attack [45] succeed and are used against Google [6,13], EBay [28], networks [95,117] yet these systems enjoy great success. The Tor privacy network is architected to provide reasonable performance instead of perfect security against known attacks [44]. Similarly, the banking industry finds it more effective to allow "bad guys to take a cut" [17] than attempting to deploy a system where all attackers are defeated. Further, the TSA admits it cannot defeat all terrorists, and instead simply mitigates risks [65].

Finally, we note that law enforcement are interested in catching the most dangerous people. Similarly, civil investigators are interested in catching the people that have caused the largest damages. In both cases, there is no evidence that such dangerousness is necessarily correlated with technical savvy. Furthermore, it is not possible for one savvy criminal to destroy, hide, or obfuscate the evidence of everyone else; in contrast, security work must consider that one person can leverage a vulnerability to attack every computer that uses the system.

2.3 Lessons to Learn

In this section, we review past security papers from a forensics viewpoint. Some explicitly invoke the concept of forensics; some do so implicitly. We point out the assumptions, often limited or incorrect, made about forensics in these papers, and discuss how these assumptions limit the impact of the contributions. Our goal is not to denigrate others' contributions, but instead to show if and how these contributions fit within the forensic framework we've discussed in previous sections.

We separate our survey into several broad classes, corresponding to lessons learned about forensic practice in the previous sections: 1) Investigations are about people and their activities; 2) Forensic investigators are not all-powerful, and while the legal system can grant impressive powers, they are constrained in many ways; 3) A proposed system that depends on access to data across organizational boundaries may fail — as this access is not often permitted; 4) Proposed systems that expand the view of investigators can be useful, but simply expanding the amount of data collected by itself generally is not.

2.3.1 Problem Exists Between Keyboard and Chair

Forensics investigations of individual's computers arise because of user's actions. Investigators are therefore most interested in people and how they used the system. Security researchers, however, tend to focus on technical aspects of security system failures. Solutions that add additional information about system events [43,83,115] tend not to benefit investigators directly. Instead, researchers can have impact by focusing on mechanisms that support common investigations types, such as theft of intellectual property, violations of organizational misuse policies, and embezzlement [136]. These are common problems, and solutions will have high impact. Work in this area needs to consider the types of users who commit these acts. Most have weak computer skills, so approaches that might be trivially thwarted when used against security experts can be very effective, as we outline in Section 2.2.4. To paraphrase a police maxim, it is useful to catch the stupid ones.

2.3.2 Lines in the Sand

There are many clear lines lawful investigators cannot cross. Wiretapping and analogous inspection of traffic is not permitted without a warrant, nor, presumably are manipulations of Internet traffic beyond normal participation in protocols. Actively watermarking packets, by manipulating their contents or timing [53,93,153,154,160] to defeat anonymity systems crosses this line, and would poison the evidence acquired by a law enforcement. For civil cases, organizations might choose to use these techniques on their internal networks, but are unlikely to provide access, mark traffic, or help recover timing information for external parties. Further, many such techniques work best to confirm a suspicion, as they require both manipulation of traffic at a source, and the observation of traffic at a suspected endpoint. Evidence required for this level of suspicion may rise to the level that would lead to a warrant.

But if an investigator had sufficient evidence for a warrant, a more direct search would likely be preferable — if not, an electronic bug in the computer's audio system almost certainly would. As evidence ultimately supports a court case, recordings of a suspect or copies of emails would be far more useful than a watermark. Systems that assume technically sophisticated attacks, such as inference based on clock skew, temperature or power consumption changes, and acoustic or electromagnetic measurements, or packet size [8,82,90,106,151] are similarly useless to a criminal investigator, due to Kyllo, and obsolete after court order or warrant.

On the flip side, many systems are built to withstand cursory investigation, such as disk encryption systems. It is an unsettled question of law as to whether cryptographic keys are a more analogous to a physical key, which an individual can be compelled to produce in both civil and criminal courts, or to testimony, which in the criminal context is protected by the Fifth Amendment. Failing that, systems focused on key recovery can be forensically valid [64].

Some systems attempt to sidestep this question by providing users with plausible deniability (for example, Rubberhose and its successor TrueCrypt). These systems may be valid responses to overly intrusive governments. But they are inappropriate in more common use cases such as secure corporate record keeping, where allegations of wrongdoing will likely require court-ordered key revelation or else result in being found in contempt of court.

2.3.3 No Keys to the City

Investigators working in different organizations or with different goals may not be willing or able to collaborate with one another. Sharing of data across institutional boundaries is not always feasible or even legal. Wide-scale intrusion or anomaly detection, de-anonymization, or flow attribution systems [26,120,133,135,159] that require a network-wide view or deep packet inspection are akin to a massive surveillance campaign. Monitoring of this breadth could never be lawful for law enforcement without a court order — and even then, is likely too broad in scope. These systems are still useful, outside of the criminal investigation context: ISPs or private organizations could collect this data and use it internally to improve security and performance. They cannot collect it at police request, however, as that would turn these organizations into de facto agents of the law. In many cases, it would also be unlikely that the information would be willingly shared between organizations in civil cases. This is particularly true when it might expose the organization to legal jeopardy, such as showing that it acted as a gateway or stepping stone for attacks.

2.3.4 Don't Grow the Haystack; But Do Find More Needles

A system that increases the amount of information available to investigators is often a double-edged sword. The investigator may benefit from additional information, but that benefit is directly proportional to the information's quality. For example, Bratus et al. [19] and Piatek et al. [119] highlight how current practices in DMCA copyright enforcement focus on broad and highly automated techniques resulting in unacceptably high rates of false positives. This problem isn't just limited to p2p investigations. Clark and Landau [33] also question the utility of packet attribution in forensics. We believe their criticisms extend to other areas such as de-anonymization. Poor information may lead to tangible costs such as wasted resources or intangible costs such as emotional distress for the falsely accused.

2.4 Summary and Conclusions

In this chapter, we discussed the key principles of digital forensics as well as the goals and constraints of investigators. We use this information to provide a set of useful lessons for researchers, and as the foundation of the forensic techniques proposed in this thesis.

Computer security researchers have the potential to make significant contributions to digital forensics; however, they must first understand the forensics context and its differences with existing security models. Similarly, the onus is on computer security venues to support these efforts by recruiting knowledgeable reviewers who are familiar with the challenges and requirements of forensics; otherwise, the security community risks encouraging low-impact work while rejecting worthwhile solutions to forensic problems.

CHAPTER 3
MOBILE PHONE TRIAGE

In this chapter, we apply the lessons from Chapter 2 to our first forensic scenario: mobile phone triage. We present DEC0DE, a system for recovering information from phones with unknown storage formats, a critical problem for forensic triage. Because phones have myriad custom hardware and software, we examine only the stored data. Via flexible descriptions of typical data structures, and using a classic dynamic programming algorithm, we are able to identify call logs and address book entries in phones across varied models and manufacturers. We designed DEC0DE by examining the formats of one set of phone models, and we evaluate its performance on other models. Overall, we are able to obtain high performance for these unexamined models: an average recall of 97% and precision of 80% for call logs; and average recall of 93% and precision of 52% for address books. Moreover, at the expense of recall dropping to 14%, we can increase precision of address book recovery to 94% by culling results that don't match between call logs and address book entries on the same phone.

3.1 Introduction

When criminal investigators search a location and seize computers and other artifacts, a race begins to locate off-site evidence. Not long after a search warrant is executed, accomplices will erase evidence; logs at cellular providers, ISPs, and web servers will be rotated out of existence; and leads will be lost. Moreover, investigators make the most progress during on-scene interviews of suspects if they are able to ask about on-scene evidence. Mobile phones are of particular interest to investigators. Address book entries and call logs contain valuable information that can be used to construct a timeline, compile a list of accomplices, or demonstrate intent. Further, phone numbers can provide a link to a geographical location via billing records. For crimes involving drug trafficking, child exploitation, and homicide, these leads are critical [107].

The process of quickly acquiring important evidence on-scene in a limited but accurate fashion is called *forensic triage* [104]. Unfortunately, digital forensics is a time-consuming task, and once computers are seized and sent off site, examination results are returned after a months-long work queue. Getting partial results on-scene ensures certain leads and evidence are recovered sooner.

Forensic triage is harder for phones than desktop computers. While the Windows/Intel platform vastly dominates desktops, the mobile phone market is based on more than ten operating systems and more than ten platform manufacturers making use of an unending introduction of custom hardware. In 2010, 1.6 billion new phones were sold [103], with billions of used phones still in use. *Smart phones* store information from thousands of applications each with potentially custom data formats. *Feature phones*, while simpler devices, are quick to be released and replaced by new models with different storage formats. Both types of phones are problematic as phone application, OS, and file system specifications are closely guarded as commercial secrets. Companies do not typically release information required for correct parsing. We focus on feature phones in this chapter and smart phones in Chapter 4.

Assuming the phone is not locked by the user, the easiest method of phone triage is to simply flip through the phone's interface for interesting information. This time-consuming process can destroy the integrity of evidence, as there is no guarantee data will not be modified during the browse. Similarly, backups of the phone may be examined, but neither backups nor manual browsing will recover deleted data and data otherwise hidden by the phone's interface. Hidden data can include metadata, such as timestamps and flags, that can demonstrate a timeline and user intent, both of which can be critical for the legal process.

Forensic investigation begins with data acquisition and the parsing of raw data into *information*. The challenge of phones and embedded systems is that too often the exact data format used on the device has never been seen before. Hence, a manual process of reverse engineering begins — a dead-end for practitioners. Recent research on automated reverse engineering is largely focused on the instrumentation of the system and executables [22, 41]. While accurate and reasonable for the common Windows/Intel desktop platform, construction of a new instrumentation system for every phone architecture-OS combination in use would require significant time for each and an expertise not present in the practitioner community.

In this chapter, we focus on a *data-driven approach* to phone triage. We seek to quickly parse data from the phone without analyzing or instrumenting software. We aim to obtain high quality results, even for phones that have not been previously encountered by our system. Our solution, called DEC0DE, leverages success from already examined phones in the form of a flexible library of probabilistic finite state machines. Our main insight is that the variety of phone models and data formats can be leveraged for recovering information from new phones. We make three primary contributions:

- We propose a method of *block hash filtering* for revealing the most interesting blocks within a large store on a phone. We compare small blocks of unparsed data from a target phone to a library of known hashes. Collisions represent blocks that contain content common to the other phones, and therefore not artifacts specific to the user, e.g., phone numbers or call log entries. Our methods work in seconds, reducing acquired data by 69% on average, without removing usable information.

- To recover information from the remaining data, we adapt techniques from natural language processing. We propose an efficient and flexible use of probabilistic finite state machines (PFSMs) to encode typical data structures. We use the created PFSMs along with a classic dynamic programming algorithm to find the maximum likelihood parse of the phone's memory.

- We provide an extensive empirical evaluation of our system and its ability to perform well on a large variety of previously unexamined phone models. We apply our PFSM set — unmodified — to six other phone models from Nokia, Motorola, and Samsung and show that our methods are able to recover call logs with 97% recall and 80% precision and address books with 93% recall and 52% precision for this set of unseen models.

There are a series of commercial products that parse data from phones (e.g., .XRY, cellebrite, and Paraben). However, these products rely on slow, manual reverse engineering for each phone model. Moreover, none of these products will attempt to parse data for previously unseen phone models. Even the collection of all such products does not cover all phone models currently on the market, and certainly not the set of all models still in use. In contrast, we design and evaluate a general approach for automatically recovering information on previously unseen devices, one that leverages information from past success.

3.2 Methodology and Assumptions

Our goal is to enable triage-based data recovery for mobile phones during criminal investigations. Below, we provide a definition of triage, our problem, and our assumptions. We focus on feature phones in this chapter and extend our techniques to smart phones in Chapter 4.

Unlike much related work, our focus is not on incident response, malware analysis, privilege escalation, protocol analysis, or other topics related to security primitives. We aim to have an impact on any crime where a phone may be carried by the perpetrator before the crime, held during the crime, used as part of the crime or to record the crime (e.g., a trophy photo), or used after the crime.

3.2.1 The triage process

The process of quickly acquiring important evidence in a limited but accurate fashion is called *forensic triage* [104]. Our goals are focused on the law enforcement triage process, which begins with a search warrant issued upon probable cause, or one of the many lawful exceptions [76] to the Fourth Amendment (e.g., incidence to arrest)[1]. Law enforcement has several objectives when executing a search and performing triage. The first is locating all devices related to the crime so that no evidence is missed. The second is identifying devices that are not relevant to the crime so that they can be ignored, as every crime lab has a months-long backlog for completing forensic analysis. That delay is only exacerbated by adding unneeded work. The third is interviewing suspects at the crime scene. These interviews are most effective when evidence found on-scene is presented to the interviewed person. Similarly, quickly determining leads for further investigation is critical so that evidence or persons do not disappear. Central to all of these objectives is the ability to rapidly examine and extract vital information from a variety of devices, including mobile phones.

Phone triage is not a replacement for gathering information directly from carriers; however, it can take several weeks to obtain information from a carrier. Moreover, carriers store only limited information about each phone. While most keep call logs for a year, other information is ephemeral. Text message content is kept for only about a week by Verizon and Sprint, and the IP address of a phone is kept for just a few days by AT&T [36]. In contrast, the same information is often kept by the phone indefinitely and, if deleted, it is still possibly recoverable using a forensic examination[2].

The less time it takes to complete a triage of each device, the more impact our techniques will have. While some crime scenes involve only a few devices, increasingly crime scenes involve tens and potentially hundreds of devices. For example, an office can be the center of operations for a gang, organized crime unit, or para-military cell. Typically little time is available and, in the case of search warrants, restrictions are often in place on the duration of time that a location can be occupied by law enforcement. In military scenarios, operations

[1]See the discussion in Chapter 2 for an introduction to U.S. legal requirements.

[2]See Section 4.5 for an analysis of residual data on smart phones.

may involve deciding which, if any, of several persons and devices in a location should be brought back using the limited space in a vehicle; forensic triage is a common method of deciding.

3.2.2 Problem definition

Our goal is to enable investigators to extract information quickly (e.g., in 20 minutes or less) from a phone, regardless of whether that exact phone model has been encountered before. We limit our results to information that is common to phones — address books and call logs — but is stored differently by each phone. Triage is not a replacement for a secondary, in-depth examination; but it does achieve shortened delay with a minimal reduction in *recall* and *precision*. Recall is the fraction of all records of interest that are recovered from a device; precision is the fraction of recovered records that are correctly parsed.

3.2.3 Data acquisition

We make the following assumptions in the context of on-site extraction of information from embedded devices. The technical process of extracting a memory dump from a phone starts off very differently compared to laptops and desktops. Data on a phone is typically stored in custom solid state memory. These chips are typically soldered onto a custom motherboard, and data extraction without burning out the chip requires knowledge of pinouts. For that reason, several other methods are in common use for extracting data. Broadly, data can be extracted representing either the *logical* or *physical* layout of memory. Often these representations are referred to as the logical or physical *image* of a device, respectively.

A logical image is typically easier to obtain and parse; however, it suffers from some serious limitations. First, it only contains information that is presented by the file system or other application interfaces. It omits deleted data, metadata about content, and the physical layout of data in memory (which we use in our parsing). Second, logical-extraction interfaces typically enforce access rules (e.g., preventing access to a locked phone) and may modify data or metadata upon access. Examples of logical extraction include using phone backup software or directly browsing through a phone using its graphical user interface. Due to the above deficiencies, our techniques operate directly on the physical image.

A physical image contains the full layout of data stored in a phone's memory, including deleted data that has not yet been overwritten; however, parsing raw data presents a significant challenge to investigators — one our techniques attempt to address. We discuss the parsing challenges further in Section 3.3.2.

Physical extraction requires an interface that is below the phone's OS or applications. There are a few different ways of acquiring a physical image. For example, some phones are compatible with *flasher* boxes [75], while others allow for extraction via a JTAG interface, or physical removal of the chip. Physical extraction typically takes between a few minutes and an hour depending on the extraction method, size of storage, and bus bandwidth. When we evaluate our techniques, we assume the prior ability to acquire the physical image of the phone.

Numerous companies sell commercial products that acquire data from phones, both logically and physically. This acquisition process is easier than the recovery of information from raw data, though still a challenge and not one we address. Of course, we do not expect our methods to be used on phones for which the format of data is already known. But no company offers a product that addresses even a large portion of the phone market and no combination of products covers all possible phones, even among the market of phones still being sold.

3.2.4 Limitations of our threat model

We assume the owner of the phone has left data in a plaintext, custom format that is typical of how computers store information. We allow for encryption and even simple obfuscation, but we do not propose techniques that would defeat either. While this threat model is weak, it is representative of phone users involved in traditional crimes. Some

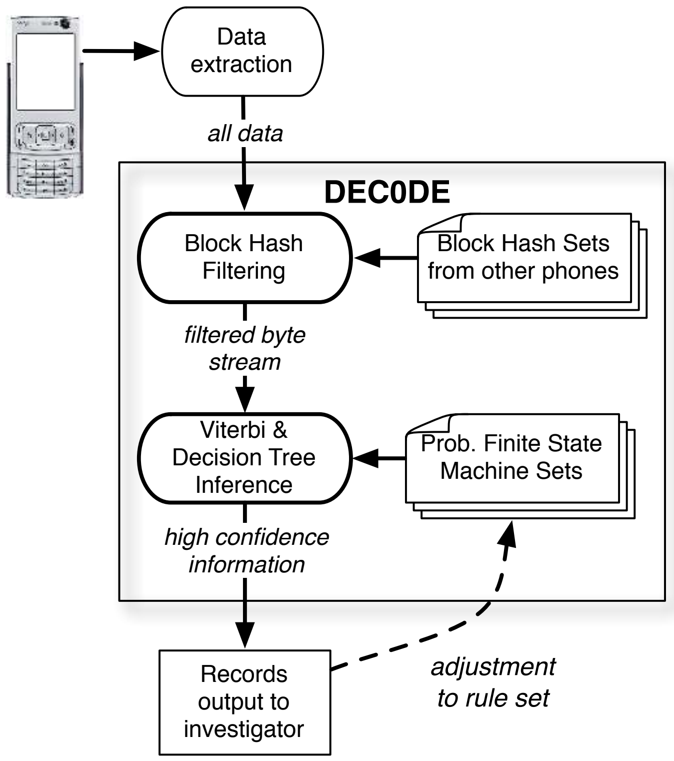

Data extraction
all data
DEC0DE
Block Hash Filtering
Block Hash Sets from other phones
filtered byte stream
Viterbi & Decision Tree Inference
Prob. Finite State Machine Sets
high confidence information
Records output to investigator
adjustment to rule set

Figure 3.1: An illustration of the DEC0DE's process. Data acquired from a phone is passed first through a filtering mechanism based on hash sets of other phones. The remaining data is input to a multistep inference component, largely based on a set of PFSMs. The output is a set of records representing information found on the phone. The PFSMs can optionally be updated to improve the process.

smart-phones encrypt data, most do not; and almost all feature phones do not. Further, it is not possible for one attacker to encrypt the data of every other phone in existence, and our techniques work on all phones for which plaintext can be recovered. In other words, while we allow for any one person to encrypt their data, it does not significantly limit the impact of our results.

3.3 Design of DEC0DE

In this section, we provide a high-level overview of DEC0DE including its input, primary components, and output.

DEC0DE takes the physical image of a mobile phone as input. We can think of the physical image as a stream of bytes with an unknown structure and no explicit delimiters. DEC0DE filters and analyzes this byte stream to extract important information, presenting the output to the investigator. The internal process it uses is composed of two components, illustrated in Figure 3.1: *(i)* block hash filtering and *(ii)* inference.

DEC0DE uses the block hash filter to exclude subsequences of bytes that do not contain information of interest to investigators. The primary purpose of this filtering is to reduce the amount of data that needs to be examined and therefore increase the speed of the system.

DEC0DE parses the filtered byte stream to extract information first in the form of *fields* and then as *records*. Fields are the basic unit of information and they include data types such as phone numbers and timestamps. Records are groups of semantically related fields that contain evidence of interest to investigators, e.g., address book entries. The inference component is designed to be both extensible and flexible, allowing an investigator to iteratively refine rules and improve results when time allows.

3.3.1 Block Hash Filtering

DEC0DE's block hash filtering component (BHF) is based on the notion that long identical byte sequences found on different phones are unnecessary for triage. That is, such sequences are unlikely to contain useful information for investigators. Mobile phones use a portion of their physical memory to store operating system software and other data that have limited utility for triage. BHF is designed to remove this cruft and reduce the number of bytes that needs to be analyzed, thereby increasing the speed of the system.

DEC0DE's block hash filter logically divides the input byte stream into small subsequences of bytes. We refer to each of these subsequences as a *block*. DEC0DE filters out a block if its hash value matches a value in a library of hashes computed from other phones. Blocks may repeat within the same phone, but only the first occurrence of each block remains after filtering. DEC0DE uses block hashes, rather than a direct byte comparison, to

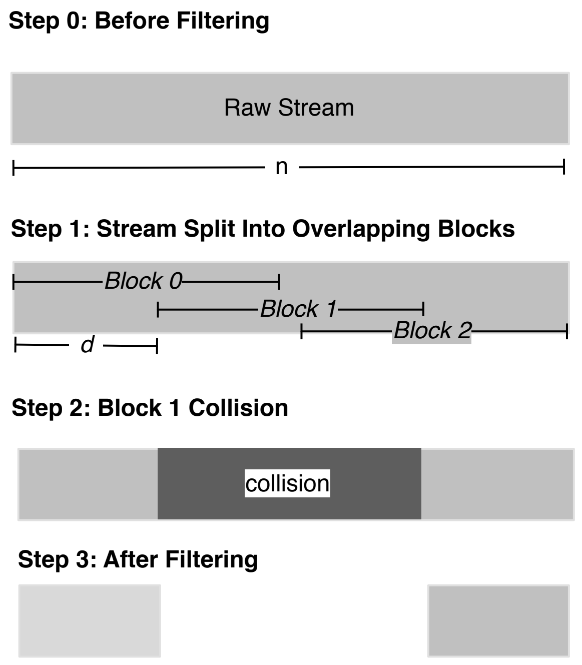


Figure 3.2: Block hash filtering takes a stream of n bytes and creates a series of overlapping blocks of length b. The start of each block differs by $d \leq b$ bytes. Any collision of the hash of a block with a block on another phone (or the same phone) is filtered out.

improve system performance; However, BHF may lead to erroneous filtering due to block collisions. One type of collision arises when blocks with different byte sequences share the same hash value. Another type of collision occurs when blocks share the same subsequence even though they actually contain user information. Currently, DEC0DE mitigates the risk of collisions by using a cryptographic hash function and a sufficiently large block size.

To make the filter more resilient to small perturbations in byte/block alignment, DEC0DE uses a sliding window technique with overlap between the bytes of consecutive blocks [142]. In other words, the last bytes of a block are the same as the first bytes of the next block.

More formally, DEC0DE logically divides an input stream of n bytes, into blocks of b bytes with a shift of $d \leq b$ bytes between the start of successive blocks. The SHA-1 hash value for each block is computed and compared to the hash library. DEC0DE filters out all matched blocks. Figure 3.2 illustrates a simple example.

As we show empirically in Section 3.5, nearly all of the benefit of block hash filtering can be realized by just using another phone of the same make and model. This result ensures BHF is scalable as the test phone need not be compared to all phones in an investigator's library.

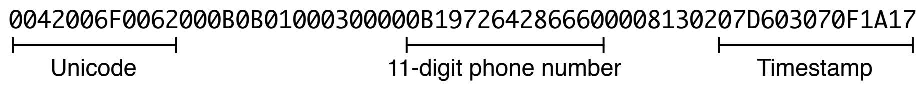
The general idea of our block hash filter is similar to work by a variety of researchers in a number of domains [55,78,142]. Our primary contribution is the empirical analysis of the technique in the phone domain. Further discussion of related work is given in Section 3.6.

3.3.2 Inference

After block hash filtering has been performed, what remains is a reduced ad hoc data source about which we have only minimal information. Our goal is to identify certain types of structured information, such as phone numbers, names, and other data types embedded in streams of this data.

Parsing phones is particularly challenging due to the inherent ambiguity of the input byte stream. Along with the lack of explicit delimiters, there is significant overlap between the encodings for different data structures. For example, certain sequences of bytes could be interpreted as both a valid phone number and a valid timestamp. For these reasons, simple techniques like the unix command `strings` and regular expressions will be mostly ineffective.

DEC0DE solves this ambiguity by using standard probabilistic parsing tools and a probabilistic model of encodings that might be seen in the data. DEC0DE obtains the maximum likelihood parse of the input stream creating a hierarchical description of information on the phone in the form of fields and records. More concretely, the output of DEC0DE is a set of call log and address book *records*. Each record is comprised of *fields* representing phone numbers, timestamps, strings, and other structures extracted from the raw stream.



```
0042006F0062000B0B01000300000B1972642866600008130207D603070F1A17
```

Figure 3.3: A simplified example of raw data as stored by a Nokia model phone, labeled with the correct interpretation. DEC0DE outputs a call log: the Unicode string “Bob”; the phone number (0xB digits long and null terminated) 1-972-642-8666; and the timestamp 3/7/2006 3:26:23 PM.

3.3.2.1 Fields and Records

Within the block filtered data source, we have no information about where records or fields begin or end, and we have no explicit delimiters. Figure 3.3 shows simplified example data that could encode an address book entry in a Nokia phone; DEC0DE would receive this snippet embedded and undelineated in megabytes of other data. Unlike large objects, such as jpegs or Word docs, such small artifacts are difficult to isolate and can easily appear randomly.

To infer information found on phones, DEC0DE uses standard methods for *probabilistic finite state machines* (PFSMs), which we describe here. As implied above, we have a lower level of *field* state machines that encode raw bytes as phone numbers, timestamps, and other types. We also have a higher level of *record* state machines that encode fields as call log entries and address book entries. For example, a call log record can be flexibly encoded as a phone number field and timestamp field very near to one another; the encoding might also include an optional text field.

Each field’s PFSM consists of one or more *states*, including a set of *start* states and a set of *end* states. Each state has a given probability of transitioning to another state in the machine. Each state *emits* a single byte during each state transition of the PFSM. The emitted byte is governed by a probability distribution over the bytes from 0x00 to 0xFF. Restricting the set of bytes that can be output by a state is achieved by setting the probability of those outputs to zero. For example, an *ASCII alphabetic* state would only assign non-zero probabilities to the ASCII codes for “a” through “z” and “A” through “Z”. Every PFSM in DEC0DE’s set is targeted towards a specific data type. If correctly defined, a field’s PFSM will only accept a sequence of bytes if that sequence is a valid encoding of the field type. We constructed the field PFSMs based on past observations (see Section 3.4.1).

Examples of DEC0DE's specific field types include 10-digit phone numbers, 7-digit phone numbers, Unicode strings, and ASCII strings. Each specific field is associated with a *generic field* type such as *text* or *phone number*. Some fields have fixed lengths and others have arbitrary lengths.

We define *records* in a similar manner. Records are represented as PFSMs, except that each state emits a generic field rather than a raw byte.

Given the set of PFSMs representing each field type that we have encoded, we then *aggregate* them all into a single *Field PFSM*. We separately aggregate all record PFSMs into a single *Record PFSM*. The aggregation naively creates transitions from every field's end state to every other field's start states with some probability, and we do the same for compiling records. (We discuss setting these probabilities below.) In the end, we have two distinct PFSMs that are used as input to our system, along with data from a phone.

3.3.2.2 Finding the maximum likelihood sequence of states

Our basic challenge is that, for a given phone byte stream that is passed to the inference component of DEC0DE, there will be many possible way to parse the data. That is, there are many ways the PSFMs could have created the observed data, but some of these are more likely than others given the state transitions and the output probabilities. To formalize the problem, let $B = b_0, b_1, ..., b_n$ be the stream of n bytes from the data source. Let $S = s_0, s_1, ..., s_n$ be a sequence of states which could have generated the output bytes. Our goal then, is to find

$$\underset{s_0,s_1,...,s_n}{\arg\max} P(s_0, s_1, ..., s_n | b_0, b_1, ..., b_n), \tag{3.1}$$

i.e., the maximum probability sequence of states given the observed bytes. These states are chosen from the set encoded in the PFSM given to DEC0DE. The probabilities assigned to

PFSM's states, transitions, and emissions affect the specific value that satisfies the above equation.

In a typical *hidden Markov model*, one assumes that an output byte is a function only of the current unknown state, and that given this state, the current output is independent of all other states and outputs. Using this assumption, and noting that multiplying the above expression by $P(b_0,...,b_n)$ does not change the state sequence which maximizes the expression, we can write

$$\begin{aligned}
& \arg\max_{s_0,...,s_n} P(s_0,...,s_n|b_0,...,b_n) \\
= & \arg\max_{s_0,...,s_n} P(s_0,...,s_n|b_0,...,b_n)P(b_0,...,b_n) \\
= & \arg\max_{s_0,...,s_n} P(s_0,...,s_n,b_0,...,b_n) \\
= & \arg\max_{s_0,...,s_n} P(s_0,...,s_n)P(b_0,...,b_n|s_0,...,s_n) \\
= & \arg\max_{s_0,...,s_n} P(s_0,...,s_n)\prod_{i=0}^{n} P(b_i|s_i). \qquad (3.2)
\end{aligned}$$

Naively enumerating all possible state sequences and selecting the best parse is at best inefficient and at worst intractable. One way around this is to assume that the current state depends only on the state that came immediately before it, and is independent of other states further in the past. This is known as a first order Markov model, and allows us to write

$$\begin{aligned}
& \arg\max_{s_0,...,s_n} P(s_0,...,s_n)\prod_{i=0}^{n} P(b_i|s_i) \\
= & \arg\max_{s_0,...,s_n} P(s_0)P(s_1|s_0)P(s_2|s_0,s_1) \\
& ...P(s_n|s_0,s_1,...,s_{n-1})\prod_{i=0}^{n} P(b_i|s_i) \\
= & \arg\max_{s_0,...,s_n} P(s_0)\prod_{i=1}^{n} P(s_i|s_{i-1})\prod_{i=0}^{n} P(b_i|s_i). \qquad (3.3)
\end{aligned}$$

The Viterbi algorithm is an efficient algorithm for finding the state sequence that maximizes the above expression. The complexity of the Viterbi algorithm is $O(nk^2)$ where n and k are the number of bytes and states. For a full explanation of the algorithm, see for example the texts by Viterbi [149] or Russell and Norvig [131].

3.3.2.3 Fixed length fields and records

Markov models are well suited to data streams with arbitrary length fields. For example, an arbitrary length text string can be modeled well by a single state that might transition to itself with probability α, or with probability $1-\alpha$ to some other state, and hence terminating the string. Unfortunately, first order Markov models are not well suited to modeling fields with fixed lengths (like 7-digit phone numbers), since it is impossible to enforce the transition to a new state after 7 bytes when one is only conditioning state transition on a single past state. In other words, a first order Markov model cannot "remember" how long it has been in a particular state.

Since it is critical for us to model certain fixed length fields like dates and phone numbers, we have two options:

- Add a *separate new state* for every position in a fixed length field. For example, a 7-digit phone number would have seven different separate states, rather than a single state.
- Implement an mth order Markov model, where m is equal to the length of the longest fixed length field we wish to model.

The first option, under a naive implementation, leads to a very large number of states, and since Viterbi is $O(nk^2)$, it leads to impractical run times.

The second option, using an mth order Markov model, keeps the number of states low, but can also lead to very large run times of $O(nk^{m+1})$. However, by taking advantage of the fact that *most* state transitions in our model only depend upon a single previous state, and other structure in our problem, we are able to implement Viterbi, even for our fixed length fields, in time that is close to the implementation for a first order Markov model with a small number of states. Similar techniques have been used in the language modeling literature to develop efficient higher-order Markov models [98].

3.3.2.4 Hierarchical Viterbi

DEC0DE uses Viterbi twice. First, it passes the filtered byte stream to Viterbi with the Field PFSM as input. The output of the first pass is the most likely sequence of generic fields associated with the byte stream. That field sequence is then input to Viterbi along with the Record PFSM for a second pass. We refer to these two phases as *field-level* and *record-level* inference, respectively.

The hierarchical composition of records from fields (which are in turn composed of bytes) can be captured by a variety of statistical models, including context free grammars. The main reason we chose to run Viterbi in this hierarchical fashion, rather than integrating the information about a phone type in something like a context free grammar, was to limit the explosion of states. In particular, because we have a variety of fixed-length field types, such as phone numbers, the number of states required to implement a seamless hierarchical model would grow impractically large. Our resulting inference algorithms would not have practical run times.

The decomposition of our inference into a field-level stage and a record-level stage makes the computations practical at a minimal loss in modeling power. The reason that DEC0DE can operate on phones that are unseen is that record state machines are very general. For example, we don't require timestamps to phone numbers to appear in any specific order for a call log entry. We require only that they are both present.

3.3.2.5 Post-processing

The last stage of our inference process takes the set of records recovered by Viterbi and passes them through a decision tree classifier to remove potential false positives. We refer to this step as *post-processing*. We use a decision tree classifier because it able to take into account features that can be inefficient to encode in Viterbi. For example, our classifier considers whether a record was found in isolation in the byte stream, or in close proximity to other records. In the former case, the record is more likely to be a false positive. Our evaluation results (Section 3.5) show that this process results in significant improvements to precision with a negligible effect on recall.

We use the Weka J48 Decision Tree, an open source implementation of a well-known classifier (`http://www.cs.waikato.ac.nz/ml/weka`). In general, a decision tree can be used to decide whether or not an input is an example of the target class for which it is trained. The classifier is trained using a set of feature tuples representing both positive and negative examples. In our case, the decision tree decides whether a given record, output from our Viterbi stage, is valid. We selected a set of features common to both call log and address book records: number of days from the average date; frequency of phone numbers with same area code; number of different forms seen for the same number (e.g., 7-digit and 10-digit); number of characters in string; number of times the record appears in memory; distance to closest neighbor record. We do not claim that our choice of features and classifier is optimal; it merely represents a lower bound for what is possible.

Post-processing does not inhibit the investigator, it is a filter intended to make the investigator's work easier. To this end, DEC0DE can make both the pre- and post-processing results available ensuring that the investigator has as much useful information as possible.

For our evaluation, the positive training examples consisted of true records from a small set of phones called our *development set* (described in detail in Section 3.5). To create the negative training examples, we used a 10 megabyte stream of random data with byte values selected uniformly at random from 0x00 to 0xFF. We input the random data to DEC0DE's Viterbi implementation and used the resulting output records as negative examples. We found that this provided better results than using negative examples found on real phones.

3.4 Implementation of State Machines

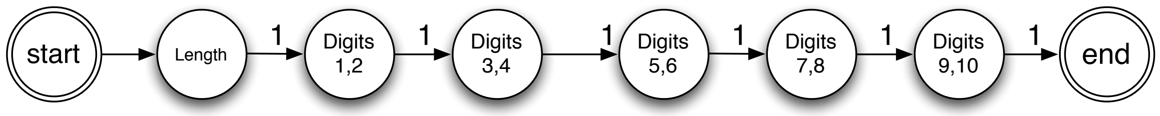
In the previous section, we presented DEC0DE's design broadly; in this section, we focus on the core of the inference process: the probabilistic finite state machines (PFSM).

Generic type	Specific type	Num. States
Records		
Call logs	Nokia call log composed of text, phone num., timestamps	8
	General call log composed of text, phone num., timestamps	9
Address books	General address book composed of phone numbers, text	5
Fields		
Phone number	ASCII	11
	Unicode	22
	Nokia 10 digit	6
Timestamp	UNIX	4
	Samsung	4
	Nokia	7
Text	ASCII bigram	6
	Unicode	7
Number index	Nokia number index	1
unstructured	unstructured	1

Table 3.1: Examples of types that we have defined in DEC0DE.

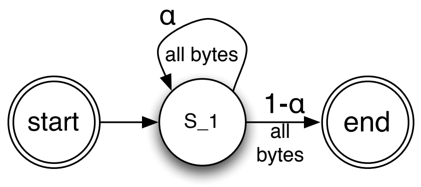
DEC0DE's PFSMs support a number of generic field types such as phone number, call log type, timestamp, and text as well as the target record types: address book and call log. Table 3.1 shows some example field types that we have defined and the number of states for each. In all, DEC0DE uses approximately 40 field-level and 10 record-level PFSMs.

Most fields emit fixed-length byte sequences. For example, the 10-digit phone number field is defined as 10 states in which state k (for $k \neq 1$) can only be reached by state $k-1$. The state machine for a 10-digit phone number as found on many Nokia phones is:



As mentioned in the previous section, each state emits a single byte; since Nokia often stores digits as nibbles, each state in the machine encodes two digits. The emission probability is governed by both the semantics of the Nokia encoding and real-world constraints. For example a 10-digit phone number (in the USA) cannot start with a 0 or a 1 and therefore the first state in the machine cannot emit bytes 0x00–0x1F, i.e., the emission probability for each of these bytes is zero.

Some fields, such as an *unstructured byte stream* have arbitrary length. Such a field is simply defined by a single state with probability α of transitioning to itself, and probability $1-\alpha$ of terminating. In fact, this specific field is special: DEC0DE uses the unstructured field as a "catch-all" for unknown or unstructured portions of the byte stream. Byte sequences that do not match a more meaningful field type will always match the *unstructured* field, which is:



We emphasize that our goal is *not* to produce a full specification of the format of a device. While we would certainly be delighted if this were an easy problem to solve, we note that we can extract significant amounts of useful information from a data source even when large parts of the format specification are not understood. Hence, rather than solving the problem of complete format specification, we seek to extract as many records as possible according to our specification of records. It is also important to note that our field and record definitions may ignore large amounts of structure in a phone format. Only a minimal amount of information about a phone's data organization is needed to define useful fields and records. We return this point in Section 3.5.4.

3.4.1 Coding State Machines

We created most of the PFSMs used in DEC0DE using a hex editor and manual reverse engineering on a small subset of phones that we denote as our *development set*. We limited the development set to one phone model each from four manufacturers with multiple instances of each model: the Nokia 3200B, Motorola v551, LG G4015 and Samsung SGH-T309. We intentionally did not examine any other phone models from these manufacturers prior to the evaluation of DEC0DE (Section 3.5) so that we could evaluate the effectiveness of our state machines on previously unobserved phone models.

We also used DEC0DE itself to help refine and create new state machines, both field and record level, for the development phones. This process was very similar to how we imagine an investigator would use DEC0DE during the post-triage examination.

Once we reached high recall for the development set, we fixed the PFSMs and other components using DEC0DE without modification for the extent of our evaluation regardless of what model was parsed.

3.4.2 Selecting Transition Probabilities

A sequence of bytes may match multiple different field types. Similarly, a sequence of fields may match multiple record types. Viterbi accounts for this by choosing the most likely type. It may appear that a large disadvantage of this approach is that we must manually set the type probabilities for both fields and records. However, Viterbi is robust to the choice of probabilities: the numerical values of the field probabilities are not as important as the probability of one field relative to another.

3.5 DEC0DE Evaluation

We evaluated DEC0DE by focusing on several key questions.

1. How much data does the block hash filtering technique remove from processing?
2. How effectively does our Viterbi-based inference process extract fields and records from the filtered data?
3. How much does our post-processing stage improve the Viterbi-based results?
4. How well does the inference process work on phones that were unobserved when the state machines were developed?

3.5.1 Experimental Setup

We made use of a number of phones from a variety of manufacturers. The phones contained some GUI-accessible address book and call log entries, and we entered additional entries using each phone's UI. A combination of deleted and GUI-accessible data was used in our tests; however, most phones contained only data that was deleted and therefore unavailable from the phone's interface but recoverable using DEC0DE. The phones we obtained were limited to those that we could acquire the physical image from memory (i.e., all data stored on the phone in its native form). The list of phones is given in Table 3.2. Our evaluation focuses on feature phones, i.e., phones with less capability than smart phones.

As stated in Section 3.4.1, we performed all development of DEC0DE and its PFSMs using only the Nokia 3200B, Motorola v551, LG G4015, and Samsung SGH-T309 phones. We kept the *evaluation set* of phones separate until ready to evaluate performance. We acquired the physical image for all phones using Micro Systemation's commercial tool, .XRY.

We focus on two types of records: address book entries and call log entries. We chose these record types because of their ubiquity across different phone models and their relative importance to investigators during triage. We evaluate the performance of DEC0DE's inference engine based on two metrics, *recall* and *precision*. Recall is the fraction of all phone records that DEC0DE correctly identified: the number of true positives over the sum of false negatives and true positives. If recall is high, then all useful information on a phone has been found. Precision is the fraction of extracted records that are correctly parsed: the number of true positives over the sum of false positives and true positives. If precision is high then the information output by DEC0DE is generally correct.

Often these two metrics represent a trade-off, but our goal is to keep both high. In law enforcement, the relative importance of the two metrics depends on the context. For generating leads, recall is more important. For satisfying the *probable cause* standard required by a search warrant application, moderate precision is needed. Probable cause has

Make	Model	Count	MB
PFSM Development Set			
Nokia	3200b	4	1.4
Motorola	V551	2	32.0
Samsung	SGH-T309	2	32.0
LG	G4015	2	48.0
Evaluation Set			
Motorola	V400	2	32.0
Motorola	V300	2	32.0
Motorola	V600	2	32.0
Motorola	V555	2	32.0
Nokia	6170	2	4.9
Samsung	SGH-X427M	2	16.0

Table 3.2: The phone models used in this study. The table shows the number we had of each and the size of local storage.

been defined as "fair probability"[3] that the search warrant is justified, and courts do not use a set quantitative value. For evidence meeting the *beyond a reasonable doubt* standard needed for a criminal conviction, very high precision is required, though again no quantitative value can be cited.

For each of our tested phones, we used .XRY not only to acquire the physical image, but also to obtain *ground truth results* that we used to compare against DEC0DE's results. It was often the case that DEC0DE obtained results that .XRY did not. And in those cases, we manually inspected the result and decided whether they were true or false positives (painstakingly using a hex editor). We made conservative decisions in this regard, but were able to employ a wealth of common sense rules. For example, if a call entry seemed to be valid and recent, but was several years from all other entries, we labeled it as a false positive. Similarly, an address book entry for "A.M." is most reasonably assumed to be a true positive while ",!Mb" is most reasonably a false positive; even though both have two letters and two symbols, the latter does not follow English conventions for punctuation. It would be impractical to program all such common sense rules and our manual checking is stronger in

[3]United States v. Sokolow, 490 U.S. 1 (1989)

that regard. Occasionally, DEC0DE extracts partially correct or noisy records. We mark each of these records as wrong, unless the only error is a missing area code on the phone number.

3.5.2 Block Hash Filtering Performance

The goal of BHF is to reduce the amount of data that DEC0DE must parse, reducing run time, without sacrificing recall. On average, we find that BHF is able to filter out about 69% of the phone's stored data without any measurable effect on inference recall. The BHF algorithm has only two parameters: the shift size d and the block size b. Our results show that the shift size does not greatly affect the algorithm's performance, but it has a profound effect on storage requirements. Also, we found that performance varies with block size, but not as widely as expected.

For each value of b and d that we tested, we kept the corresponding BHF sets in an SQL table. The database was able to match sets in tens of seconds, so we do not report run time performance results here. As an example, on a moderately resourceful desktop, DEC0DE is able to filter a 64 megabyte phone, with $b = 1024$ and $d = 128$, in under a minute.

Ideally, we (and investigators) would want our hash library to be comprised entirely of new phones. If our library contains used phones, there is a negligible chance that the same common user data (e.g., an address book entry with the same name and number) will appear on different phones, align perfectly on block boundaries, and be erroneously filtered out. Regardless, it was impractical for us to find an untouched, new phone model for every phone we tested. If data was filtered out in this fashion because of our use of pre-owned phones, it would likely have shown up in the recall values in the next section; since the recall values are near perfect, we can infer this problem did not occur.

3.5.2.1 Filtering

First, we examined the effect of the block size b on filtering. Figure 3.4 shows the overall filter percentage of our approach for varying block sizes. In these experiments, we set $d = b$

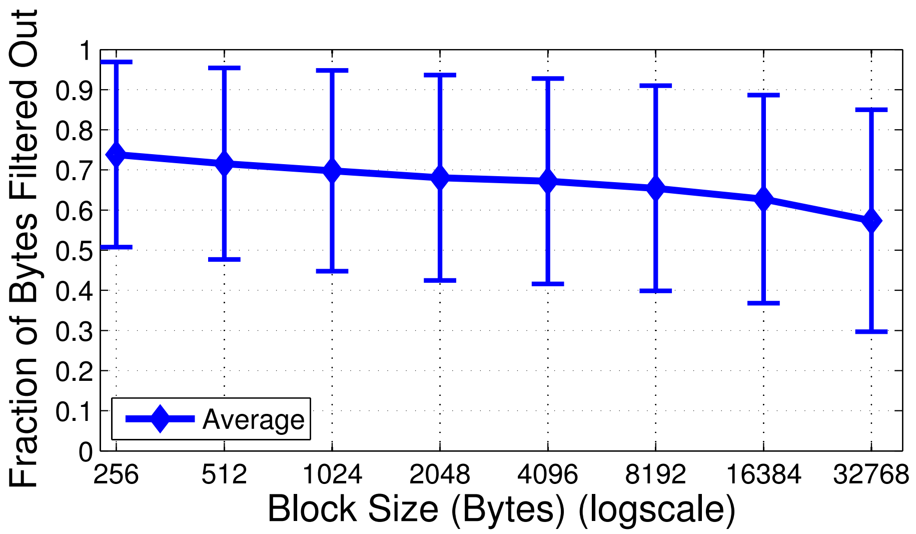
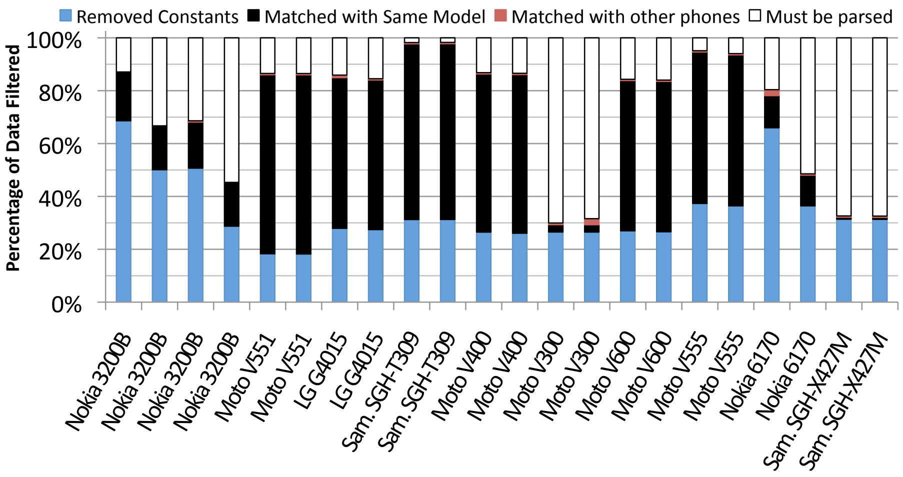


Figure 3.4: The average performance of BHF as block size varies for all phones listed in Table 3.2 (logarithmic x-axis). Error bars represent one standard deviation. In all cases we set $d = b$ (i.e., shift size is equal to block size), but performance does not vary with d in general.

so that there was never overlap. The line plots the average for all phones. As expected, the smaller block sizes make more effective filters. However, a small block size results in more blocks and consequently, greater storage requirements. On average in our tests, 73% of data is filtered out when $b = 256$, while only slightly less, 69%, is filtered out when $b = 1024$.

Second, we examined the affect of the shift amount d on filtering. In our tests, we fixed $b = 1024$ and varied d=$\{32, 64, 128, 256, 512, 1024\}$. However, there is less than a 1% difference in filtering between $d = 32$ and $d = 1024$ for all phones. (No plot is shown.) Again, the affect of d is on storage requirements, which we discuss below.

Third, we isolated what type of data is filtered out for each phone using fixed block and shift sizes of $b = 1024$ and $d = 128$; we use these values for all other experiments in this chapter. Figure 3.5 shows the results as stacked bars; the top graph shows filtering as a percentage of the data acquired from the phone, and the bottom graph shows the same results in megabytes. For each of the 25 phones, the bottom (blue) bar shows the percentage of data filtered out because the block was a repeated, constant value (such as a run of zeros). The middle (black) bar shows the percentage of data that was in common with a different



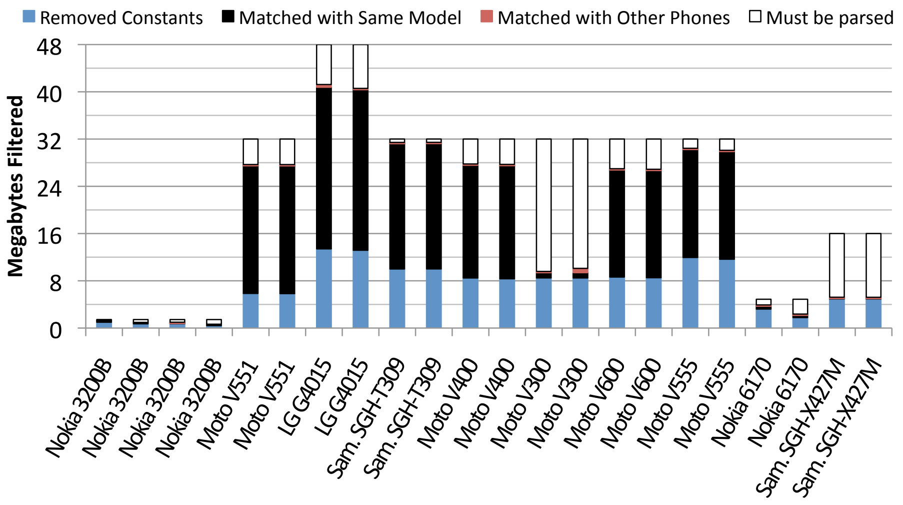


Figure 3.5: The amount of data remaining after filtering is shown as solid white bars, as a percentage (top) and in MB (bottom). On average, 69% of data is successfully filtered out. Black bars show data filtered out because they match data on another instance of the same model. Blue bars show data filtered out because it is a single value repeated (e.g., all zeros). Red bars show data filtered out because it appears on a different model. ($b = 1024$ bytes, $d = 128$ bytes)

instance of the same make and model phone. The top red bar shows the percentage of data that can be filtered out because it is only found on some phone in the library that is a different make or model. The data that remains after filtering is shown in the top, white box.

On average, 69% of data is removed by block hash filtering. Generally, the technique works well. On average, half of the filtered out data was found on another phone of the same model. These percentage values are in terms of the complete memory, including blocks that were filled with constants (effectively empty). Therefore, as a percentage of non-empty data, the percentage of filtered out data is higher. These results suggest that it is often sufficient to only compare BHF sets of the same model phone. However, in some models less than 3% of data was found on another instance of the same model. This poor result was the case for the Samsung SGH-X427M and Motorola V300. Finally, the results shown in the Figure 3.5 (bottom), suggest that the performance of BHF was not correlated with the total storage space of the phone.

Our results in the next section on inference, in which DEC0DE examines only data remaining after filtering, demonstrate that filtering does not significantly remove important information: recall is 93% or higher in all cases.

3.5.2.2 Storage

An important advantage of our approach is that investigators can share the hash sets of phones, without sharing the data found within each phone. This sharing is very efficient as the hash sets are small compared to the phones. The number of blocks from each phone that must be hashed and stored in a library is $O((n-b)/d)$, though only unique copies of each block need be stored. Given that $n >> b$, the number of blocks is dependent on n and d and the affect of b on storage is insignificant. However, since it is required that $d \leq b$, the algorithm's storage requirements does depend on b's value in that sense. As an example, for a 64 megabyte phone, when $b = 1024$ bytes and $d = 128$ bytes, the resulting BHF set is 524,281 hash values. At 20-bytes each, the set is 10 megabytes (15% of the phone's storage).

Since we need perhaps only one or two examples of any phone model, the cumulative space needed to store BHF sets for an enormous number of phone models is practical. Since BHF gains nearly all benefit from comparing phones of the same model, comparison will always be fast.

In order to be effective, the library needs to be constructed using the same hash function and block size for all phones; however, the shift amount need not be the same. The storage requirement of the library is inversely proportional to the shift size and thus is minimized when $d = b$. Conversely, BHF removes the most data when $d = 1$. We can effectively achieve maximal filtering with minimal storage using $d = b$ for the library and $d = 1$ for the test phone. The cost of this approach is more computation and consequently higher run times. A full analysis is beyond the scope of this chapter.

3.5.3 Inference Performance

To evaluate our inference process, we used DEC0DE to recover call log and address book entries from a variety of phones. In our results, we distinguish between the performance of the Viterbi and decision tree portions of inference. Additionally, we make clear the performance of DEC0DE on phones in our development set versus phones in our evaluation set. All results in this section assume that input is first processed using BHF.

Figure 3.6 shows the performance of our inference process for call logs; the top results are before the post-processing step and the bottom after post-processing. The white-space break in the chart separates the development set of phones (on the left), and the evaluation set (on the right). We put the most effort toward encoding high quality PFSMs for the Nokia and Motorola phones. Not surprisingly, the results are best in general for these makes, indicating that the performance of DEC0DE is dependent on the quality of the PFSMs. However, the results also show that DEC0DE can perform well even for the previously unseen phones in the evaluation set. Overall, recall of DEC0DE is near complete at 98% for development phones and 99% for evaluation phones. Precision is more challenging, and after Viterbi is at

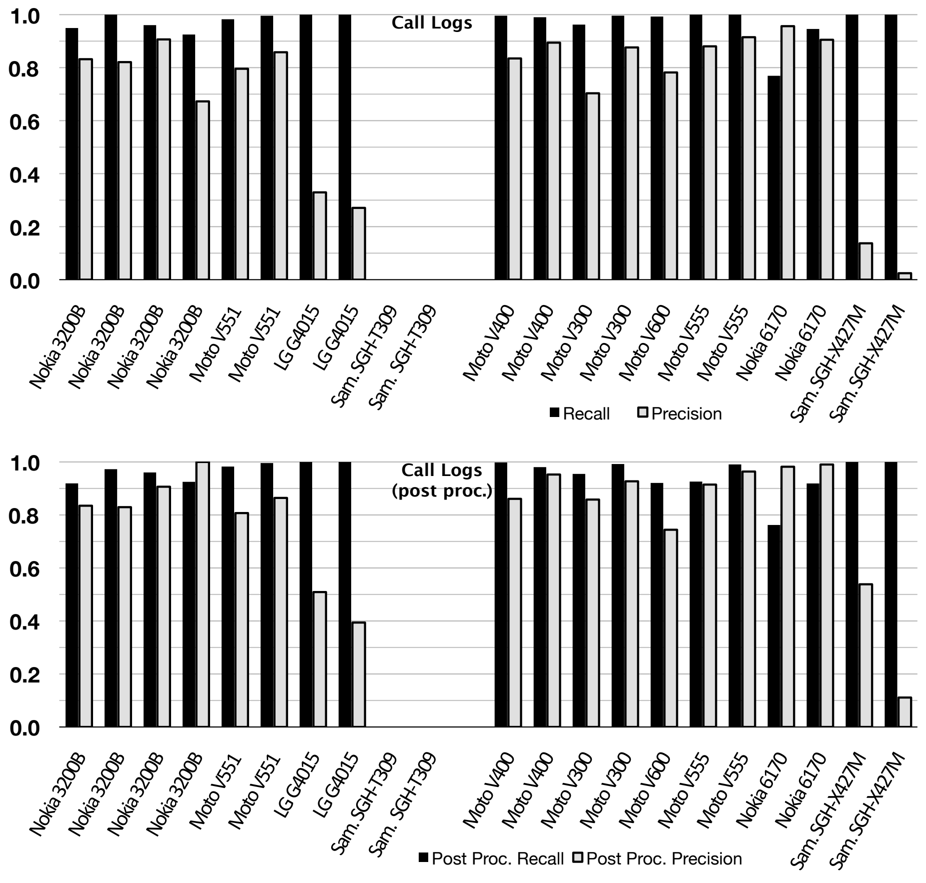
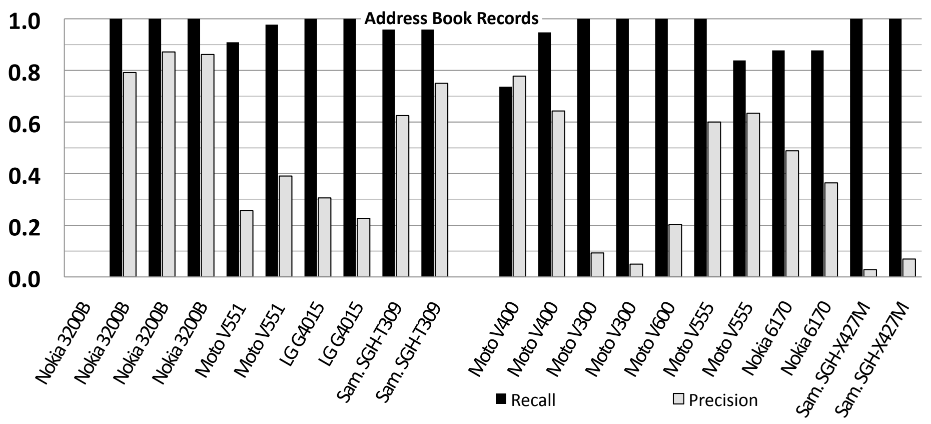


Figure 3.6: Precision and recall for call logs. (Top) Results after only Viterbi parsing. (Bottom) Results after post-processing. Left bars are development set; right bars are evaluation set. In all graphs, black is recall and gray is precision. On average, development phones have recall of 98%, and precision of 69% that increases to 77% after post processing. On average, evaluation phones have recall of 97%, and precision of 72% that increases to 80% after post processing. The T309 had no call log entries, which explains in part DEC0DE's poor performance for the X427M.



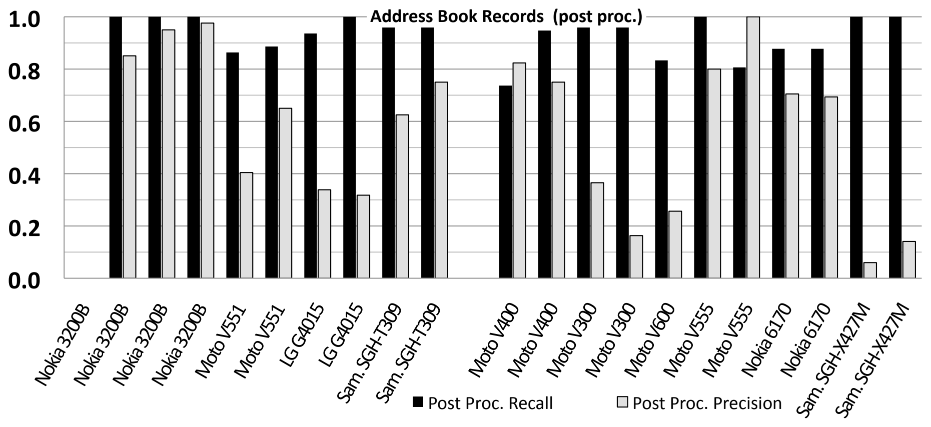


Figure 3.7: Precision and recall for Address Book entries. (Top) Results after only Viterbi parsing. (Bottom) Results after post-processing. On average, development phones have recall of 99%, and precision of 56% that increases to 65% after post processing. On average, evaluation phones have recall of 93%, and precision of 36% that increases to 52% after post processing. N.b, The first Nokia has no address book entries at all.

69% for development phones and 72% for evaluation phones. It is important to note that no extra work on DEC0DE was performed to obtain results from the phones in the evaluation set, which is significant compared to methods that instrument executables or perform other machine and platform dependent analysis. After post-processing, the precision for the development and evaluation phones increased to 77% and 80% respectively.

Figure 3.7 shows the performance of our inference process for address book records. As before, the top results are after filtering but not post-processed while the bottom are post-processed. Overall, recall of the DEC0DE is again high at 99% for development phones and 93% for evaluation phones. Precision after Viterbi is 56% for development

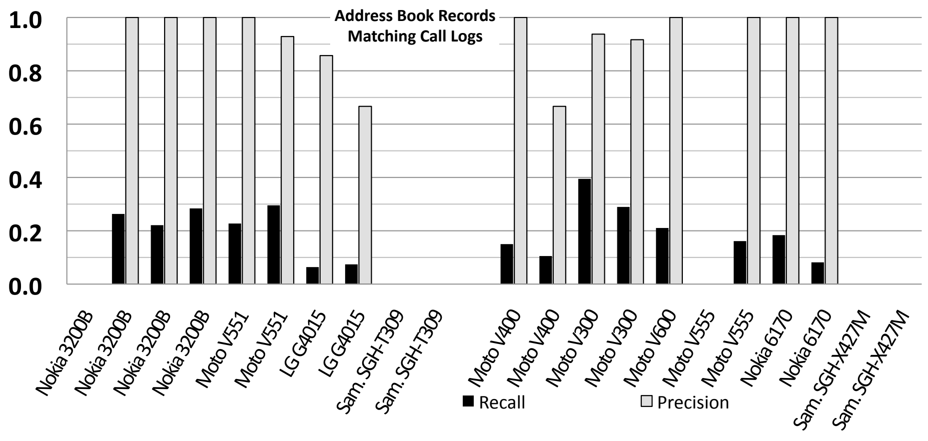


Figure 3.8: Precision and recall for Address Book entries *after results are culled that do not match phone numbers in* DEC0DE*'s call logs for the same phone.* For some phones, all results are culled. On average, development phones have recall of 16%, and precision of 92% (when results are present). On average, evaluation phones have recall of 14%, and precision of 94% (when results are present).

phones and 36% for evaluation phones. After post processing by the decision tree, the precision for all phones increased, by an average of 61% over the Viterbi-only results, a significant improvement. For development phones, precision increases to 65% on average. (Note that the development phones are used to train the classifier.) For evaluation phones, precision increases significantly to 52%.

While performance is not perfect, we could likely improve performance by using a different set of PFSMs for each different phone manufacturer. In our evaluation, all PFSMs for all manufactures are evaluated at once. Because our goal is to allow for phone triage, we don't reduce the set of state machines for each manufacturer; however, a set of manufacturer-specific state machines could improve performance at the expense of being a less general solution.

We also note that when recall is high, it is easier to discover the intersection of information found on two independent phones from the same criminal context; that intersection is likely to be a better lead than most.

When necessary, we can prioritize precision over recall. Figure 3.8 shows the results of culling records for where the phone number in the address book does not also appear in the call log: precision is increased to 92%, although recall drops to 14%. (We don't show the same process for call logs.) This simple step shows how easy it is to isolate results for investigators that deem precision of results more important than recall. Moreover, the results that are culled are still available for inspection.

3.5.3.1 Execution time

Inference is the slowest component of DEC0DE. The post processing step takes a few seconds, but the Viterbi component takes significantly longer. On average, DEC0DE's Viterbi processes 12,781 bytes/sec. The smaller phones in our set (Nokias) finish in a few minutes, while the larger Motorola can completed in about 15 minutes. Since the Viterbi processing already works with distinct blocks of input produced by the BHF component, we implemented a parallel version of DEC0DE for our work with smart phones (Chapter 4), thereby greatly increasing speed.

3.5.4 Limitations

Our evaluation is limited in a number of ways in addition to what was previously discussed. First, as with any empirical study, our results are dependent on our test cases. While our set of phones is limited, it contains phones from a variety of makes and models. In Chapter 4 we test DEC0DE against smart phones and find that it performs poorly, returning thousands of irrelevant results; we address these problems with LIFTR, a new post-processing approach based on investigator-feedback. Second, our tests are performed only on call logs and address book entries. We extended DEC0DE to support SMS text messages; however, the phone set contained so few text messages that we elide those results.

Our approach also has a number of limitations. First, we don't address the challenge of acquiring the physical memory image from phones, which is an input needed for DEC0DE. Here, we have leveraged existing tools to do so. However, acquisition is an independent endeavor and varies considerably with the platform of the phone. Part of our goal is to show that despite hardware (and software) differences, one approach is feasible across a spectrum of devices. Second, DEC0DE's performance is tied strongly to the quality of the PFSMs. Poorly designed state machines, especially those with few states, can match any input. We do not offer an evaluation of whether it is hard or time consuming to design high quality PFSMs or other software engineering aspects of our problem; we report only our success. Third, a single PFSM has an inherent endianness embedded in it. DEC0DE does not automatically reorganize state machines to account for data that is the opposite endianness. Fourth, we have not explicitly demonstrated that phones do indeed change significantly from model to model or among manufactures. This assertion is suggested by DEC0DE's varied performance across models but we offer no overall statistics.

It is also important to note that DEC0DE is an *investigative* tool and not necessarily an *evidence-gathering* tool. Tools for gathering evidence must follow a specific set of legal guidelines to ensure the admissibility of the collected evidence in court. For example, the tool or technique must have a known error rate (see Daubert v. Merrell Dow Pharmaceuticals, 509 U.S. 579 (1993)).

Finally, our approach is to gather artifacts that match a description that may be too vague in some contexts. For example, using our current set of PFSMs DEC0DE ignores important metadata that is encoded in bit flags that may indicate if a entry is deleted. Such metadata can be useful in investigations. To capture this information, we would need to first reverse engineer the metadata format and then create model-specific machines tailored to that metadata.

3.6 Related Work

Our work is related to a number of works in both reverse engineering and forensics. We did not compare DEC0DE against these works as each has a significant limitation or assumption that does not apply well to the criminal investigation of phones.

Polyglot [23], Tupni [41], and Dispatcher [22] are instrumentation-based approaches to reverse engineering. Since binary instrumentation is a complex, time-consuming process, it is poorly suited to mobile phone triage. Moreover, our goal is different from that of Polyglot, Tupni, and Dispatcher. We seek to extract information from the data rather than reverse engineer the full specification of the device's format.

Other previous works have attempted to parse machine data without examining executables. Discoverer [40] attempts to derive the format of network messages given samples of data. However, Discoverer is limited to identifying exactly two types of data — "text" and "binary" — and extending it to additional types is a challenge. Overall, it does not capture the rich variety of types that DEC0DE can distinguish.

LearnPADS [49,50,158] is another sample-based system. It is designed to automatically infer the format of ad hoc data, creating a specification of that format in a custom data description language (called PADS). Since LearnPADS relies on explicit delimiters, it is not applicable to mobile phones.

Cozzie et al. [39] use Bayesian unsupervised learning to locate data structures in memory, forming the basis of a virus checker and botnet detector. Unlike DEC0DE, their approach is not designed to parse the data but rather to determine if there is a match between two instances of a complex data structure in memory.

In our preliminary work [143], we used the Cocke-Younger-Kasami (CYK) algorithm [68] to parse the records of Nokia phones. While this effort influenced the development of DEC0DE, it was much more limited in scope and function.

The idea of extracting records from a physical memory image is similar to *file carving*. File carving is focused on identifying large chunks of data that follow a known format, e.g., jpegs or mp3s. Some file carving techniques match known file headers to file footers [112, 128] when they appear contiguously in the file system. More advanced techniques can match pieces of images fragmented in the file system relying on domain specific knowledge about the file format [113]. In contrast, our goal is to identify and parse small sequences of bytes into records — all without any knowledge of the file system. Moreover, we seek to identify information within unknown formats that only loosely resemble the formats we've previously seen.

DEC0DE's filtering component is similar to number of previous works. Block hashes have been used by Garfinkel [55] to find content that is of interest on a large drive by statistically sampling the drive and comparing it to a bloom filter of known documents. This recent work has much in common with both the `rsync` algorithm [142], which detects differences between two data stores using block signatures, as well as the Karp-Rabin signature-based string search algorithm [78], among others.

3.7 Summary and Conclusions

We have addressed the problem of recovering information from phones with unknown storage formats using a combination of techniques. At the core of our system DEC0DE, we leverage a set of probabilistic finite state machines that encode a flexible description of typical data structures. Using a classic dynamic programming algorithm, we are able to infer call logs and address book entries. We make use of a number of techniques to make this approach efficient, processing data in about 15 minutes for a 64-megabyte image that has been acquired from a phone. First, we filter data that is unlikely to contain useful information by comparing block hash sets among phones of the same model. Second, our implementation of Viterbi and the state machines we encoded are efficiently sparse, collapsing a great deal of information in a few states and transitions. Third, we are able to improve upon Viterbi's result with a simple decision tree.

Our evaluation was performed across a variety of phone models from a variety of manufactures. Overall, we are able to obtain high performance for previously unseen phones: an average recall of 97% and precision of 80% for call logs; and average recall of 93% and precision of 52% for address books. Moreover, at the expense of recall dropping to

14%, we can increase precision to 94% by culling results that don’t match between call logs and address book entries on the same phone.

CHAPTER 4
EXTENDING TRIAGE TO SMARTPHONES

When forensic triage techniques designed for feature phones, such as DEC0DE, are applied to smart phones, these recovery techniques return hundreds of thousands of results, only a few of which are relevant to the investigation. We propose the use of relevance feedback to address this problem: a small amount of investigator input can efficiently and accurately rank in order of relevance, the results of a forensic triage tool. We present LIFTR, a novel system for prioritizing information recovered from Android phones. We evaluate LIFTR's ranking algorithm on 13 previously owned Android smart phones and three recovery engines — DEC0DE, Bulk Extractor, and `Strings`— using a standard information retrieval metric, Normalized Discounted Cumulative Gain (NDCG). LIFTR's initial ranking improves the NDCG scores of the three engines from 0.0 to an average of 0.73; and with as little as 5 rounds of feedback, the ranking score increases to 0.88. Our results demonstrate the efficacy of relevance feedback for quickly locating useful information among the large amount of irrelevant data returned by current recovery techniques. Further, our empirical findings show that a significant amount of important user information persists for weeks or even months in the expired space of a phone's memory. This phenomenon underscores the importance of using file system agnostic recovery techniques, which are the type of techniques that benefit most from LIFTR.

4.1 Introduction

As we discussed in Chapter 3, understanding the limits of what information is recoverable from phones is important for both lawful investigations and for individuals seeking confidentiality when phones are lost or stolen.

Our previous work on triage, DEC0DE, focused on using probabilistic inference on feature phones (Chapter 3). Others have used deterministic feature search for desktop systems [12,57]. In this chapter, we show that these existing approaches encounter significant problems when applied to smart phones. First, smart phones contain a great deal more information than feature phones. While feature phones are often limited to less than 100 MB of storage space, current smart phones store gigabytes of data. Consequently, locating important information often means an investigator must sift through an impractical amount of data.

Second, while the relatively limited number of smart phone operating systems would seem to make the investigator's job easier, mobile phones are supported by many different file systems and growing number of proprietary NAND storage devices with poorly documented flash translation layers (FTLs). Just as with feature phones, important data frequently lies in the deleted and expired sections of memory (Section 4.5), and it is often impractical and time consuming to create a specialized parser for each phone.

Finally, smart phones contain a wider variety of information, much of which is not relevant to the current investigation. Record definitions for address book entries or call logs are not sufficient by themselves. The data may be parsed correctly as a phone number, text, URL, or date, but it is not always content that is relevant to the context of the investigation.

We present the design and evaluation of a novel system, LIFTR, for prioritizing information recovered from Android phones. Our system is designed to build upon our work in Chapter 3 and to complement other existing triage approaches. LIFTR ranks information according to a combination of scoring and relevance feedback rounds from the investigator. LIFTR is a general approach that can be used with any data recovery technique that provides a string representation and offset of the recovered information. As we show, current techniques can return thousands, and in some cases, millions of *unranked* results per phone. Our approach prioritizes information relevant to an investigation, enabling the investigator to complete the triage process quickly and efficiently.

The primary insight behind LIFTR is that once we can locate a small piece of relevant information, we can quickly locate more by leveraging the spatial locality of data and semantic relationships between NAND pages.

LIFTR is especially powerful for recovering information from expired storage or in cases where the file system cannot be reconstructed — we find that for our set of pre-owned evaluation phones, upward of half of the NAND pages are expired. Other data on phones will be easily recoverable because the files are in allocated storage. In these cases, we expect parsing information is relatively easy (though labor intensive, it does not present a research challenge). On the other hand, reconstruction of files from expired space presents a fundamental problem in forensics [56,128]. Fragmentation or loss of critical segments can prevent recovery of a full file, requiring a specialized parser per application file format, without guarantees of recovering any information. Systems like DEC0DE and Bulk Extractor [57] overcome this limitation due to their file system agnostic inference techniques. However, being generalized *recovery engines*, they do not consider the semantics behind the content recovered and hence return large amounts of irrelevant results bearing no connection to the phone user. LIFTR tries to address these problems using an algorithm for ranking results from recovery engines in the order of their relevance to the investigation.

We make several contributions.

- We show empirically that previous approaches to forensic triage (DEC0DE and Bulk Extractor) do not scale to resource-rich smart phones. For one phone, DEC0DE returned over 6.2 million results, of which only about 12 thousand were relevant.
- We introduce LIFTR, a system that quickly identifies the important information by ranking inference results using relevance feedback. LIFTR works in concert with

existing triage techniques to provide a reliable ranking of results. It is especially powerful when analyzing expired or deleted storage areas.

- We examine LIFTR on 13 pre-owned Android phones from 6 different manufacturers, using three recovery engines: DEC0DE, Bulk Extractor, and `Strings`. All of the phones contained residual data from the previous owners[1]. The recovery engines return unranked results, resulting in a ranking score near zero (out of 1.0). LIFTR is able to rank results initially with an average ranking score of 0.73 when provided with 5 items of information from the investigator (e.g., first or last names, phone numbers or email addresses). Without such hints, LIFTR can leverage information about the file system and provide an initial ranking score of 0.43. With just 5 rounds of investigator feedback, the ranking score increases to 0.88 and 0.73, respectively. Increasing to 20 questions to the investigator, improves the ranking further to 0.91 and 0.79 for the two approaches.
- We develop techniques to parse and analyze the Yaffs file system. We use these techniques to characterize the lifetime and expiration of data on pre-owned phones; we find that data can live on the device for weeks or months after it has been logically deleted. Further, over half of the NAND pages (56%) contained deleted or expired data. Our results are consistent with previous work in secure data deletion on flash memory [124,125,155].

4.2 Problem Definition and Methodology

Our goal is to extract user-centric information from mobile phones that is relevant to an investigation. Our work in Chapter 3 focused on retrieving all information from a feature phone's data store, regardless of its provenance. While this approach is appropriate for

[1]Our procedures were approved by our Institutional Review Board.

phones with small amounts of storage and few applications, smart phones store data from innumerable applications, some of which is significantly more important than others.

Our approach takes the output of these systems as a starting point (we call them *recovery engines*), and identifies the content that is most important to an investigation, according to investigator feedback. Specifically, all information is ranked based on a combination of the investigator's relevance feedback, the actual content, and storage system locality information. We output a ranked list so that the investigator need only examine the most-highly ranked information among the tens of thousands of results returned by the recovery engines. We evaluate the success of LIFTR based on relevance of the results returned and the amount of feedback required from the investigator.

The intuition behind our approach is that *(i)* an investigator can quickly provide outside context that is important to the investigation but impractical to encode into a general recovery engine; and *(ii)* we can leverage the semantic relationship between fields to locate more information. For instance, imagine a scenario in which the recovery engine returns the phone numbers "413-555-1234" and "1-800-555-2266". While both fields may have been correctly parsed, the former has an area code local to the investigation while the latter is a toll-free number. Arguably, the local number is more likely to be relevant to the investigation. Once identified by the investigator as useful, we have higher confidence in other records containing the number, e.g., call logs and text messages. Further, if one of those text messages mentions a name, we may be able to use that name to find additional records, such as emails and chat logs.

4.2.1 NAND Flash

Our focus is on phones; as such, LIFTR's design and evaluation is in the context of NAND-based flash storage. Unlike magnetic disk drives, flash does not overwrite memory in place. Instead, whenever a file is changed, the modified portion is written to unallocated storage, leaving the *expired* data in the flash memory. These expired pages persist for an indefinite period of time — potentially days or weeks under normal usage; see Section 4.5. Recovery engines, such as DEC0DE, take advantage of this *deletion latency* to recover information that has been logically removed.

NAND file systems fall into two general categories: log-structured file systems designed to work with the raw storage (e.g., Yaffs) and block-device file systems that require an intermediate flash translation layer (FTL). FAT, Ext4, and Samsung's RFS are three common block-device file systems found on Android phones.

For block-device file systems, reconstructing files (or even identifying expired data) from the raw bytes requires knowledge of the FTL — often intractable given the proprietary and diverse nature of these algorithms. Instead, LIFTR assumes as little as possible about the system that stores the information, so that it is compatible with previously unexamined operating systems, file systems, and flash translation layers. Even so, LIFTR can take advantage of file system information when it is available.

4.2.2 Information Types

We are particularly interested in recovering information from the deleted (i.e., expired) pages in memory. These are the pages that are not accessible through a logical examination (i.e., via API calls to the phone's operating system). Because NAND does not overwrite memory in place, multiple copies of a page with slightly different content may be spread across the phone. As a result, even if a phone owner deletes an address book entry that record may still reside on the phone.

Our system works with whatever information is output by recovery engines, as long as the recovered information includes its location on the storage device. Such information includes address book entries, call logs, SMS messages, Facebook chats, and data from smart-phone applications.

4.2.3 Pre-processing Data

Our system recovers data from the *physical image* of a phone. Unlike the *logical image*, the physical image contains the complete layout of bytes in the phone's memory, including deleted data, and is free from any access restrictions that could prevent our system from using data from certain applications.

We do not address the problem of image extraction, assuming that the phone's physical image is already accessible. Also, we assume that the data in the extracted image is not encrypted; otherwise, the tested recovery engines would not be able to extract any information.

LIFTR requires that a phone's data has been pre-processed in three stages.

- First, the phone's physical image is acquired. Depending upon the model and file system of the phone at hand, hardware acquisition techniques (e.g., Joint Test Action Group (JTAG) connections), or software acquisition techniques (e.g., `nandump` or `dd`) can be used [67,147,150].
- Second, portions of the acquired image are filtered out when they are not expected to have user-centric information and are therefore of no interest to the investigator. These portions include binaries and resource files from the operating systems and applications. For example, DEC0DE's *block hash filtering* efficiently removes such data by removing blocks of data that have been observed on other phones, compared by hash value.
- Third, a recovery engine transforms raw data into information, including names, phone numbers, email addresses, and text messages. Examples include DEC0DE, DIMSUM [92], Bulk Extractor [57], and more simply, `Strings`. We require that the tool return the corresponding offset in the original phone image and string-based output. E.g., phone numbers should be normalized to text.

The results are input to LIFTR, which we detail in the next section.

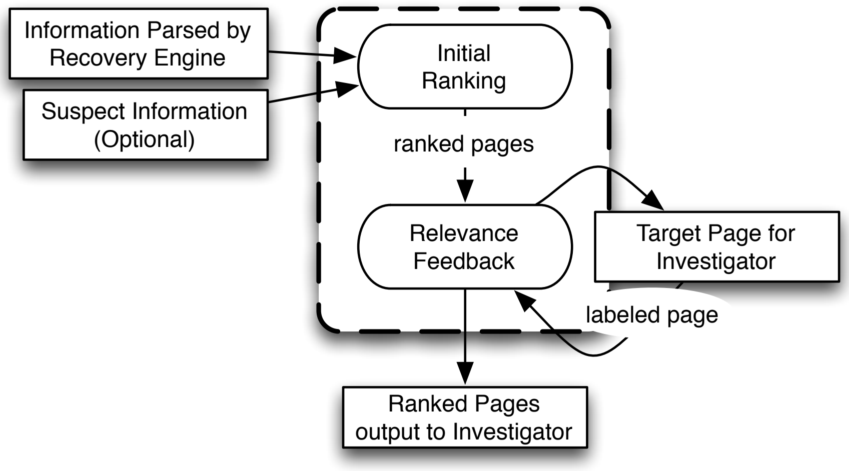

Information Parsed by Recovery Engine
Suspect Information (Optional)
Initial Ranking
ranked pages
Relevance Feedback
Target Page for Investigator
labeled page
Ranked Pages output to Investigator

Figure 4.1: A high-level overview of LIFTR and its two stages: initial ranking, and relevance feedback. LIFTR takes an unranked list of information parsed by the recovery engine and returns a ranked list of the most important NAND pages.

It's important to note that **DEC0DE and Bulk Extractor return an unranked, unscored list of results**. The results are grouped together by type (e.g., all credit cards and phone numbers), but grouping is also not practical for thousands or millions of results. Moreover, being unaware of the underlying file system layout, they are not designed to treat files like `contacts2.db`, that are rich in user data, any differently from others. Consequently, these engines have no way of prioritizing user-centric results over the rest.

4.3 Design of LIFTR

In this section, we provide a general overview of LIFTR and detail its components. We evaluate the effectiveness of LIFTR in Section 4.4.

Phone triage tools are often designed to return all data that can be parsed as legitimate information. This design works well for feature phones, but not for smart phones. As we show in Section 4.4, DEC0DE can return millions of results for smart phones containing data from real users, but only hundreds or thousands of results are relevant user data during triage. Sifting through false positives quickly becomes counterproductive for the investigator.

LIFTR expects that in a triage process, investigators can only review the top *n* results. Accordingly, it ranks information returned by a recovery engine (e.g., DEC0DE or Bulk Extractor) and asks the investigator to confirm the top-ranked page of results. Based on the feedback, it re-ranks the list. As more feedback is provided, our results show that accuracy improves, but of course the amount of feedback must be kept to a minimum. (We explore the effectiveness of up to 20 responses from a simulated investigator in our evaluations). LIFTR combines the feedback with information from the file and NAND storage-system, and statistics about the content itself to re-rank the results.

An overview of LIFTR is illustrated in Figure 4.1. Parsed *information* is input from a recovery engine, and combined with information, if any, known to the investigator. Such *a priori* information can include a few names or phone numbers. LIFTR's operation is comprised of two primary stages.

1. **Initial sorting.** Prior to asking for any investigator relevance feedback, LIFTR ranks the input information according to a relevance metric. At this stage, LIFTR also makes use of any *a priori* information provided. In general, better initial sorting implies that less feedback is needed from the investigator.
2. **Relevance feedback.** After the initial sorting, the investigator is asked to label a subset of the pages as true or false positives. The results are re-ranked after each instance of feedback.

We discuss both of these stages in greater detail below.

4.3.1 Page-level granularity

NAND file systems store files in pages. Files are stored across one or more pages, that are not necessarily contiguous in memory. When a file is deleted, the pages remain but are expired. Before they are reused by another file, NAND pages are wiped completely. Therefore, all information on a page belongs to a single file — there is no slack space in a NAND page.

Tools such as DEC0DE and Bulk Extractor search for information in units that we refer to as *fields*; e.g., a date, a phone number, or a credit card number. Each field is returned independent of and without its page information. However, we find that it is often more useful to consider information at a page-level granularity. As such, LIFTR both asks for feedback and presents results at the page level.

First, a page is the unit of write for the file system and, as such, the page typically belongs to just one file.

Second, pages are small and contain few fields, reducing the chances that an investigator will overlook a field and result in a false negative. In our experiments with real phones, DEC0DE returned on average 24 fields per page, and on average 7 relevant fields per true positive page; Bulk Extractor returned on average 6 fields per page, and on average 5 relevant fields per true positive page.

Third, in our experiments, we found that when fields from the same page are presented together to an investigator, they provide context that allows the investigator to determine more accurately their meaning. For example, a date field in isolation may be hard to evaluate as a true positive or not; but a surrounding set of phone numbers and strings, or similar dates, provide a context for deciding the correct feedback.

4.3.2 Initial Ranking

LIFTR's algorithm for the *initial ranking* takes as input a set of information fields discovered by a recovery engine, each tagged with its byte offset in the original image. The byte offset is used to group the fields by NAND page. Our goal is to rank the phone's pages based on how likely they are to contain information relevant to the user and investigation. To do so, LIFTR assigns an initial *quality score* to a page p using a *normalized weighted sum* of a set of features calculated for each field:

$$\text{quality}_0(p) = \frac{\sum_{i=1}^{|F_p|} \sum_{j=1}^{m} w_j a_{ij}}{|F_p|} \tag{4.1}$$

where F_p is the set of fields contained on page p; a_i is a vector of m features calculated for the i^{th} field using one of the features we discuss below, e.g., file system knowledge or *a priori* knowledge, and text value; and w is a vector of weights for each of these features, that sum to 1. LIFTR's initial ranking of pages is sorted from the highest to lowest quality scores.

For our evaluation, we consider different combinations of three field features, based on an investigator's *a priori* knowledge, limited file system information, and the quality of the field text, respectively. LIFTR can be easily extended to support additional features.

1. **Investigator's *a priori* knowledge.** In some cases, before forensic analysis begins, an investigator will have basic external knowledge about the case, the phone owner, or an accomplice; e.g., a first or last name, email address, or phone number. When we allow its use in our evaluations, LIFTR sets the value of fields that contain strings supplied by the investigator with a 1, and 0 otherwise. In our evaluations we allow only 5 strings as *a priori* input; see Section 4.4.
2. **File system knowledge.** In some cases, an investigator will have knowledge about the OS and how it is designed to store information. For example, in Android phones, the `contacts2.db` and `mmssms.db` files are used for storing call logs, address book records, and SMS messages. If used, this feature sets the value of fields found in pages belonging to these two files with a 1, and a 0 otherwise.
3. **Text false positives.** Smart phones contain a large amount of text, some of which is user created, but most is detritus such as cached Web pages, application resources, and high level code or configuration. We use a set of fixed rules to filter out these often-seen false positives. First, we assign a feature value of 0 to text fields that are three characters or fewer in length. Second, we look for fields that contain HTML documents and other bits of code. We give a value of 0 to fields using CamelCase or words commonly found in code (e.g., `SELECT` or `div`). We generated this list ahead of time using information found on unused, freshly installed Android phones. For

efficiency, we only calculate this feature value for pages that have a non-zero score for one of the above two features.

At first blush, it may seem that with the file system feature we are providing the answer key to LIFTR and no work remains. However, these two files contain a lot of information that is not easily recoverable by DEC0DE, Bulk Extractor, or `Strings`, and much of what is recovered is not useful (e.g., the metadata and portions of the schema). It's important to note that we evaluate LIFTR primarily using unallocated pages, under the assumption that the phone's information has been deleted and reinstalled (for many of our phones, this was actually the case). The original files are not easily reconstructed from these pages because not all pages are present, and it is not always clear if a page belongs to a particular file. We could have written a specialized SQLite fragment parser for these files, but our goal is to provide a general technique that makes use of only the fields output by recovery engines — in turn, those engines also seek to recover information in a way that is independent of the file type being recovered, without solving the difficult problem of generalizable file reconstruction. In short, these engines also seek to be generalizable to the largest number of scenarios. Section 4.5 contains more details on our recovery of unallocated pages.

4.3.3 Relevance Feedback Stage

LIFTR's initial ranking orders only those pages with a non-zero *a priori* or file system score; all other pages will have a zero quality score after initial ranking. During the relevance feedback stage, LIFTR refines the initial ranking by using investigator feedback.

At a high-level, our algorithm works as follows. LIFTR presents the investigator with the top-ranked page and asks him to label all of the relevant fields on that page. Using the positive labels, LIFTR then increases the quality score for all semantically related pages based on how strongly each relates to the current page. Then, LIFTR updates the page ranking, and it asks for feedback on the top-ranked page that hasn't been sent to the investigator. This cycle is repeated as many times as the investigator wishes. We discuss the details of each step below.

In general, the more feedback, the better the final ranking; but we find that the investigator need only label a few pages to bring about significant improvements in the page order. Further, the number of fields per page is typically very small with around 24 fields per page on average for DEC0DE, and only 7 useful fields per relevant page.

The details of the process are as follows.

4.3.3.1 Marking fields

As part of our implementation of LIFTR, we have written a small interface that makes it easy to quickly label the fields on a page. An investigator marks entire fields as relevant or not. LIFTR then breaks fields into *tokens* by splitting on whitespace and punctuation. (Phone numbers are not split by hyphens or other punctuation.) When the investigator marks all fields on a page, we say that a *round* has completed.

In sum, for a given page p, F_p is the set of all fields. We let K_p be the set of *all* tokens from all fields on page p. We say a token is relevant if the field it came from was marked as relevant. We let $T_p \subseteq K_p$ be the subset of all tokens that are *relevant* on a page p. To be clear, once a field (and thus its derivative tokens) is marked relevant on any page, it is relevant for all pages on the phone.

4.3.3.2 Finding semantically related pages

We consider two pages to be *semantically related* if the pages share a token that has been previously marked as relevant and is not in a token blacklist (described below). The intuition here is that once the investigator marks a few tokens as relevant (whether names, email addresses, or phone numbers), we should evaluate other pages that share this content.

In sum, we let $\mathcal{R} = \{r_1, \dots, r_i\}$ be the ordered set of pages that have been marked by the investigator, where r_i is the page marked in the ith round. We let $\mathcal{P}$ be the unordered set of pages as-yet-unmarked that share at least one token with a field from a page in $\mathcal{R}$, where the field has been marked as relevant.

4.3.3.3 Blacklisting tokens

LIFTR maintains a blacklist of tokens to exclude when calculating the quality score for the pages. Tokens are added to the blacklist for one of three reasons: *(i)* they are found in a natural language dictionary; *(ii)* they match a token found on a set of newly installed and unused Android phones; *(iii)* they are present in a field not marked as relevant for a page in $\mathcal{R}$. We let $\mathcal{B}$ be the set of blacklisted tokens.

The negative feedback helps to identify tokens that have no connection to the user and hence are of no interest to the investigator. This becomes important when a positively marked field string contains irrelevant tokens. For example in the string "I called John Doe today", the tokens "John" and "Doe" are highly relevant whereas "I", "called", and "today" provide little information. This approach prevents LIFTR from giving undue credit to pages bearing semantic relations to such tokens.

4.3.3.4 Calculating the new quality score

When a field is marked by the investigator as relevant on the current page, LIFTR increases the quality score for the pages in $\mathcal{P}$ as follows. For a given $p \in \mathcal{P}$, each $\tau \in T_p$ contributes a quality score proportional to its *Inverse Document Frequency (IDF)*. The IDF of a token is a measure of how often a token appears in the set of all pages, and it is calculated as:

$$\text{idf}(\tau) = \log\left(\frac{|\mathcal{P} \cup \mathcal{R}|}{|P_\tau|}\right) \text{where } P_\tau = \{q | q \in \mathcal{P} \cup \mathcal{R}, \tau \in K_q\} \tag{4.2}$$

Note that P_τ is the set of pages in $\mathcal{R}$ and $\mathcal{P}$ that contain τ. We use IDF as a means of reducing the impact of tokens that appear too frequently throughout the phone. It allows us to have investigators mark whole fields without distinguishing separate tokens, and yet the impact of irrelevant tokens present in relevant fields is diminished. For instance, we find that many frequently occurring irrelevant tokens, that are not in the blacklist $\mathcal{B}$ are specific to the phone at hand and hence no generalized rules can be hard-coded to avoid them. However, owing to high frequency, they have a low IDF score and would therefore have minimal impact on the quality scores.

Once a new page p has been marked by the investigator in round i (and moved from $\mathcal{P}$ to $\mathcal{R}$), all the pages remaining in $\mathcal{P}$ are re-scored. The quality score for a page $q \in \mathcal{P}$ after i rounds of relevance feedback is the sum of quality score from the previous round and the IDF scores for each relevant token that q shares with p (ignoring those tokens found in the blacklist $\mathcal{B}$).

$$\text{quality}_i(q) = \text{quality}_{i-1}(q) + \sum_{\tau \in T_q - \mathcal{B}} \text{idf}(\tau) \tag{4.3}$$

The value of $\text{quality}_0(q)$ is the initial ranking score for q, as shown in Equation 4.1. Note that $\text{quality}_0(p) = 0$ for pages that did not appear in the bootstrap set. Also, tokens are only evaluated in the rounds during which they first appear; e.g., if a token “foo” appears in page r_1, then it is ignored if it later appears in r_2.

4.3.3.5 Presentation Order

We refer to the order in which LIFTR presents pages to the investigator as the *presentation order*. Currently, we present the unlabeled pages with the highest current quality score. While this approach is not necessarily the most efficient for minimizing the number of questions that we need to ask the investigator, we find it is an effective heuristic.

4.4 LIFTR Evaluation

In this section, we evaluate LIFTR’s initial ranking and relevance feedback stages. Broadly, we focus on answering three important questions:

	DEC0DE				**Bulk Extractor**				**`Strings`**			
	Pages	True Pages	Fields	True Fields	Pages	True Pages	Fields	True Fields	Pages	True Pages	Fields	True Fields
HTC Desire HD	228,860	1,834	6,247,115	12,275	518	55	5,233	216	189,182	1,629	6,509,666	13,875
Samsung Galaxy Y	175,873	600	4,594,060	2,954	263	0	694	0	139,109	1,526	3,747,699	26,492
Motorola XT701	144,618	207	3,049,543	869	118	0	650	0	120,923	207	2,762,831	984
HTC Evo 4g	78,680	3,178	1,853,638	22,850	2,689	323	21,806	1,250	68,216	2,761	1,774,627	18,243
Samsung Galaxy Y Duos	60,422	3,612	1,530,630	21,399	2,150	1,715	18,550	14,235	51,324	4,093	1,281,450	35,334
HTC Wildfire	55,759	137	1,019,033	837	118	36	1,031	321	46,360	115	1,176,778	872
Samsung Galaxy Mini	53,456	1,390	1,329,715	4,031	1,329	275	5,599	1,139	40,288	434	1,020,422	2,134
Sony Xperia x10	30,063	16	855,464	154	20	4	57	22	22,447	22	958,791	159
Dell XCD35	20,494	221	594,469	1,621	90	24	528	49	16,191	222	539,018	2,268
Dell XCD28	17,932	118	495,099	841	27	6	190	20	13,189	78	433,917	734
HTC Legend	10,238	19	198,873	121	26	0	28	0	7,721	14	194,448	108
Huawei Ideos	8,829	667	170,143	6,799	81	9	1,106	15	7,777	683	202,654	7,022
Huawei 8500	6,305	3	116,524	41	12	0	28	0	5,764	3	135,157	50

Table 4.1: Page and field statistics for the three recovery engines. On average, less than 2% of the results returned by the recovery engines are relevant; for more than half of the phones this percentage is below 0.5%.

1. How effective is LIFTR at ranking relevant pages using different recovery engines?
2. How much feedback is required for LIFTR to be effective, and how does that translate to time required of investigators?
3. What page properties have an effect on relevance feedback?

We discuss each of these questions below.

4.4.1 Evaluation Methodology

LIFTR's goal is to help focus an investigator's limited resources by identifying pages that are most likely to contain information relevant to the investigation. It works in concert with existing inference approaches to seamlessly increase precision and scale. Once we have identified the important pages, the investigator can then perform a more thorough examination and perhaps employ more specialized techniques to extract additional information.

For our experiments we simulated feedback from an investigator, such that a field on a page is marked relevant if it contains a token also present in the `contacts2.db` or the `mmssms.db` file, e.g., a name, phone number, or email address.

4.4.1.1 Inference Engines

We tested the effectiveness of LIFTR to improve the results of three different triage techniques:

1. **DEC0DE** (Chapter 3), a probabilistic parsing approach that supports many types of data commonly found on mobile phones.
2. **Bulk Extractor** [12,57], a regular-expression based approach originally designed for desktop systems.
3. **`Strings`**, a common UNIX utility for identifying strings of printable characters in a file.

In many ways, DEC0DE and Bulk Extractor represent opposite design philosophies for recovering data. While DEC0DE is designed to be very flexible and prioritize the recovery of all information at the expensive of false positives, Bulk Extractor uses strictly defined regular expressions to limit the amount of irrelevant information, paying the cost of decreased true positives.

`Strings` represents a simple baseline approach that makes minimal assumptions about the structure of the underlying data. LIFTR's good performance with `Strings` demonstrates that the effectiveness of our system does not depend on the underlying engine.

We did not modify Bulk Extractor or `Strings` before applying them to smart phones, and made only minor changes to the DEC0DE workflow. DEC0DE typically groups the inferred fields into records — e.g., nearby text and phone number fields might be grouped together as an address book entry — and returns a list of these records to the investigator. We found it necessary to forgo the record step and only consider the field-level results. Records, as they are defined by DEC0DE, are centered around phone numbers existing in close proximity to names and other identifying information. However, since Android uses relational databases for storing data, the name for an address book entry will not necessarily appear in close proximity to the phone number in memory.

With careful modification, it is likely that DEC0DE and Bulk Extractor will perform significantly better than what we observed. For example, we could add additional regular expressions to Bulk Extractor that are specifically targeted toward Android data. However, our intention was not to measure the performance of the inference techniques, but instead to evaluate the effectiveness of LIFTR and demonstrate how it can benefit each of the recovery engines.

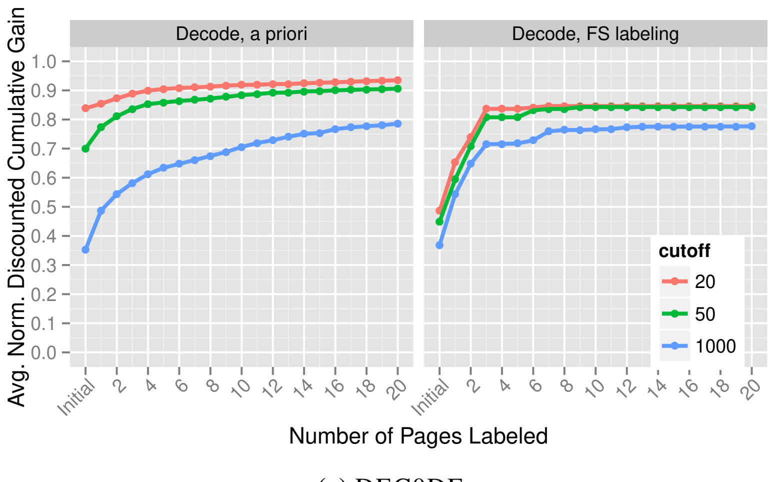
4.4.1.2 Phones

A partnering research group provided us with both the logical and physical images of 13 previously owned phones[2]. These 13 Android phones, listed in Table 4.1, were from 6 different manufacturers with users that lived in Canada, Hungary, India, Israel, Singapore, Serbia and the US. Each of these phones contained varying amounts of residual data from their previous users.

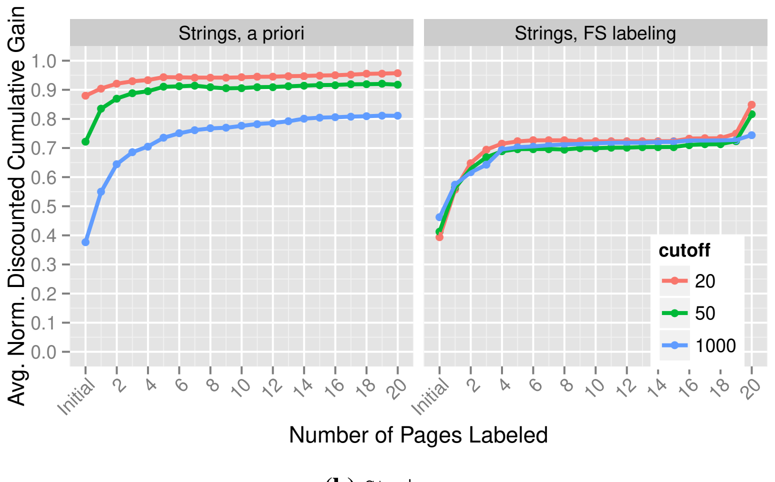
Before applying the recovery engines, we pre-filtered each physical image, to *(i)* reduce the amount of data passed to the inference tools, and *(ii)* when it could be identified, limit our analysis to the user data partition in the image. For 8 of the 13 phone images, we applied a filtering technique based on the Yaffs file system; we describe this approach in more detail in Section 4.5. For the remaining 5 phones phones, we employed the block hashing filtering approach described in Chapter 3.

Table 4.1 shows the number of pages returned by each of the tested inference techniques. For DEC0DE and `Strings`, the number of returned results is typically orders of magnitude greater than the actual number of relevant results. The percentage is less drastic for Bulk Extractor, but the number of fields returned by the tool is still often in the hundreds or thousands.

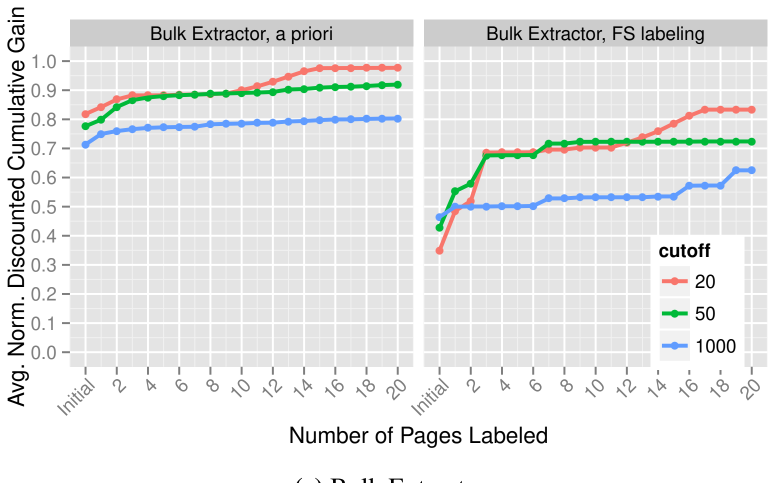
[2]We received IRB approval for this process. Phone images provided by Simson Garfinkel at the Naval Postgraduate School.



(a) DEC0DE



(b) `Strings`



(c) Bulk Extractor

Figure 4.2: The average NDCG score for DEC0DE, `Strings` and Bulk Extractor. The high initial score for $k = 20$ shows that LIFTR's *a priori* and *file system* ranking effectively places relevant pages early in the list. These top pages bootstrap the feedback process, enabling LIFTR to discover the large number of remaining pages — evidenced by the steady NDCG increase for $k = 1000$. LIFTR's results are consistent for all three recovery engines.

Because we did not have direct access to the phones, our evaluations were limited to the provided images, i.e., we could not manually manipulate the phones. However, this limitation mimics the restrictions placed on real world investigations, wherein the investigator must take great care not to modify the phone (beyond the extent necessary to extract the image).

4.4.1.3 Ground Truth

We collected ground truth from the contacts and SMS databases taken from the logical image of each of the phones. That is, we used tokens collected from the *allocated* versions of the SQLite databases `contacts2.db` and `mmssms.db`, respectively.

We considered a field provided by a recovery engine to be *relevant* if it contained a token — name, phone number, email address — found in one of the two ground truth databases. For example, if the allocated version of the `contacts2.db` file included the number 212-555-0123, we considered any field from any page containing this number as relevant. Consequently, relevant fields are not limited to the pages in contacts or SMS databases. We described the field tokenization process previously in Section 4.3.3.

LIFTR presents results to the investigator at a page-level granularity. We define a *relevant page* as any page that contains at least one relevant field — a field with a token pertinent to the investigation. For our evaluation, field types were either phone numbers, names or email addresses.

Limiting our ground truth to pages that have tokens belonging to the SMS and contacts databases gives us a lower bound on the amount of data present on the target phone. While there are other potential data sources we could draw our ground truth from (e.g., the geolocation database), grabbing this information would require modification to DEC0DE and Bulk Extractor. Further, our goal is to test the effectiveness of our feedback approach, and not that of the recovery engines. In other words, DEC0DE may not correctly infer all of the relevant fields on the phone. When evaluating our ranking, we only consider the relevant fields that each recovery engine was able to identify.

4.4.1.4 Initial Ranking Features

In our experiments, we consider two combinations of the initial-ranking features from Section 4.3.2: *(i)* file system knowledge paired with text quality and *(ii) a priori* information paired with text quality. For the *a priori* method we use all of the phones from our set, 13 in total. For the file system feature, we limit the evaluation to the eight phones with a parsable Yaffs user data partition. We used a feature weight of 0.6 for file system and *a priori*, and 0.4 for text quality.

Our set includes nine phones with the Yaffs file system. We wrote a special Yaffs parser to identify expired pages and label each page with its likely filename; we describe this process in Section 4.5. One of nine Yaffs phones could not be parsed properly by our techniques. This is because we were unable to separate the user data partition from the rest of the physical image. The other four phones used either Ext4 or RFS (a variant of FAT) and their physical images were acquired below a flash translation layer, making it impossible for us to label the pages with their suspected file names without knowledge of the FTL's mapping algorithms. In the non-Yaffs phones, LIFTR was not aware of the file to which a page belonged.

For the *a priori* setting, we select five relevant tokens randomly from the pool of all relevant/user related tokens like a name, phone number or an email address. The initial sorting score of pages to which these tokens belong would be increased, causing them to rise above the rest. The performance of LIFTR using the *a priori* approach is averaged over 30 trials for each of the 13 phones. In the setting where we use file system knowledge, the initial sorting scores of pages associated with user-content rich files like contacts and SMS are stepped-up instead, causing such pages to improve their ranks. Leveraging file system information, being a deterministic approach, needs to be run only once for each phone.

We tried a combination of all three ranking features, but found that it does not provide substantial improvement over the combination of *a priori* and text quality. We suspect this lack of improvement is due to our evaluation methodology wherein all simulated *a priori* knowledge was on the phone. In a real-world scenario, it is possible that only a subset of the investigator's *a priori* knowledge appears in the recovered fields.

4.4.1.5 Filtering

We filtered the parsable Yaffs images to only include the expired pages; see Section 4.5 for details. For the other phones, we used block hash filtering, which cannot distinguish between current and expired pages.

4.4.1.6 Normalized Discounted Cumulative Gain

We use *normalized discounted cumulative gain* (NDCG) [74] to measure the effectiveness of LIFTR's sorting. An NDCG score of 1 means an ideal ranking, that is, all of the top *k* pages are relevant. An NDCG score of 0 means the worst possible ranking, where none of the top *k* pages are relevant. Here, *k* is a *cutoff rank*, which depends on the expected number of relevant results and on how many top-ranked results would the user be willing to sift through. More formally, the discounted cumulative gain at rank *i* is calculated as follows.

$$\mathrm{dcg}(i) = v_1 + \sum_{j=2}^{i} \left(\frac{v_j}{\log_2 j}\right) \tag{4.4}$$

where $v_j = 1$ if the page at rank j is relevant, and 0 otherwise. We obtain the normalized DCG score by dividing the DCG score by the maximum possible score for some cutoff *k*.

$$\mathrm{ndcg}(k) = \frac{\mathrm{dcg}(k)}{1 + \sum_{j=2}^{k} (1/\log_2 j)} \tag{4.5}$$

Order matters for NDCG. For example, given a ranked list A, where only the top half is relevant, and another ranked list B, where only the bottom half is relevant, then $\mathrm{ndcg}(A) > \mathrm{ndcg}(B)$, assuming A and B are the same length.

4.4.2 Impact of Initial Sorting and Relevance Feedback

Figures 4.2a, 4.2b, and 4.2c show the NDCG results for LIFTR using DEC0DE, `Strings`, and Bulk Extractor, respectively, averaged over all phones.

4.4.2.1 Initial Scoring

DEC0DE, Bulk Extractor, and `Strings` all produce unsorted results: their NDCG scores are near to zero. LIFTR's initial sorting is significantly better. Using 5 tokens of *a priori* information, provides a significant benefit for LIFTR's initial sorting stage, resulting in an average NDCG of 0.73 across all three recovery engines for $k = 50$. The NDCG for initial sorting is denoted by the "Initial" tick mark on the far left side of x-axes in Figures 4.2a, 4.2c, and 4.2b. Initial sorting with file system information gives an average NDCG of 0.43, for $k = 50$, across the three recovery engines. This discrepancy is due in part to the number of pages that belong to the contacts and SMS databases that are not actually relevant. For example, many of these pages contain schema related information and other database metadata. Giving a higher initial score to contacts and SMS databases has the effect of also wrongly benefiting such irrelevant pages belonging to these files.

4.4.2.2 Relevance Feedback

The experiments demonstrate that the relevance feedback stage results in a significant improvement in the NDCG scores with a small amount of feedback. For example, after labeling just 5 pages, the score for the *a priori* scenario rises from an initial NDCG value of 0.73 to 0.88, for $k = 50$; and from 0.48 to 0.71, for $k = 1000$.

The graphs for DEC0DE and `Strings` also depict the differences in increases of the NDCG score when given more feedback, for different cutoff ranks. There is a steeper increase for $k = 1000$ as compared to $k = 20$ or $k = 50$. This shows that the initial ranking algorithm is able to fill up the top few ranks with relevant pages (high NDCG at initial for $k = 20$), however the initial sorting incorrectly ranks a significant number of irrelevant pages over relevant pages at ranks above 50. We see that the relevance feedback quickly corrects these mistakes (steep increase in NDCG for $k = 1000$). This demonstrates the importance of relevance feedback in LIFTR. At the same time, it also establishes the need for having initial ranking for bootstrapping the feedback stage. A few relevant pages in the early ranks greatly increases the pace at which LIFTR discovers other good pages, and helps LIFTR ignore the large pool of bad ones.

4.4.2.3 Variation

Each line in Figures 4.2a, 4.2c, and 4.2b is an average of all phones for the given experiment. The average of all NDCG scores allows us to aggregate many experiments into these three figures. There is variation among phones that is hidden by the average NDCG, but it is not representative of the variation within each phone. Figure 4.3 shows the per-phone NDCG values for DEC0DE using the *a priori* approach, for a cutoff of $k = 1000$ (i.e., the blue line in the left plot of Figures 4.2a). Error bars show 95% confidence intervals over 30 trials, each using a randomly selected set of 5 tokens. For ten of the phones, relevance feedback always improves the NDCG. For three phones, the relevance feedback approach provides no advantage.

4.4.2.4 Discussion

The average NDCG score for all approaches and settings increases with feedback but does not reach its maximum value of 1. There are two possible reasons for this limitation. First, there are certain relevant pages on the phone that are not semantically related to other relevant pages and so no amount of feedback can guide LIFTR in identifying such pages that are sitting by themselves. Second, the NDCG score for a cutoff of k is not only affected by the number of relevant results among the top k, but also the positions at which they appear.

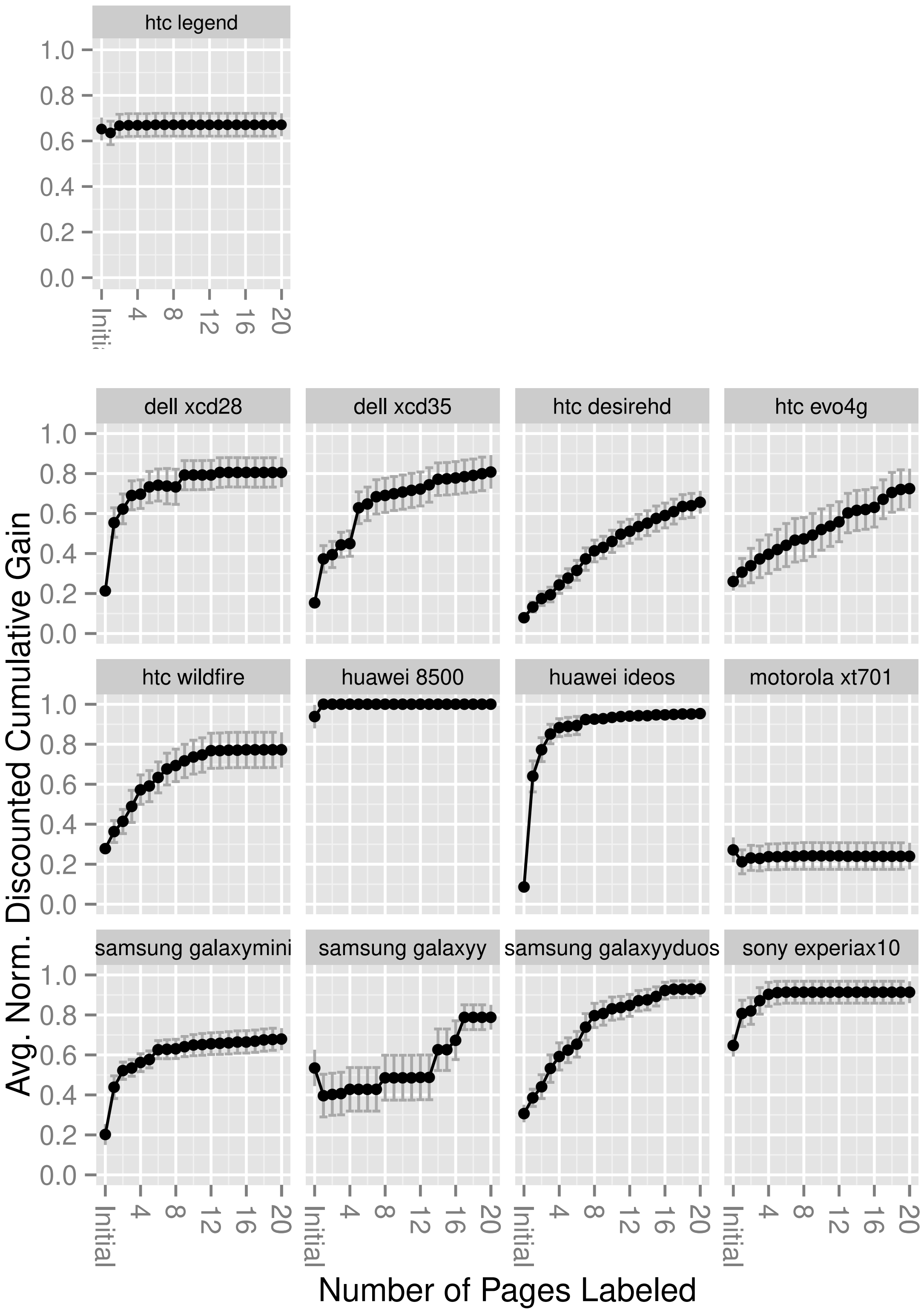


Figure 4.3: Per-phone average NDCG results for DEC0DE using the *a priori* approach, for a cutoff of $k = 1000$. Error bars show 95% c.i. across 30 trials per point. For most of the phones, relevance feedback consistently improves the NDCG score.

Even if LIFTR is able to rank the relevant pages among the top *k* positions, unless all of those pages appear before all irrelevant pages, the score will be less than unity.

It is also important to note that the pages recovered during relevance feedback are not limited to those belonging to contacts and SMS databases. They could include pages associated with other files that have relevant tokens. For instance, some phones have user data in the SQLite Write Ahead Log file, which is like a rollback journal used by SQLite for atomic commits and transaction rollbacks. This shows that simply extracting contacts and SMS database files would not yield all the relevant data on the phone.

It is also interesting to note that the NDCG plot for Bulk Extractor is not as smooth as DEC0DE or `Strings`, due to the fact that only a few phones have have more than 50 pages and fewer have more than a 1000 pages among the inference results after using Bulk Extractor. Hence, the NDCG values are averaged across fewer phones.

4.4.3 Measuring Investigator Work

The intuition behind LIFTR's feedback approach is that an investigator can perform a small amount of manual analysis to greatly improve the overall quality of the returned results.

For our evaluation, we measure investigator work in terms of the number of pages that they must manually label. Because a single page may require multiple labels, we differentiate between *(i)* the total number of pages that the investigator labels and *(ii)* the number of fields that the investigator must label. Recall from Section 4.3.3 that an investigator need only label the positive fields, as LIFTR will default to marking everything it has shown the investigator as a false positive. On average, this approach leads to 7 labels per relevant page. If we assume that it takes an investigator 5 seconds marking a label, then it would take him around 35 seconds per page, and around 11 minutes to complete 20 pages. As our results for the three recovery engines suggest, 20 pages is sufficient for most phones to achieve an NDCG of at least 0.8.

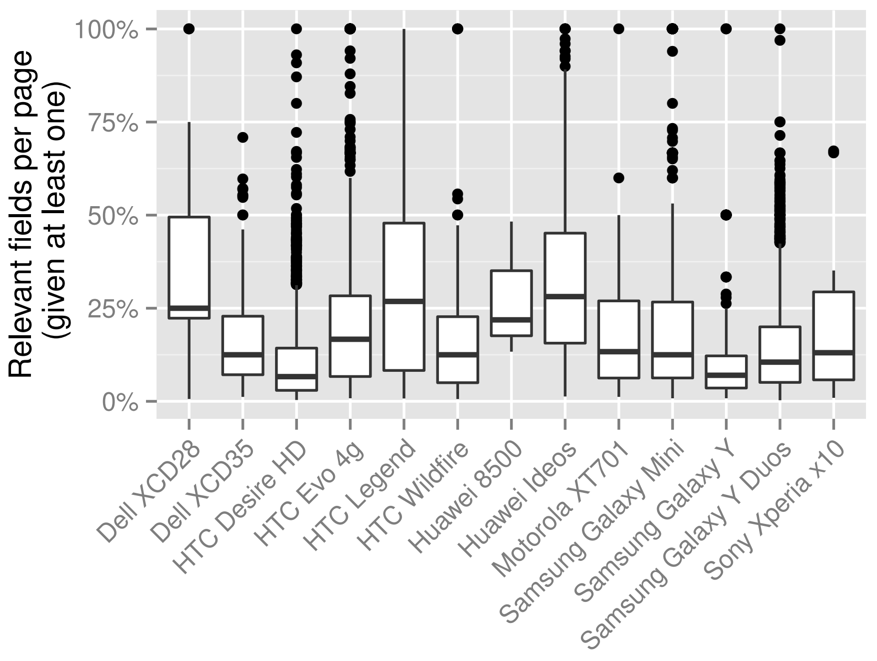


Figure 4.4: The percentage of relevant fields averaged across all relevant pages. The majority of fields returned by the recovery engines — even for relevant pages — are false positives.

We examined another approach where the investigator marked whole pages as relevant or not, rather than the individual fields on each page. This alternate approach did not perform as well. Figure 4.4 shows why: among pages with at least one relevant field, typically less than half of the fields on the page are relevant.

4.4.4 Strongly Related Pages

Our technique is most effective when the investigator provides positive labels to pages that are semantically well connected to other pages, owing to the co-occurrence of relevant tokens among them.

In order for relevance feedback to be effective within *i* pages of feedback, the initial sorting must include a relevant page in the top *i* result pages. We find that it is more effective to bring up true positive pages than it is to penalize false positive ones. This is largely because irrelevant pages greatly outnumber relevant ones; 20 pages of feedback are not sufficient for identifying most of the irrelevant pages or tokens.

4.5 Residual Data in Android

In this section, we detail our techniques for parsing and analyzing the Yaffs file system. We use these techniques with LIFTR to more effectively filter the physical images, and to provide the file labels for initial sorting.

We also show that significant amounts of data, including user information, persists in the expired segments of the phone's memory for periods as long as weeks or months. Further, this expired data makes up more than half of the NAND pages for some phones. Since this data cannot typically be recovered by parsers that aim at reconstructing the file system, file system agnostic inference techniques used by recovery engines like DEC0DE and Bulk Extractor are a potential solution to the problem. The sheer amount of irrelevant results returned by these engines, however, highlights the importance of using LIFTR for quickly locating information with evidentiary utility, from these results.

NAND flash does not overwrite data in place. Instead, when an object is modified, the data is written to a new page, leaving the expired chunk in storage. These expired pages persist for an indefinite period of time. This residual data offers interesting opportunities for forensic triage, as the old pages contain deleted data and can potentially be used to track the changes over time. Our results are consistent with previous work in secure data deletion on flash memory [124,125,155].

4.5.1 Yaffs Overview

Yaffs, common on Android phones prior to Gingerbread, is a log-structured file system designed to work with NAND storage. The file system treats everything as an object or piece of an object. Each Yaffs object is stored as a sequence of *chunks*. Typically, a chunk is equivalent to a NAND page. Yaffs uses *header chunks*, to store object metadata such as the object type, timestamps, and permissions, and *data chunks* to store the actual bytes of a file. Non-file objects such as directories and hard-links, are made up of just a single (header) chunk. In Yaffs, chunks are the unit of write and *blocks* are the unit of erasure. Blocks are made up of contiguous chunks in memory, often 64 chunks to a block.

Chunks may be in one of three states: erased, expired, or current. A chunk is considered *erased* if it resides on an erased block, that is, the block is devoid of any data and contains all `0xFF` values. In contrast, *expired* chunks contain once-valid data that has been replaced by a more recent chunk. These chunks either belong to deleted objects or are old pieces of current objects. Both erased and expired chunks are ignored by the file system.

Finally, *current* chunks contain the most recent version of each object chunk. Yaffs keeps track of all of the current chunks and uses them to reconstruct all objects in the file system.

When a file is modified, one or more chunks transition from the current state to expired. Eventually, through a garbage collection process, Yaffs will reclaim storage space by erasing an entire block of expired chunks. Recall that the unit of erasure in NAND is a block. However, it is common for expired and current chunks to reside on the same block.

Upon phone startup, Yaffs scans through all of the chunks to reconstruct the file system state. In order to determine which chunks are current and which are expired, Yaffs uses the *out-of-bounds area* (OOB) — a small region of memory adjacent to each page — to store file system metadata. This metadata includes the *block sequence number* which is assigned when a block is picked for writing. Because Yaffs always fills the current block before moving to the next block (in absence of power loss or shutdown), the block sequence number, and the order of chunks within the block, represents a temporal ordering: the higher the sequence number, the more recently the block was written. Note that blocks with consecutive sequence numbers may not be physically adjacent in memory.

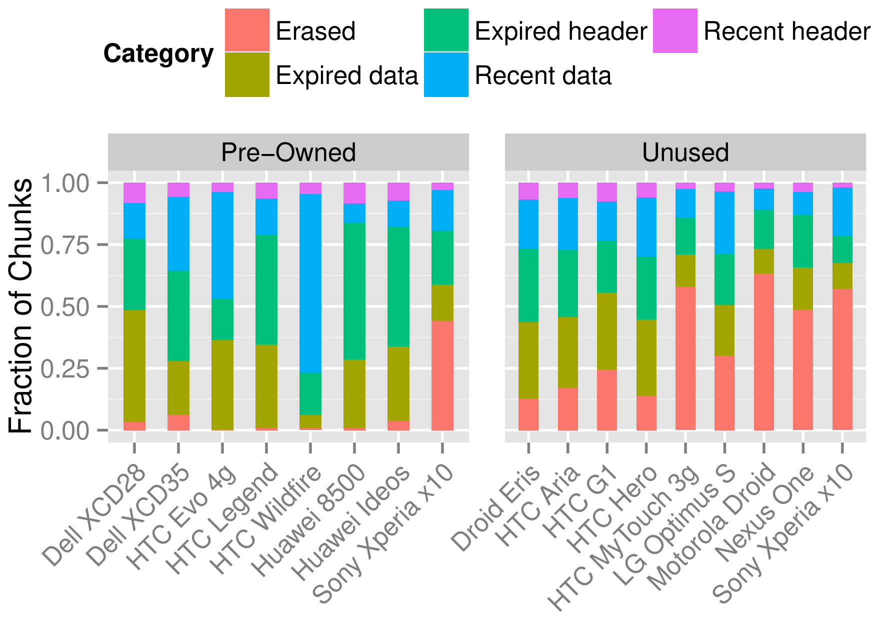


Figure 4.5: Fraction of each chunk type for the user data partition. On the pre-owned phones, over half (56%) of all storage belongs to expired memory chunks.

4.5.2 Breakdown of Chunk Composition

A significant portion of the phones consists of residual data, with 56% on average for the set of pre-owned phones. Figure 4.5 shows the fraction of different chunk types for a set of Android phones. As we discussed above, Yaffs chunks are either headers or data and are always in one of three states: erased, expired, or current. This gives us the 5 chunk categories shown in Figure 4.5.

The phone set includes 8 of the pre-owned phones from Section 4.4 (those with a parseable Yaffs user data partition), in addition to 9 unused phones loaded with synthetic data[3]. We limit our analysis to the user data partition of each phone.

On average across both the unused and pre-owned phones, roughly 26% of the chunks are current, with 27% erased and the remaining 47% expired data. Pre-owned phones have a

[3]We obtained the 9 phones with synthetic data from Via Forensics: `https://viaforensics.com/`.

greater fraction of current chunks than the unused phones: an average of 31% versus 20%, respectively. And pre-owned phones have fewer erased chunks than unused phones: an average of 12% versus 43%, respectively. The difference in erased chunks is due to the garbage collection process. Garbage collection is expensive and typically only erases blocks as needed.

By definition, each object may only have one current header chunk and, if it is a file object, one or more current data chunks. On average across the used phones, current headers make up only 5% of the chunks whereas current data chunks make up roughly 26%. Interestingly, the ratio of header to data chunks is significantly higher in the expired data. For the pre-owned phones, expired headers make up 30% and expired data chunks make up 27%. Anytime an object is modified, whether it be the object's metadata or actual contents, Yaffs will write a new header chunk. In this way, expired headers track changes to an object over time.

Much of the storage space is dedicated to SQLite databases and associated files. The majority of the expired chunks belong to deleted objects, typically SQLite temporary files such as the journal files used for rollbacks.

4.5.3 Filtering Images Using Yaffs

Recovery engines such as Bulk Extractor or DEC0DE are designed to be able to recover information from deleted (i.e., expired) portions of an image. With information from Yaffs metadata, we can more effectively filter the phone image to focus on the expired chunks. We implemented a parser to identify the unique set of expired data chunks, and for the 8 pre-owned phones, we were able to reduce the size of the images by an average of 76% for our experiments in Section 4.4. Note that expired header chunks contain only outdated metadata, rather than the type of information the inference engines and investigators target. Because the header data and allocated chunks are interleaved with expired data, a technique that is blind to the file system, such as block hash filtering would not be able single out the expired chunks.

4.5.4 Block Churn

Our analysis of the 8 pre-owned phones showed that blocks remain in memory for weeks or months after originally written. Any expired chunks in those blocks are accessible via DEC0DE or other recovery engines. We describe our findings and the process for estimating block age below.

The block sequence numbers provide a temporal ordering of blocks in the file system; the block with the highest sequence number was written most recently while lowest number block is the oldest. However, the sequence numbers do not directly tell us the exact write time of a block.

More precisely, a block's *write period* is a time range between when the first and last chunks were written to the block. The block write period is useful for *(i)* providing a bound on when expired chunks were valid and *(ii)* as a means for estimating the rate at which blocks are written.

Header chunks contain object timestamps. If a block contains a header, we can use that header to help estimate the block's write period. If a block does not contain a header, we can use the time estimates for adjacent blocks (by sequence number).

At a high level, we employ the following algorithm to estimate the write period for each block. First, for each block we record the object timestamps stored in any header chunks that are present on the block. Header chunks are written anytime an object is created or modified, and contain three timestamps: `atime`, `mtime`, and `ctime`. Unlike the Unix equivalents, `atime` does store last access time by default. Typically, `atime` stores the creation time of the object. The write time for the header chunk is then the greater of either the `ctime` or `mtime`, ignoring the effects of caching which appear to be negligible in practice. The block's write time is bounded by the oldest and most recent header write times.

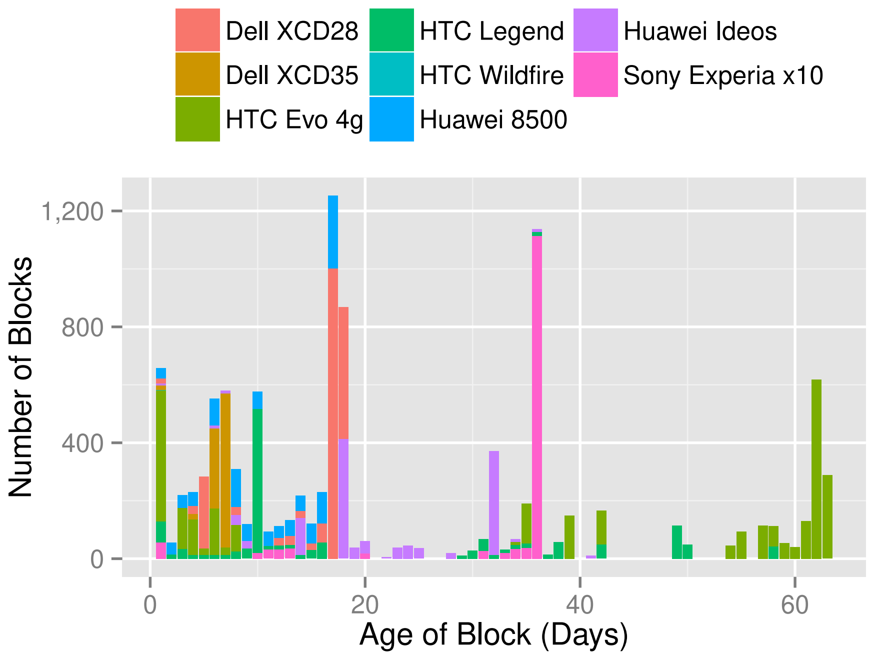


Figure 4.6: The age of blocks, in days, for the user data partition of pre-owned phones. Blocks remain in memory for weeks or months after originally written, with half older than 14 days and a quarter older than 34 days.

When a block does not contain any header chunks, we have to estimate the write period using adjacent blocks. For example, consider blocks b_1, b_2, b_3 with sequence numbers 1, 2, and 3 respectively. Assuming, we already have write period estimates for b_1 and b_3 (calculated using the header chunks). The sequence numbers give us the write order for the three blocks. In other words, b_2 must have been written after b_1 but before b_3. Therefore, the write period of b_2 must be between the last write of b_1 and the first write of b_3.

In practice, we observe that some header chunks used seemingly inconsistent timestamps, e.g., some headers stored a modification time that was before the object's creation time. This is due in part to garbage collection, as we discuss further below. To avoid this issue, we only use timestamps from newly created objects. Focusing solely on new objects does not represent a limitation as phones frequently create new and temporary files. We can find newly created objects by looking for expired headers such that the header is the oldest chunk found for an object and the number of bytes listed in the OOB field is zero.

Figure 4.6 shows the block ages for the pre-owned phones. We calculated the block age relative to the age of the most recent block of each phone. For the HTC Evo 4g 37% of the blocks were over 60 days old. Across all phones, half of all blocks were older than 14 days, and a quarter were older than 34 days.

4.5.5 Garbage Collection

Yaffs periodically copies current chunks to new blocks, freeing the original block for deletion. These chunks are copied exactly, so any moved header chunks will retain their old timestamps. Consequently, chunks moved due to garbage collection may appear to be newer than they actually are. For example, imagine the most extreme case in which a data chunk is written early in the phone's life, but always remains current because the file it belongs to never changes. Over time that chunk will be moved to new blocks as the result of garbage collection with each subsequent block having a higher sequence number. In other words, there may be a significant difference between when a chunk was written to a given block and when the chunk was originally written.

Because blocks sequence numbers are always incremented, we can estimate the rate of block deletion and garbage collection by looking at the missing sequence numbers in the image.

4.5.6 Inferring Yaffs Parameters

Before we can analyze the Yaff image, we have to infer the important Yaffs parameters that may be different for each phone. For each phone image we need to infer the block size, chunk size, OOB size, and OOB tag offset. Knowing the OOB tag offset enables us to parse the Yaffs metadata store in the OOB, e.g., the block sequence number and object identifier.

This OOB offset is not directly controlled by Yaffs. We can quickly determine this offset by comparing the OOBs for different headers of the same file object. For the same object, the object identifier should remain consistent across different header chunks. This approach relies on file objects that are unlikely to have been deleted, e.g., the contacts database.

To estimate the Yaffs parameters, we scan the image for header chunks belonging to a file known to be on the image. For example, the `contacts2.db` file is present on most Android phones. We repeat the scan assuming a variety of different parameter values. The most likely parameters are those that produce the most valid file headers. In practice, all of the phones we examined used a chunk size of 2048 bytes, an OOB of 64 bytes, and a block size of 64 pages; however, we saw at least three different OOB offset values.

4.6 Related Work

In addition to DEC0DE and Bulk Extractor, LIFTR is related to prior work in acquiring physical images from mobile phones [123,147,150] and Android file system analysis [47, 137].

Park et al. [114] propose a technique for clustering and recovering fragmented SQLite records of the same file residing in expired pages, without parsing or recreating the underlying file system. LIFTR could be used to augment their technique.

Beebe et al. [7] implement an unsupervised learning algorithm for clustering results from simple text search queries on raw data, using self-organizing neural networks. In contrast, LIFTR is a supervised approach, and would complement their mechanisms.

Foster [51] proposes a file system independent technique, sector hashing, for identifying if a target file was ever present on a storage device. The work does not target user-generated information.

Marturana et al. [100,101] use machine learning models to determine if copyright-infringing data or child abuse materials are present on a mobile device. Such models are more difficult to train in a general setting, where the investigator is interested in user-specific details which typically lack distinctive features.

4.7 Summary and Conclusions

We proposed the use of relevance feedback to quickly pinpoint information relevant to an investigation. Our system, LIFTR, addresses a major issue limiting current forensic triage techniques: only a small fraction of the information returned by recovery engines is relevant to an investigation.

When applied to smart phones, the recovery engines returned hundreds of thousands of results, most of which were irrelevant. LIFTR overcomes this limitation by finding those pages that are most likely to contain useful information, getting feedback from the investigator, and using that information to rank the results. Further, we show how a small amount of background information about a suspect can greatly improve LIFTR's performance.

Our evaluation was performed using Android phones from a variety of makes and models. We tested LIFTR with three different recovery engines, each differing significantly from the others in its inference approach. Our results demonstrate that feedback on as few 20 NAND pages is more than sufficient to identify the top 100 most relevant pages out of the hundreds of thousands of false positives returned by the recovery engine.

CHAPTER 5
CP TRAFFICKING ON P2P NETWORKS

In Chapters 3 and 4 we discussed the first of two important forensic scenarios: mobile phone triage. In this chapter, and in Chapter 6, we focus our attention on another: network-based investigations.

Law enforcement has limited resources to investigate and prosecute crimes. Unfortunately, the extent of criminal activity often exceeds police capabilities. Such is the case with contraband trafficking on peer-to-peer networks.

5.1 Introduction

Peer-to-peer (p2p) networks are the most popular mechanism for the criminal acquisition and distribution of *child sexual exploitation imagery*, commonly known as *child pornography* (CP)[1]. Investigating CP trafficking online is critical to law enforcement because it is the only effective *proactive*[2] method of finding persons, known as *contact offenders*, who directly and physically abuse children. A previous study found that 16% of investigations of CP possession ended with discovering contact offenders [157]. These investigations have two primary goals: to stop the distribution of CP; and to catch child molesters and help children that are being victimized, often by family members.

Numerous studies of p2p networks have explored the availability, performance, and traffic characteristics of file sharing. Unfortunately, no study of copyrighted movies or

[1]These are not "sexting" crimes by late teens: 21% of CP possessors have images depicting sexual violence to children such as bondage, rape, and torture; 28% have images of children younger than 3 years old [157].

[2]This method is proactive in that law enforcement is not waiting for someone, a third-party or the victim (if old enough to speak), to come forward and report the abuse.

music provides any assistance to law enforcement seeking to arrest CP perpetrators, discover sexually abused children, or inhibit the trade of images of exploitation. These past works are neither performed within nor evaluated under the constraints and goals of criminal investigations. The study we present in this chapter is based on methodology sufficient for court scrutiny[3], makes specific recommendations for law enforcement strategy, and provides an empirical characterization suitable for goals ranging from informing sentencing hearings to setting national enforcement priorities.

The fundamental problem faced daily by CP investigators is *triage*[4]. Over a one year period, we observed over 1.8 million distinct peers on the eMule p2p network and over 700,000 peers on Gnutella, from over 100 countries, sharing hundreds of thousands of files verified as CP. We observed that the majority of CP files are shared by a relatively small set of aggressive users, but a smaller set of files are shared so redundantly that their daily availability is guaranteed. While most CP files are only available for a short amount of time (only about 30% are available for more than 10 days of the year), there are at least tens of thousands of unique CP files available on p2p networks for download each day. These quantities cannot be addressed by investigators in an ad hoc fashion. In other words, investigators need clear and effective strategies for prioritizing their limited resources.

Accordingly, we examine *(i)* methods of target selection designed to reduce content availability (an NP-hard problem); and *(ii)* an empirical justification for focusing on sub-groups of peers that are the most aggressive, in terms of their duration and scope of activity, volume of shared content, or attempts to escape attribution. Our key contributions are as follows.

- We propose and evaluate three strategies for prioritizing law enforcement resources in investigating CP trafficking. We conclude that removing peers with the largest

[3]The data in this study formed the basis of 2,227 search warrant affidavits.

[4]See Chapters 3 and 4 for a discussion of triage in the context of mobile phones.

contributions (a weighted measure of days of uptime and files made available) is most effective, but with Pareto-like diminishing returns.

- We examine subgroups of aggressive peers, such as peers seen using the Tor anonymity system, peers on multiple p2p networks, and four other subgroups. We find that all appear to be more active in their trafficking, having more CP files and more uptime than the average peer sharing CP. These aggressive subgroups deserve priority in investigation over millions of other potential targets.
- We find that offenders using Tor use it inconsistently. Over 60% of linkable user sessions send traffic from non-Tor IPs at least once after first using Tor, thus removing its protection; over 90% of sessions observed on three or more days fail likewise. This result not only speaks broadly to the failure of Tor in practice; it shows that, fortunately, investigators need not employ complicated, theoretical attacks on Tor users that share CP.
- We examine our dataset for evidence of users purposely aliasing as multiple peers on the network to hide their actions. We find little evidence, on a day-to-day basis, to suggest that users are changing their application-level identifiers but keeping their libraries.

Our findings are based on a partnership with over 2,000 U.S. law enforcement officers nationwide to collect data on CP trafficking. To enable our study, our group built several tools for conducting forensically valid investigation of these crimes. All investigators completed multi-day trainings on the tools, and collected evidence with the expectation that a court would examine the data. These tools are now in daily use in all U.S. states and several other countries. As a result, thousands of persons, many of whom had directly abused children, have been arrested for these crimes.

5.2 Criminal Investigation

The highest impact research in criminal forensics works within, and is evaluated under, the constraints and goals of investigations. In this chapter, we follow that principle, rather than presenting a set of isolated, exploratory characterizations of users. See Section 2.2 for a review of the U.S. legal constraints on criminal investigation.

Arrests in these criminal cases are typically not based on the network-acquired evidence. They are based on the fruits of the search and the person identified as possessing the contraband materials. There is no notion of a false positive of a person during a search; only that the search itself was barren of evidence. Investigators and warrant-issuing judges prefer to minimize as much as possible the number of such ineffectual searches, but they are not impermissible.

Because of pre-warrant restrictions, there is nothing in our data collection methodology that is secret or hidden. We have no special agreement with law enforcement, other than their identification, by hash value, of known files of interest. We log and analyze only what is available to any member of the public and research community.

Finally, we note that our work follows a forensics model and not the traditional security attacker model. Our techniques can be applied very successfully even though there exist many ways to defeat them. But as we show, many people do not attempt to hide. We identified over 1.8 million eMule GUIDs (globally unique identifiers), with many sharing multiple CP files. Each such shared file matches a list of known CP, identified by a cryptographic hash. Not one of the 1.8 million took the time to change a single bit of the file and thus the hash. Why? We can only speculate, but changing the file hash serves little purpose when the file names already contain sexually explicit terms, intentionally named for easy discovery by other peers.

5.3 Forensic Measurement

This study is based upon the analysis of a large number of observations of CP files on p2p networks, and the behavior of the peers that share them. In this section, we describe the

Network	Date Range	Files	GUIDs	Records
Gnutella (FOI only)	10/1/2010 – 9/18/2011	139,604	775,941	870,134,671
Gnutella Browse	6/1/2009 – 9/18/2011	87,506,518	570,206	434,849,112
eMule (FOI only)	10/1/2010 – 9/18/2011	29,458	1,895,804	133,925,130
IRC (no file data)	6/2/2011 – 9/18/2011	N/A	N/A	7,272,739
Ares (no file data)	5/31/2011 – 9/18/2011	N/A	N/A	17,706,744

Table 5.1: All datasets are observations of CP activity only, but IRC and Ares data do not contain information about files or GUIDs. Except when otherwise stated, a record corresponds to a law enforcement observation and contains date, time, IP address, application-level identifier, geographic location as determined by an IP geolocation database, and a file hash.

sources of these datasets and provide salient details relevant to our analysis. In Section 5.7, we identify sources of bias in the data and potential limitations of our study.

Most previous studies of p2p networks have taken place over just several days [72,73], several weeks [84], or a few months [63,102,121,139]; our study is comprised of thousands of observations per day for a full year. This duration is especially critical in the context of criminal investigations; scientific studies of crime are often submitted as supporting facts during trial and sentencing.

Our focus is on *files of interest* (FOI), which include CP images, as well as stories, child erotica, and other collections that are strongly associated with this crime. We logged only content with hash values matching, by cryptographic hash, a list put together by law enforcement by visual inspection.

5.3.1 Background

This chapter is based on data collected with the help of national and international law enforcement. Starting in January 2009, our group began deploying a set of forensic tools to investigators in the U.S. and internationally for online investigation of p2p CP trafficking.

Prior to our collaborative efforts, the standard method for online CP investigation was to make isolated cases: leads were not shared among agencies or officers, other than by phone or email. Officers leveraged their own experience to prioritize suspects.

5.3.2 Tools

The suite of tools built by our group, called *RoundUp* [91], has enabled seamless sharing of public view observations of online CP and associated activities on various filesharing networks. The shared data, collected in order to make these cases, provide each investigator with a longitudinal view of CP offenders and provide a method of triage for selecting targets for further investigation; and of course, the data enable this study. Because over 2,000 investigators have been trained on our tool to date, and because it is in use by hundreds of investigators daily, the aggregate set of observations we have used for this study is incredibly detailed. The tools are still in use, and currently, law enforcement execute approximately 150 search warrants nationwide per month based on data collected using our tools. We do not, however, present search warrant or arrest data in this study[5].

5.3.3 Datasets

Our datasets, summarized in Table 5.1, include law enforcement observations from Gnutella and eMule p2p networks. The Gnutella and eMule datasets span a one-year period from October 1, 2010 to September 18, 2011. Each record in these datasets corresponds to a law enforcement observation of a particular peer making available one or more FOI, and minimally contains date, time, IP address, application-level identifier, geographic location as determined by an IP geolocation database, and a file hash.

Most file sharing protocols include an application-level identifier unique to an installation of the application. In both Gnutella and eMule, these identifiers are persistent across users' sessions, and are referred to as *GUIDs* (globally unique identifiers). Peers on these networks are uniquely identified by their GUID, and we use peer and GUID interchangeably to identify unique running instances of the corresponding p2p software.

All FOI are uniquely identified using hash values; law enforcement manually classify files as FOI from a variety of sources, such as post-arrest forensic analyses. An enormous

[5]Our study's procedures were approved by our Institutional Review Boards.

number of such FOI are shared on Gnutella and eMule. Respectively, there are 139,604 and 29,458 known FOI shared by 775,941 and 1,895,804 GUIDs. Our tool searched only for FOI in a list containing about 384,000 entries; this list was updated several times over the course of this study. It is a small sample: the National Center for Missing and Exploited Children reports reviewing more than 60 million child pornography images and videos[6]. As such, our work presents only a lower bound on the amount of activity present in these networks.

In a limited fashion, we use two other datasets. Our IRC dataset, based on a more recent tool that we developed, covers a four-month period from June to September 2011. The IRC dataset is a log of IP addresses that were involved in public activity related to the sexual exploitation of children in public chatrooms; no file observations are in this dataset. We also use a dataset of CP-related activity on the Ares p2p network[7] collected using a tool we did not write, but collected by the same law enforcement officers responsible for all data in this chapter. The Ares dataset contains only IP addresses and has no information about files shared, but addresses were only logged for peers that shared known FOI.

5.3.4 Other Details

Gnutella allows a peer to be *browsed* and thus investigators can enumerate all files shared by peers. Our *Gnutella Browse* dataset consists entirely of peer browses and includes all files a peer is sharing, not just FOI. Some Gnutella peers cannot be browsed; we collected only FOI data from these peers. eMule does not permit browses. Regardless, each of these datasets includes only peers that share one or more FOI; peers without FOI were not logged.

We draw a distinction between a time-limited view of a peer's shared files and the set of all files with which a given peer was ever observed. We define a GUID's *library* to be the set of files that were observed being shared by that GUID on a given day. A GUID's

[6]See `http://www.missingkids.com/missingkids/servlet/NewsEventServlet?&PageId=4604`.

[7]`http://aresgalaxy.sourceforge.net/`

corpus is the set of all files shared by that GUID over the entire duration of the study. In both cases, we typically only include FOI, but we make it clear when a corpus or library includes non-FOI observed as the result of a browse.

5.4 Availability and Resilience

In investigating the trafficking of CP on p2p networks, the goal of law enforcement is to prioritize criminals whose arrest will have the greatest impact. But the strategy to achieve this goal depends upon the impact desired: finding contact offenders who go otherwise unreported, finding those who create new CP, and decreasing the availability of FOI on the network are all priorities. In this section, we focus on strategies for reducing the availability of FOI.

Effective CP removal strategies are especially important as a means to prioritize law enforcement's limited resources and time. After online evidence is collected, days or weeks of off-line processes are required in each case until an arrest is made. Additional resources are required to go to trial. It is infeasible for investigators to arrest all users sharing CP and remove all FOI. Investigators need a triage strategy for deciding upon which small fraction of online leads to act.

An enormous set of perpetrators are active every day around the world. Even with unlimited resources, U.S. law enforcement can only partially impact file availability. Our results, discussed below, suggest the need for a coordinated international effort.

5.4.1 FOI Redundancy and Availability

Before we further discuss the implications of removing files, we characterize the redundancy and availability of FOI on Gnutella and eMule.

5.4.1.1 File Redundancy Across GUIDs

Many FOI on Gnutella and eMule are not widely redundant among GUIDs within the same network. Figure 5.1 shows the relative *redundancy* of FOI, which is the number

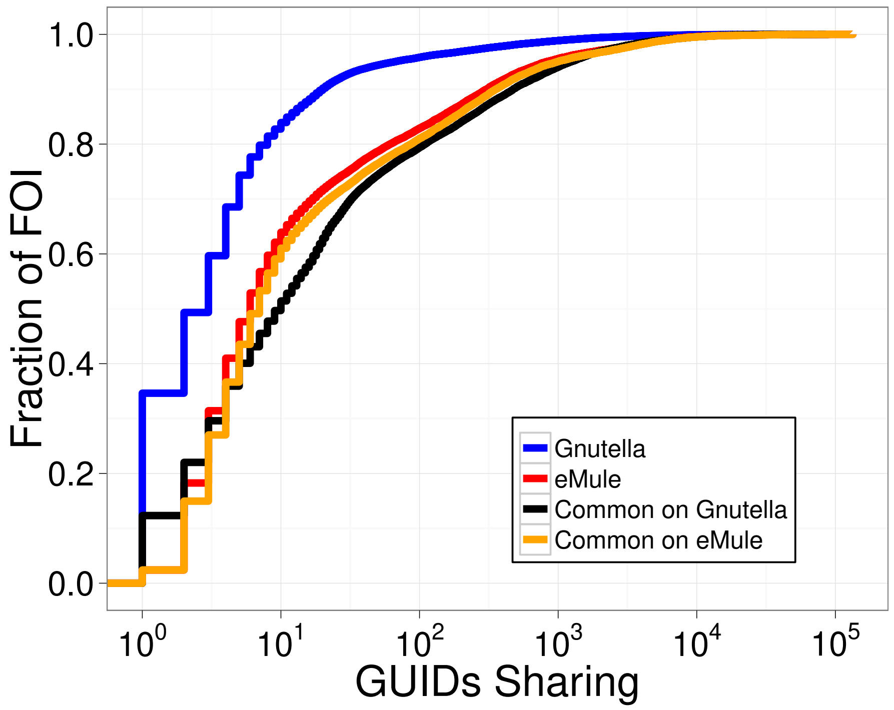


Figure 5.1: Redundancy of FOI (files of interest) among multiple GUIDs as a CDF. Some files are seen on both networks, but the distribution of these observations is different. The "Common on Gnutella" line shows the CDF of these common files as seen on Gnutella, and similarly for the "Common on eMule" line.

of GUIDs that possess and make available each file. The distribution is presented as a cumulative distribution function (CDF), which shows on the *y*-axis the fraction of FOI that are shared by *at most x* GUIDs. For example, 90% of files on Gnutella were shared by at most 20 GUIDs; 99% of files were shared by at most 1,167 GUIDs; and 99.9% of files were shared by at most 9,129 GUIDs.

Figure 5.1 also shows the relative redundancy for the subset of FOI appearing on both networks. The set of files common to both networks is significantly more redundantly shared on each network than the set of all files on each network. There is a high degree of FOI overlap among the two networks: 26,136 of the FOI on the eMule network (nearly 89%) were also seen on the Gnutella network, and 97% of Gnutella GUIDs were observed with at least one file that can be found on the eMule network. The overall low redundancy of most files suggests the strategy of prioritizing the investigation of users who possess a

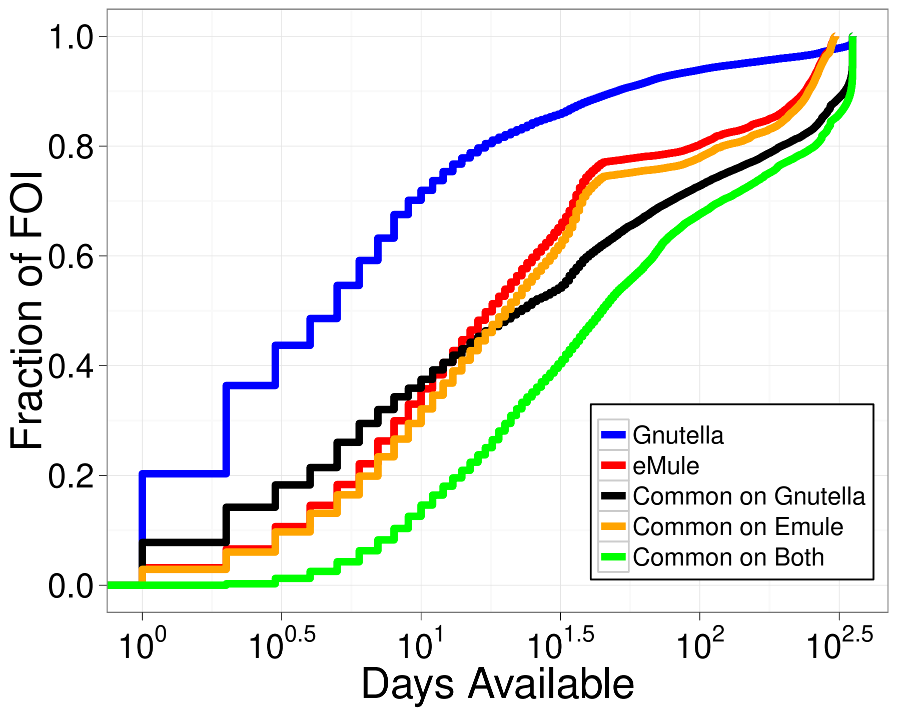


Figure 5.2: CDF showing the days available per FOI (during 353 days for Gnutella and 329 days for eMule). As in Figure 5.1, the "Common on Gnutella" line shows the CDF of files common to both networks as seen on Gnutella, and similarly for the "Common on eMule" line. The "Common on Both" line shows these common files available on either network on any given day.

large amount of less redundant FOI in order to remove it from the network and prevent its proliferation. An easily intuited proxy for this measure is to target GUIDs who possess large corpora. Since most FOI are relatively less redundant, the GUIDs with the largest libraries likely have the most FOI with low redundancy.

5.4.1.2 File Availability Across Days

We say a file is *available* on a given day if at least one peer is sharing that file on that day. This approach is simple in that it does not take into account bandwidth and reachability considerations, which are difficult to measure globally. We do not expect this definition to limit the applicability of our results, as the assumption of high bandwidth and reachability is conservative from the perspective of law enforcement.

Figure 5.2 plots the availability of FOI as a CDF on a semi-log scale. Gnutella files tend to have lower availability than eMule, with 80% of files available for more than one day; about 30% are available for more than 10 days; and about 5% of files are available for more than 100 days. Generally, files that are available for a single day are unique to a specific GUID; files that tend to have longer availability are possessed by many GUIDs, not all of whom are online on a given day. Again we see that the files that are common to both networks are more available than is typical on each individual network: about 30% of these common files are available for more than 100 days. We have also calculated that on a daily basis, an average of 9,712 distinct files are available, with a peak of 32,020 files during our study.

5.4.2 Law Enforcement Strategy

Our *law enforcement model* is as follows. Investigators have a global, historical view of GUIDs and their corpora, including known FOI and other files. Investigators look to reduce FOI *availability*, by arresting the users that correspond to peers and removing their corpora from the network. Investigators aim to remove files from the network completely.

Content can be removed from these networks only by arresting users and taking their shared libraries offline, as the protocols and implementations inhibit falsifying or polluting content (using hash values). Our goal is to find out which peers should be removed such that we minimize the number of files that are available at least one day.

In Section 5.8 we show that this problem is NP-Hard. Here, we evaluate four greedy heuristics aimed at reducing the availability of CP by removing peers. Our evaluation consists of removing subsets of peers from the data and examining the effect on availability. Specifically, we examine the following heuristics: *(i)* removing peers that were *observed* most often, i.e., largest number of days observed; *(ii)* removing peers with the largest *corpus* size; *(iii)* removing peers with the largest *contribution* to availability (as defined below); and *(iv)* removing peers selected randomly, as a baseline. For an arbitrary file on an arbitrary

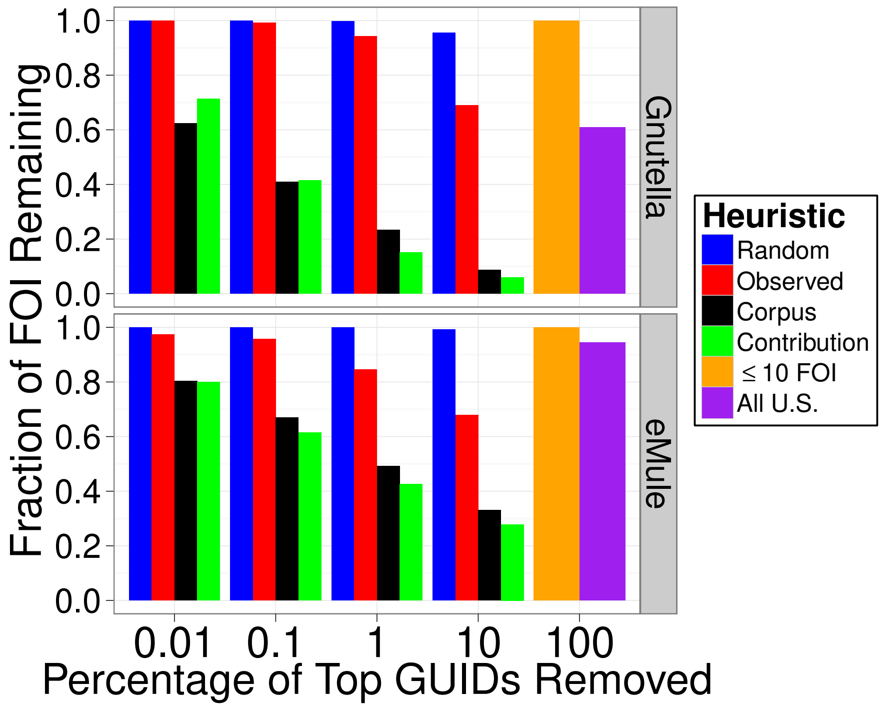


Figure 5.3: The remaining fraction of FOI available at least one day given a percentage of GUIDs removed according to different heuristics: random, number of days observed, corpus size, and contribution to file availability on Gnutella and eMule. We removed the top 0.01, 0.1, 1, and 10 percent of GUIDs according to each heuristic. In Gnutella, the Corpus and Contribution heuristics achieve equal results when 0.113% of GUIDs are removed. Also shown is the impact of removing 100% of peers with 10 or fewer FOI, and 100% of peers in the U.S.

day, n peers possess that file. We say that each peer provides a *file-contribution* of $\frac{1}{n}$th of that file. A peer's *contribution* to file availability is the sum of the file-contributions of the files in their corpus over the duration of the study.

An alternative measure of availability is *daily redundancy*, the number of peers that share a file on a specific day. The algorithm to optimally reduce the maximum redundancy over all files shared is simple: remove the peers with the largest corpus size first. It is unclear that minimizing redundancy, unless it is to zero (equivalent to unavailability), is useful or effective. To evaluate the effect of reducing redundancy to a small value, we would require reachability, bandwidth, and propagation models of the underlying p2p overlays. Thus, we do not consider daily redundancy further.

5.4.2.1 Comparison of the Efficiency of Heuristics

Figure 5.3 compares the effectiveness of each of the above heuristics. Interestingly, removing the peers that were seen the most often has almost no effect on the availability of FOI. Removing peers by either contribution or corpus size is most effective; these measures are correlated, so their similarity in performance is unsurprising.

The vast majority of files are shared only by a relatively small set of prolific GUIDs. Consider Gnutella (similar trends hold for eMule): If we remove the top 0.01% of 775,941 GUIDs as determined by corpus size, only 59% of the known FOI remain available in the network. In other words, 41% of the unique files on the network are made available by a group of only about 80 GUIDs. The top 0.01% have 3,242 distinct FOI on average, with the top peer possessing about 25,000 FOI. Most of these files, however, are only available for a relatively short amount of time; as Figure 5.2 shows, only 28% are available for more than 10 days during our study. Some of this is due to the relatively low number of days these high-contributing GUIDs were observed; this also explains the failure of the observed days heuristic. These prolific GUIDs have a worldwide presence. Removing them requires tremendous multi-national cooperation as we discuss below.

5.4.2.2 Impact of Geography on Availability

Our data are mostly based on the efforts of U.S. law enforcement, and the files they are looking for are arguably tuned to U.S. perpetrators. As law enforcement agents are limited by jurisdiction, the locational diversity of these users provides a resistance to the straightforward approach of prioritizing them. Only a small majority of top Gnutella GUIDs (by corpus size)—57 out of 100—are located in the U.S. The rightmost bar ("All U.S.") in Figure 5.3 shows the reduction in availability if we restrict our removal to U.S. GUIDs (that is, GUIDs with an IP located in the U.S.) only. Note that we remove *all* such GUIDS in our analysis, a clearly infeasible approach in practice. Just 30% of files are unavailable (internationally) after removing all GUIDs in the U.S.; removing just the top 0.01% internationally (a group

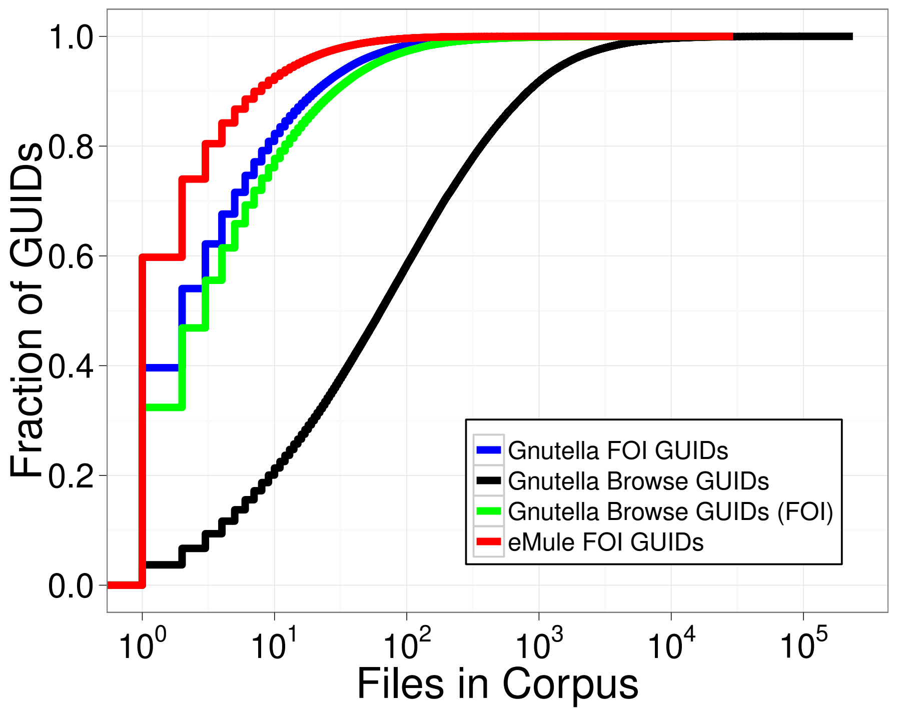


Figure 5.4: CDF showing the corpus size per GUID, for various measurement types. The black line ("Gnutella Browse GUIDs") show the corpus size distribution for all files seen at GUIDs whose libraries were browsed, and the corresponding green line shows the distribution of FOIs within those browses. The other two lines show all FOI observed in any manner. (n.b., eMule does not allow browses.) Most GUIDs have very few files in their corpus.

of about 80 GUIDs) has a similar effect, suggesting the utility of a coordinated international approach.

Within the U.S., the problem is similarly large in scope. The top 5% of GUIDs in the U.S. comprises a set of 14,410 GUIDs, each with a corpus of at least 40 known FOI. Due to the weeks of manual effort required for each arrest, the limited resources in the U.S. allow for 3,100 arrests per year for both offline and online offenses [146].

5.4.2.3 Impact of Low-Sharing GUIDs on Availability

A large portion of GUIDs have comparatively few files. As shown in Figure 5.4, about 82% have 10 or fewer FOI. There are several reasons peers may appear to have few files. They may have files that are CP, but are not yet known to be FOI. They may be downloading

FOI and not subsequently sharing them. They may have downloaded the files incidental to other activities. Finally, they may simply be sharing a smaller library. We expected removal of such low-sharing users to impact file availability significantly, since very many peers possess few files. Contrary to our expectations, removal of these GUIDs sharing few files has essentially no effect on file availability, as shown in the second-rightmost bar in Figure 5.3 ("$\leq$ 10 FOI"). This result provides further evidence that file availability depends primarily on those GUIDs with the largest corpora, though it does not consider the contribution to redundancy that these low-sharing GUIDs provide.

5.5 Comparing Aggressive Peers

In Section 5.4, we show that strategies for removing content from the entire ecosystem must target offenders from all countries. In the absence of a unified effort—and no such collaboration exists—investigators need a triage strategy. In this section, we characterize triage metrics for local investigators. Ideally, investigators would target the most *dangerous* offenders: those that are personally, physically abusing children. Unfortunately, such information is typically not available until months or years after arrest [18,145].

In lieu of that ideal, local investigators can target peers that are the most aggressive offenders: peers that exhibit greater evidence of *intent* [80] beyond the average case, which is an important practical legal concern. This includes peers such as those that are online for the longest duration and share the largest number of FOI. Similarly, investigators may target offenders that are conduits between p2p network communities (e.g., by sharing on both eMule and Gnutella), or offenders that seek to escape detection and justice by using Tor or network relays.

We quantify the activity of six subgroups of aggressive peers sharing FOI. We characterize the contribution of each subgroup to the duration of CP availability and the amount of CP content. The subgroups are: *(i)* the top 10% of GUIDs sharing the largest corpora; *(ii)* the top 10% of GUIDs seen sharing FOI the most number of days; *(iii)* the top 10% of GUIDs

Identifier	Network	
	Gnutella	eMule
All GUIDs	775,941	1,895,804
Multi-Network GUIDS	84,925 (11%)	147,904 (7.8%)
Tor GUIDs	3,666 (0.47%)	16,290 (0.86%)
Tor GUIDs (> 2 days)	2,592 (0.33%)	11,998 (0.63%)
Relayed GUIDs	76,478 (9.9%)	78,223 (4.1%)
Top 10% Observed	84,235 (11%)	190,797 (10%)
Top 10% By Corpus	77,782 (10%)	189,951 (10%)
Top 10% By Contr.	77,595 (10%)	189,581 (10%)

Table 5.2: Sizes of each GUID subgroup. Definitions of each subgroup appear in this section.

ranked by the *contribution* metric defined in Section 5.4.2; *(iv)* the set of GUIDs sharing FOI on at least two p2p networks (linked by IP address); *(v)* GUIDs that use a known Tor exit node; *(vi)* GUIDs sharing FOI that use an IP address that we infer to be a non-Tor relay.

Our results show that all of these subgroups are more active than a group that consists of all peers that we observed. The exception is the subgroup of GUIDs using non-Tor relays, as we explain below. The differences of each subgroup to the set of all GUIDs are significant ($p < 0.001$).

Below we provide characteristics of each subgroup, and details of the behavior of each. For example, we show that GUIDs using Tor to share FOI use Tor irregularly, and therefore their true IP addresses are easily identifiable.

5.5.1 Peer Subgroups

The size of each subgroup is shown in Table 5.2. The size of the top 10% by corpus and observed days subgroups are slightly larger than 10%. This variability is due to ties in the ranked lists of GUIDs. We include all such GUIDs to avoid arbitrary tie-breaking.

5.5.1.1 Top 10% Groupings

Users can actively participate in p2p networks in two primary ways: by contributing a large number of files or a large amount of time. For example, one peer may share 100 files for a single day, and another may share a single file for 100 days. In the first case, the

Network	IP Addresses		
	Total	Private	Tor
Gnutella	3,025,530	32,195	7,357
eMule	5,643,350	1,256	21,025
Ares	1,714,894	225	1,799
IRC	88,658	245	746

Table 5.3: Number of IP addresses per network observed sharing FOI. In the case of IRC, the IP addresses correspond to clients observed in public chat rooms related to child sexual exploitation. The Tor column refers to the number of distinct public IPs where Tor-using GUIDs were seen, including but not limited to known Tor exit nodes.

Networks		IP Addresses Intersection		
A	B	$\%A$	$A \cap B$	$\%B$
Gnutella $\cap$	eMule	6.8%	199,824	3.1%
	IRC	0.1%	3,562	4.1%
	Ares	1.0%	30,596	1.8%
eMule $\cap$	IRC	0.1%	4,654	5.3%
	Ares	0.9%	56,921	3.3%
IRC $\cap$	Ares	2.1%	1,813	0.1%
Intersection of all			308	

Table 5.4: Overlap of IP addresses across multiple networks, excluding Tor IPs and private IPs. A small but significant set of IPs were seen across multiple networks, indicating particularly active users.

content is large but other peers have only a limited time to take advantage. In the second case, the content is small but other peers will find it easier to gain access to the content. It is vital for investigators to address both types of activity; the contribution metric balances these two concerns.

For these reasons, we create three subgroups corresponding to the 10% of GUIDs with the largest corpora of files ($\mathcal{F}$), the 10% with the most days observed online ($\mathcal{D}$), and the top 10% of GUIDs when ranked by the contribution metric ($\mathcal{C}$). There is substantial but not overwhelming overlap among these subgroups. The overlap in Gnutella, as defined by Jaccard's index, $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$, is $J(\mathcal{C},\mathcal{F}) = 0.51$ and $J(\mathcal{C},\mathcal{D}) = 0.28$; the eMule subgroups overlap similarly.

5.5.1.2 Multi-Network Peers

Law enforcement are interested in users that are active on multiple p2p networks. Such users are more aggressive in terms of assisting in the distribution and availability of content to two communities, possibly acting as a bridge. We identify the set of GUIDs in Gnutella that are active in another network by finding all IP addresses in our Gnutella dataset that also appear in any of our eMule, Ares, or IRC datasets, and correspondingly in eMule for those that appear in any of the Gnutella, Ares, or IRC datasets. We refer to GUIDs in these sets as *multi-network GUIDs*.

The total number of IPs addresses, private IPs[8], and IPs used by GUIDs that also used known Tor exit nodes that we observed for each of these networks is shown in Table 5.3. Generally, private IPs are the result of sub-optimally or misconfigured end-user applications, as opposed to indicating privacy awareness. In contrast, Tor use indicates privacy-aware users. Table 5.4 shows the size of each pairwise network overlap. For all such intersections, we first remove private IPs and Tor exit nodes (as listed in the Tor consensus files[9]). Of all network pairs, the Gnutella-eMule overlap is the largest.

The union of all three intersections comprises our 84,925 GUID multi-network subgroup for Gnutella. We perform a similar calculation for eMule, resulting in 147,904 GUIDs.

5.5.1.3 Peers that Use Tor

Peers that use Tor are of interest to law enforcement because they are actively masking their identities, thwarting investigations of this crime. Tor does not filter application-level data: GUIDs are passed through to investigators, and thus appear in our dataset as well. We define a GUID as a *Tor GUID* if it was ever observed as having an IP address listed as a Tor exit node in the Tor consensus for the date of the observation. When a Tor GUID's IP is a

[8]Private IP addresses are those which are non-routable on the public Internet, self-assigned, or otherwise invalid, as defined by RFC 5735.

[9]Consensus files contain the list of IPs addresses acting as exit nodes on a daily basis; see `https://metrics.torproject.org/data.html`

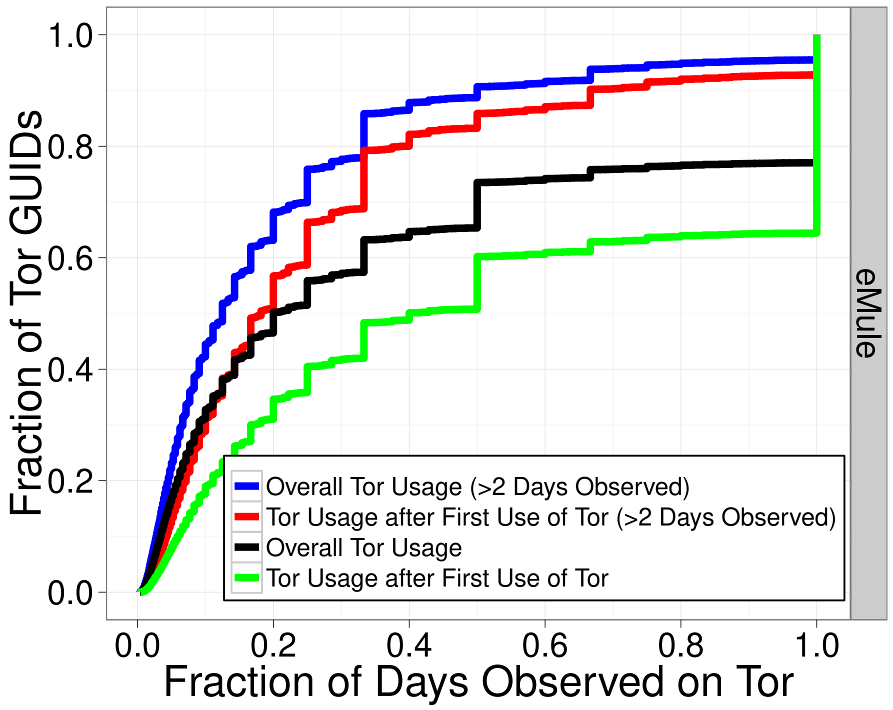


Figure 5.5: CDF of Tor usage per GUID for eMule. GUIDs do not use Tor consistently after first being observed at a Tor IP. Under 40% of Tor GUIDs consistently used Tor after first being observed using it. When considering only Tor GUIDs seen on >2 days (which comprise about 70% of all Tor GUIDs), the rate falls to below 10%. The Gnutella data show similar characteristics.

known Tor exit node we say that the GUID is *using* Tor. As Table 5.2 shows, this set is not large on either network: 3,666 GUIDs for Gnutella and 16,290 GUIDs for eMule.

It is striking that the vast majority of Tor GUIDs do not use Tor consistently, which makes it possible to detect their true IP address. In Figure 5.5, we show the CDFs of *overall Tor usage*. In both networks, only about a quarter of the Tor GUIDs used Tor every time they were observed. More significantly, for these GUIDS, under 40% consistently use Tor after their first use of Tor.

When we examine these 40% of nodes that used Tor consistently, we found that most were observed on the Gnutella and eMule networks for only one or two days. Therefore, we recomputed the distribution of Tor usage for the subset of Tor GUIDs observed three or more days, which is over 70% of all Tor GUIDs. We again also computed the CDFs of Tor usage after first using Tor. The resulting CDFs are the upper lines in Figure 5.5. In sum, over 90% of GUIDs using Tor for more than two days on eMule and Gnutella are easily linked back to a non-Tor IP address, one that is most likely their real location.

This irregular use could be due to ignorance of how Tor works, careless configuration, or frustration with the lower throughput of Tor. It is well known that Tor's *design* does not offer technical protection to p2p users because it does not hide identifying application-level data [99]. In contrast, we provide the first empirical evidence that Tor *users* do not use the software consistently, even those with a strong reason to so. Regardless of the quality of Tor's security, this evidence strongly suggests that its usability (its interface, its effects upon perceived speed, or some other factor) is lacking. We conclude that the use of Tor, as observed in practice, poses only a small hurdle to investigators. Reports by the Tor developers that "Journalists use Tor to communicate more safely with whistleblowers and dissidents"[10] should give one pause, as there is no evidence that those groups are significantly more or less tech-savvy than the users we study.

5.5.1.4 Peers that Use Suspected Relays

The final subgroup we identify is a set of peers that are using IPs that we suspect are relays (other than Tor exit nodes). To create this subgroup, we first collected the set of non-Tor IP addresses used by GUIDs that also used a Tor exit node. We discard the IPs that hosted fewer than four GUIDs (267,035 in the case of Gnutella, and 1,671,419 for eMule), and we nominate the remaining IPs as potential relays. Finally, we create the subgroup of *relayed GUIDs* as the set of GUIDs seen using the potential relays. We cannot validate these GUIDs as having definitely used relays; for example, it may be the potential relays are IP addresses that get reassigned frequently. However, we consider their use of these shared IPs sufficient to define them as a distinct set.

[10]Quoted from `https://www.torproject.org/about/overview.html.en`

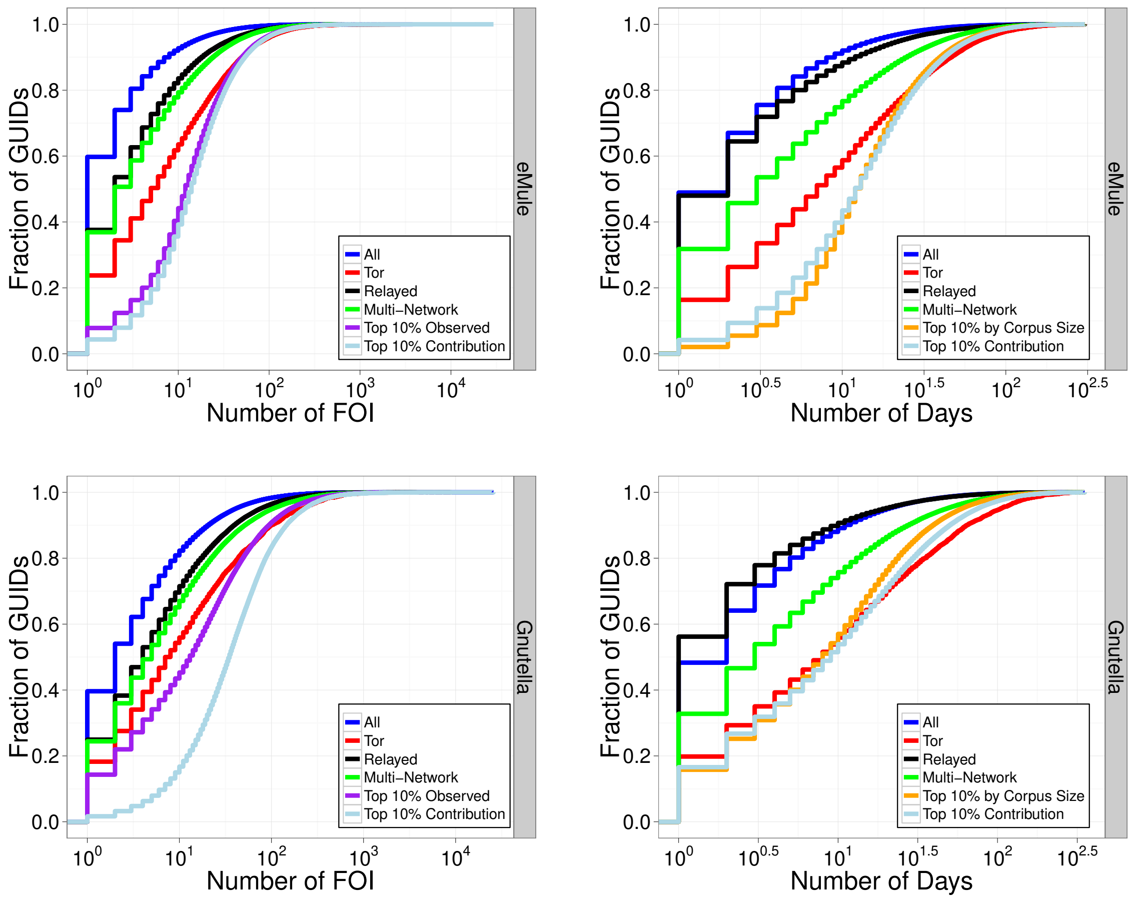


Figure 5.6: Characterizations, as CDFs, of per-GUID corpora and days observed for eMule and Gnutella. The aggressive subgroups, sans relayed, appear to be more active in their trafficking, having more FOI and uptime than the average peer sharing FOI.

5.5.2 A Comparison of Peer Behavior

There are substantive and statistically significant differences among the subgroups in terms of per-GUID corpora and number of days observed. These differences can be seen in Figure 5.6 and are summarized in Table 5.5. In particular, the subgroups generally have a larger corpus size and more days observed online than the set of all GUIDs. The three top-10% subgroups show this effect most strongly, but the Tor subgroup and multi-network subgroups show similar effects. Notably, these latter two subgroups are selected independently of corpus size and days online. This result confirms a hypothesis that tech-savvy groups, whether through Tor or multi-network use, are more active.

The set of GUIDs in the top 10% contribution subgroup represent a combination of the other aggressiveness metrics. This result can be viewed by comparing CDFs in the figure, or by comparing means in the table. For example, the top 10% contribution subgroup's mean corpus size is higher than the top 10% observed, and its mean number of days observed is higher than the top 10% corpus subgroup. The contribution metric could easily be parameterized to weight observations more heavily, though we do not show such results here.

The relayed subgroup in general has larger number of FOI than the all group, and appears online more days on average than the all group in eMule. However, the relayed subgroup shows fewer days observed online than the all group in the Gnutella network. This result suggests that either this subgroup, as we've defined it, is not particularly active, or that our process for identifying non-Tor relays is faulty. It may also be that the peers in the relayed subgroup are more successful at aliasing themselves as different GUIDs that appear on the network fewer number of days each. In the following section, we examine the general problem of user aliasing in this data set.

5.6 Analysis of User Aliasing

The relationship between p2p network GUIDs and real users is not one-to-one in our dataset. In fact, it is possible for a single user to correspond to multiple, distinct GUIDs. We refer to this phenomenon as *user aliasing*, and for some users it is intentional. In this section, we examine observable user aliasing, and we also attempt to quantify its effects upon the analyses in the previous sections. In sum, we find that GUIDs that share at least three FOI any given day generally have distinct libraries. In Gnutella, we can compare all files shared by a GUID, and in that case users sharing a library of at least two files are generally distinct on a given day. We also find little evidence to suggest users are changing their GUIDs and then continuing to share the same library or a portion of it later that day. Parallel results

GUID Group		Mean Value (99% CI)	
		Corpus Size	Days Obs.
	All	10.9 (10.7, 11.1)	5.2 (5.2, 5.2)
Gnutella	Tor	43.9 (39.0, 49.6)	23.4 (21.8, 25.1)
	Relayed	18.9 (18.3, 19.5)	4.8 (4.7, 4.9)
	Multi-Network	25.9 (24.9, 27.0)	10.8 (10.6, 11.0)
	Top 10% Obs.	41.8 (40.7, 43.0)	28.7 (28.5, 29.0)
	Top 10% Corp.	75.9 (74.3, 77.7)	16.2 (16.0, 16.5)
	Top 10% Contr.	69.1 (67.6, 70.9)	19.5 (19.3, 19.8)
	All	4.3 (4.3, 4.4)	4.1 (4.1, 4.1)
eMule	Tor	21.2 (19.9, 22.5)	17.4 (16.9, 18.0)
	Relayed	9.2 (8.9, 9.6)	5.5 (5.4, 5.6)
	Multi-Network	10.8 (10.6, 11.0)	9.5 (9.4, 9.7)
	Top 10% Obs.	23.5 (23.2, 23.8)	22.3 (22.2, 22.4)
	Top 10% Corp.	27.8 (27.4, 28.5)	18.7 (18.6, 18.8)
	Top 10% Contr.	25.8 (25.4, 26.5)	19.0 (18.9, 19.1)

Table 5.5: The expected value and 99% confidence interval of each characteristic for each subgroup of GUIDs. Each subgroup's mean differs from the mean of the "All" group. Each such difference is statistically significant ($p < 0.001$), as determined by a computational permutation test ($R = 10,000$). Confidence intervals are computed by bootstrap ($R = 10,000$).

generally held for eMule, though without the ability to browse eMule user libraries, we are less certain of that result.

The true user aliasing rate in our data is unknowable to us. However, the reasons for deliberate aliasing can be enumerated: *(i)* if a user has two computers (or multiple accounts on a single computer), each with an installation of Gnutella, he will control two unique GUIDs; and *(ii)* a user may reinstall or upgrade their p2p client on a single computer or otherwise modify their GUID over time. We have no way of detecting the first case from only network data; however, the second case can be detected if the user does not alter what files they are sharing, as the file library acts as a kind of signature for the user. It is this latter case that we evaluate in the remainder of this section.

Most users, as identified by GUIDs, are seen with very small libraries of a single file or two. This fact is illustrated in Figure 5.4 in Section 5.4 (and in a week-by-week breakdown in Figure 5.16). We posit that such small libraries are not particularly differentiable. By

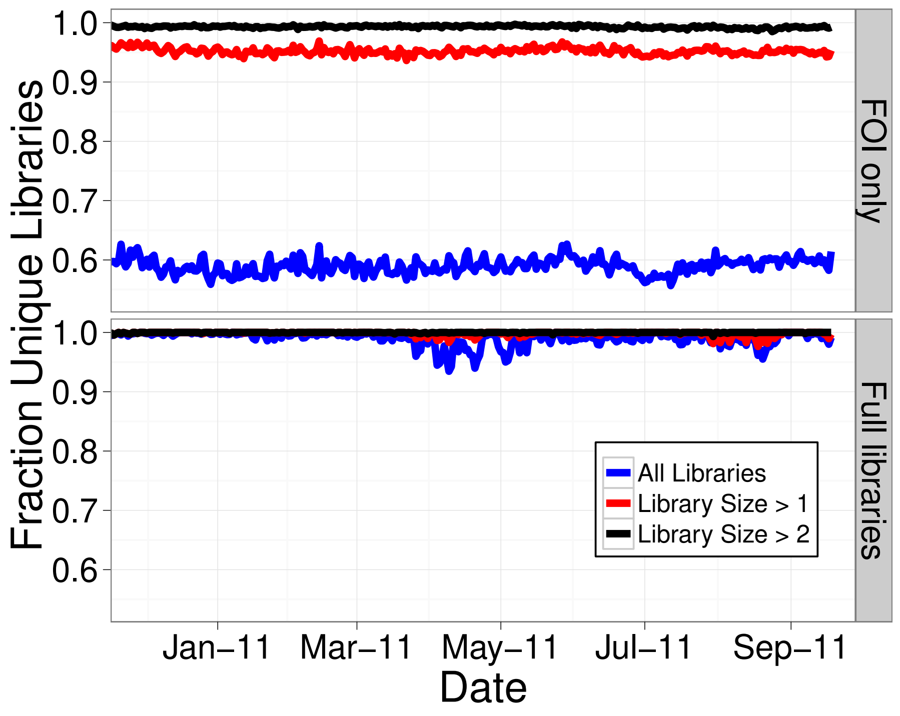


Figure 5.7: Fraction of Gnutella GUIDs with unique libraries on specific days, where uniqueness is defined as libraries that completely match. When considering libraries of at least two FOI, approximately 95% are unique. Similar results hold for eMule. When considering full (browsed) libraries, over 93% are unique.

excluding them, we can determine a lower bound on the user aliasing of type *(ii)* that may be occurring.

We computed day-to-day similarities between Gnutella libraries to determine a lower bound on user aliasing, or alternatively, an upper bound on the number of unique libraries present in the dataset. Generally, we found most libraries to be distinct.

Figure 5.7 shows a comparison of Gnutella GUID libraries, plotting the fraction of GUIDs with libraries that are a unique collection of files. In the upper portion of the figure, a comparison is made of just the files of interest at each GUID; the lower portion compares all files in the library of each GUID (from a Gnutella browse request). GUIDs that have tens of files or more are easy to distinguish from others.

Figure 5.7 shows that in general, GUIDs with a single file are easily aliased with other GUIDs with the same single file: only about 58% of GUIDs have unique libraries on a given

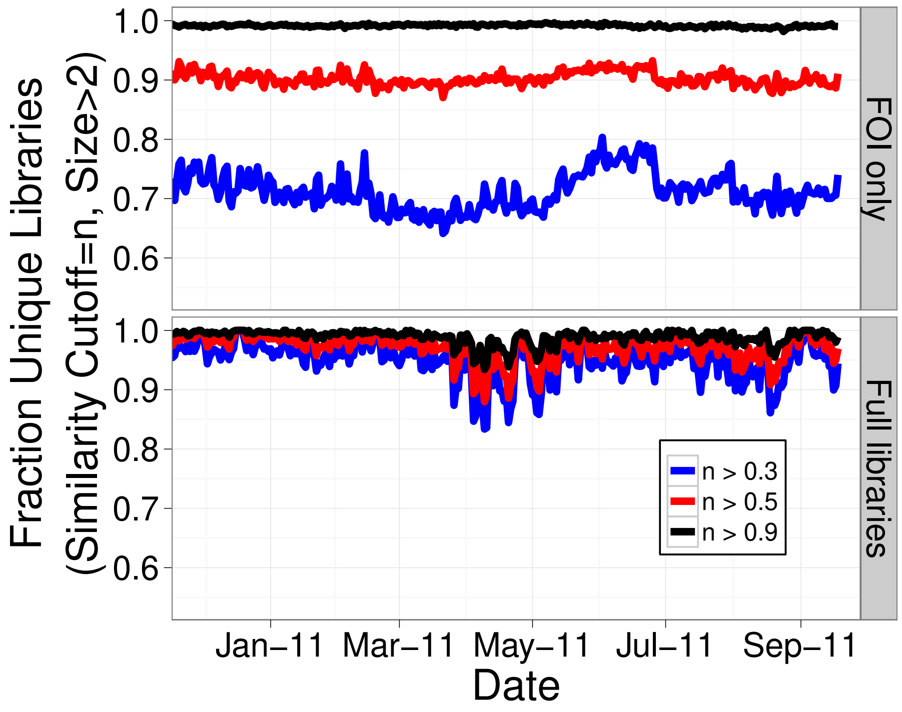


Figure 5.8: Fraction of Gnutella GUIDs with a unique library, where uniqueness is defined as there being no other library with a similarity greater than n. The similarity of two libraries is defined their Jaccard index, $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. On most days, 90% of libraries have no more than half their files in common.

day of our dataset. Among the 40% of Gnutella GUIDs that have two or more FOI, over 95% have unique libraries. Among the 25% of GUIDs with three or more FOI, over 99% have distinct libraries.

Fewer aliases are present when comparisons can be made of the complete libraries, as is possible with Gnutella browse information, by including all files, not just FOI. This is illustrated in the lower portion of Figure 5.7. Note that GUIDs with a single FOI typically possess more than one file, and thus they are more likely to be unique. Typically, GUIDs seen with two or more files in their library had a unique library about 95% of the time; GUIDs with three or more files were unique over 99% of the time.

The above data suggest that we can treat GUIDs as uniquely distinguishable when their libraries contain at least two FOI or when we consider all files that they share. The analysis also suggests that users are rarely if ever changing their GUID and appearing on the same day with the same library. They would appear as aliases if so, and if this was common, the fraction of unique libraries would be lower.

Based on a similar analysis, we also make the claim that there is no compelling evidence that many users are changing GUIDs appearing on the network that day and preserving only *most* of their shared libraries. Figure 5.8 quantifies the uniqueness of partial and complete libraries using the Jaccard index: $J(A,B) = \frac{|A \cap B|}{|A \cup B|}$. In the upper portion of Figure 5.8, we see that for GUIDs with at least three FOI, approximately 90% of libraries have no more than half their files in common on most days of our study. In the lower portion of Figure 5.8, we compare all files in each GUID's library, not just FOI. Here we see on most days, 85% of libraries have no more than 30% of their files in common.

A limitation of our calculations above is that we compare GUIDs only within a day's time. We haven't computed all-pairs, all-times equivalence or similarity among GUID libraries across multiple days because the computation is too lengthy to handle in a reasonable timescale for our dataset.

5.7 Measurement Limitations

The limitations of our study prevent us from providing more than conservative lower bounds on the observable activity of CP perpetrators. First, our set of known FOI is likely biased towards files and filenames shared by traffickers in the U.S. Traffickers in other countries are likely underestimated by our study. Second, all of our records would ideally be associated with a browse, in other words, a complete listing of the peer's current files. eMule does not support browse functionality at all, and investigators do not browse all Gnutella peers on all days. For example, a peer may be identified as having file *A* on day 1 and day 3, but that file is not seen on day 2 because the appropriate keyword or hash search was not run. As a result, we may be underestimating the amount of CP content possessed by each peer as well as the number of days they are online. Third, peers that are online more often are also more likely to be found using a search. We might be underestimating the number of peers that are rarely online and have few files.

On the other hand, one user might have one or more installations of the p2p client software, with each installation showing up as a different GUID. Hence, the number of GUIDs in these networks serves as a rough upper bound on the number of users (for the FOI we knew about).

We also note that before, during, and after the collection of the datasets we analyze, law enforcement were and are active in investigating and arresting CP traffickers. We do not know which peers were removed from the network, and we do not take these removals into account in our analyses. The specific metrics we report on do not rely on linking arrested users to a search warrant and the outcome of a subsequent trial.

5.8 Proof Sketch of NP-Hardness

We define the PEER REMOVAL problem as follows. Given a set of peers and the set of files they shared over D days, remove at most r peers such that the number of files available for at least one day is minimized. In other words, we minimize $\sum_i f_i$, where $f_i = 1$ if a file i is available from at least one peer during at least one day of the D days, and 0 otherwise.

We show that PEER REMOVAL is NP-hard by reducing the MINIMUM k-COVERAGE problem to it. The goal of this NP-hard problem [148] is to select k sets from a collection of n sets such that the cardinality of their union is the minimum.

Given any instance of the MINIMUM k-COVERAGE, we construct an instance of PEER REMOVAL in LSPACE as follows. Let each of the n sets represent a peer with each element of the set representing a file owned by him. Removing $r = n - k$ peers such that availability is minimized would be the same as selecting k peers such that the cardinality of the union of their corpora is minimized.

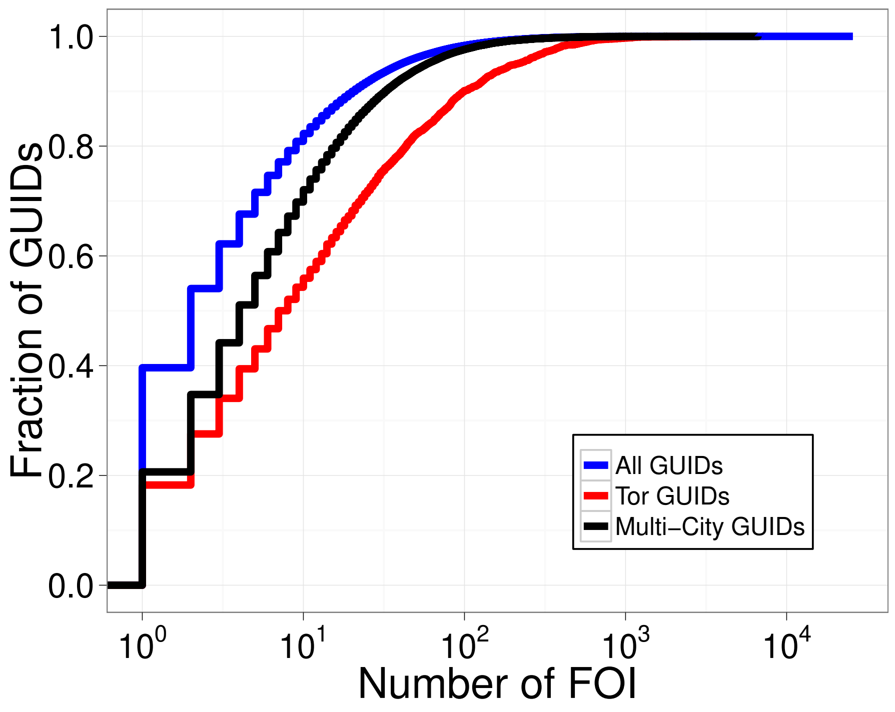


Figure 5.9: CDF of the fraction of GUIDs versus number of files of interest. Multi-city GUIDs do not share significantly more FOI than the set of all GUIDs, and much fewer than Tor GUIDs. “All GUIDs” and “Tor GUIDs” data is repeated from Figure 5.6c.

5.9 Mobility

Each IP address in our dataset is linked to a city-level geographic location using results from a commercial service. We examined whether GUIDs that appeared in multiple cities are characteristically different from other GUIDs within a given network. The results in this section show that, unlike the peer groups analyzed in Section 5.5, multi-city peers do not stand out in terms of days online or library size.

GUIDs can appear in multiple cities for several reasons. First, the user may move to different geographic locations. Second, the user may use a remote host to which he has have access, or he may use a relay. We can’t distinguish these two cases other than for IP addresses that we know or believe to be relays. Therefore, our set of multi-city GUIDs does not include GUIDs that we observed using Tor or a potential relay (see Section 5.5.1.4).

Using this definition, multi-city GUIDs account for 81,496 GUIDs in the Gnutella network. (We elide an analysis of eMule in this section.)

Figure 5.9 shows the number of FOI held by multi-city GUIDs. For comparison, the same values for all GUIDs and for Tor GUIDs in Gnutella are repeated from Figure 5.6c. The data shows that multi-city GUIDs tend to have same fraction of FOI in their corpus when compared to all GUIDs in the network.

We also attempted to find interesting subsets of multi-city GUIDs that were repeatedly in different cities. First, we coarsely computed the geographic diameter of the cities associated with each multi-city GUID. The diameter is the geographic length of the diagonal of a rectangle that covers all cities that a multi-city GUID was seen at. Specifically, we used the haversine formula to compute distance. The red line in Figure 5.10 shows a CDF of the distance covered by multi-city GUIDs. We can filter the entire group by, for each GUID, ignoring IPs in cities that were visited fewer than *n* times. Figure 5.10 illustrates that limiting a GUID to a subset of its cities yields no interesting subgroups.

Finally, we characterized multi-city GUIDs by the time spent in their *home city*, defined the as the city a GUID was observed in most. We found that 42,761 multi-city GUIDs (52%) are seen in their home city more than 50% of the time. Do GUIDs that are more nomadic contribute larger libraries to the network? We have no evidence to support such a claim, as shown in Figure 5.11. In that figure, the multi-city GUIDs are ordered by the fraction of time spent in their home city and then bucketed in 10% increments. Each bucket contains roughly 8,000 unique GUIDs and represents the number of files of interest in each multi-city GUID's corpus. The range of all boxes is between three and ten files, showing that no subset of multi-city GUIDs appear to contribute more significantly to the network.

In sum, most users stay relatively close to their home city, and the particularly multi-city GUIDs are no more aggressive than their single-city counterparts.

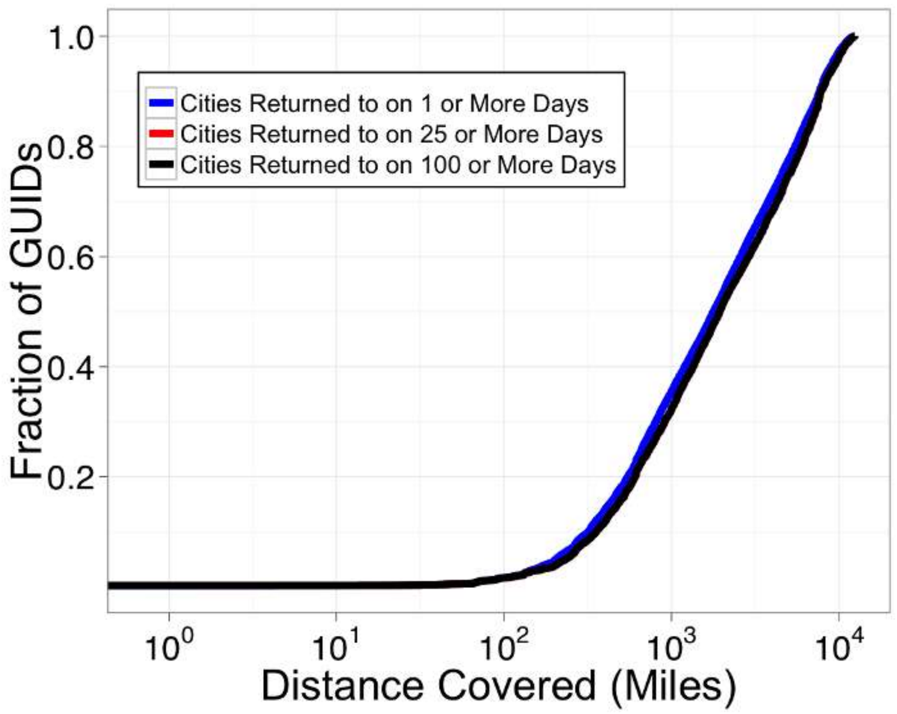


Figure 5.10: CDF of GUIDs versus the distance covered in miles. Each line reduces the valid locations of a GUID by only considering locations where a GUID returned to a city more than n times. Limiting by valid cities produces no differences in behavior.

5.10 Churn

In this subsection, we evaluate the level of user *churn* in our data. Past works on characterization of churn [32,63,127,139] reveal the highly dynamic nature of peer participation in p2p systems. However, there is noticeable difference in specific conclusions they reach, mainly due to the challenging nature of gathering unbiased data about peer participation [139]. To our knowledge, none of the previous studies evaluate data longer than a few weeks' duration. Nor have they evaluated churn of CP traffickers. Consequently, we find our analysis of churn insightful in spite of the limitations discussed in Section 5.3. In sum, we find in our dataset that while there is high churn, there are also many GUIDs which are consistently active in the network. We elide the analysis of churn in eMule.

Figure 5.12 quantifies the uptime and downtime of GUIDs with respect to consecutive days seen in the data. This graph shows that most GUIDs ($> 68\%$), when seen, are not

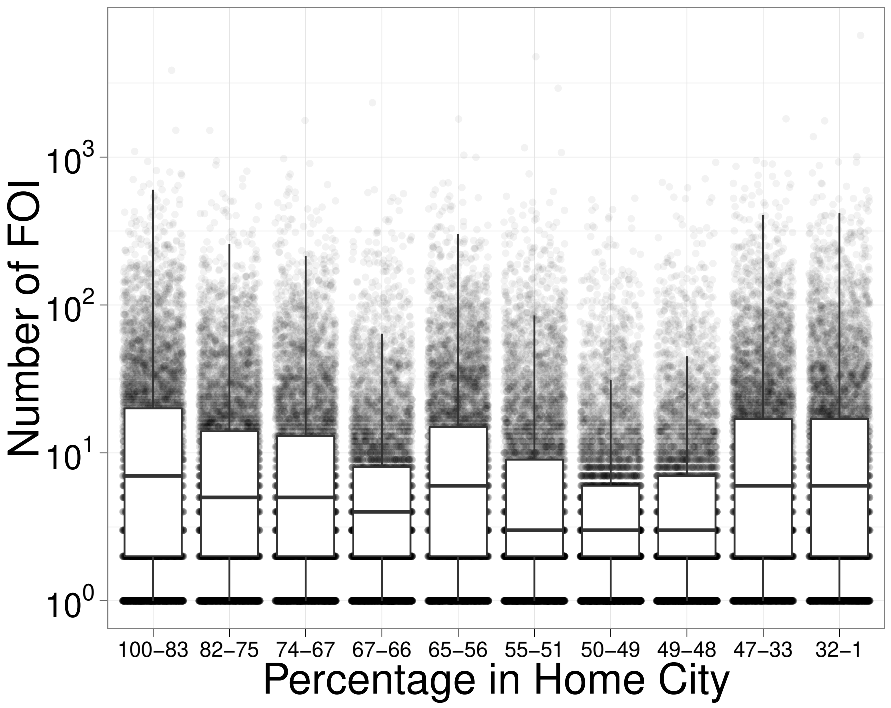


Figure 5.11: A box plot showing the number of files that multi-city GUIDs are ever seen with. The groups are created by ordering the multi-city GUIDs by the fraction of time spent within their home city and then bucketing by 10% increments so that each box contains the same number of GUIDs. Each box represents the semi-interquartile range, with the middle line showing the median; underlying data is displayed as jittered semi-transparent points. No group is strikingly different from any other group.

seen on consecutive days, and that most GUIDs ($> 51\%$) are not observed for five or more consecutive days. This data implies that most GUIDs are only intermittently observed; however, at any given time, a significant fraction of GUIDs' uptimes are longer than a single day. Stutzbach et al. [139] also observe that while a randomly selected active peer is likely to have a long uptime, a randomly selected session is more likely to be short in the Gnutella network. Their analysis, however, is more granular yet covering a much shorter period.

Figure 5.13 shows the correlation of the number of consecutive days that a GUID is observed to the median number of consecutive days remaining for that GUID. The shaded area represents the semi-interquartile range. The results show that the number of consecutive days observed is a good predictor of future uptime. Previous studies [139] also show that

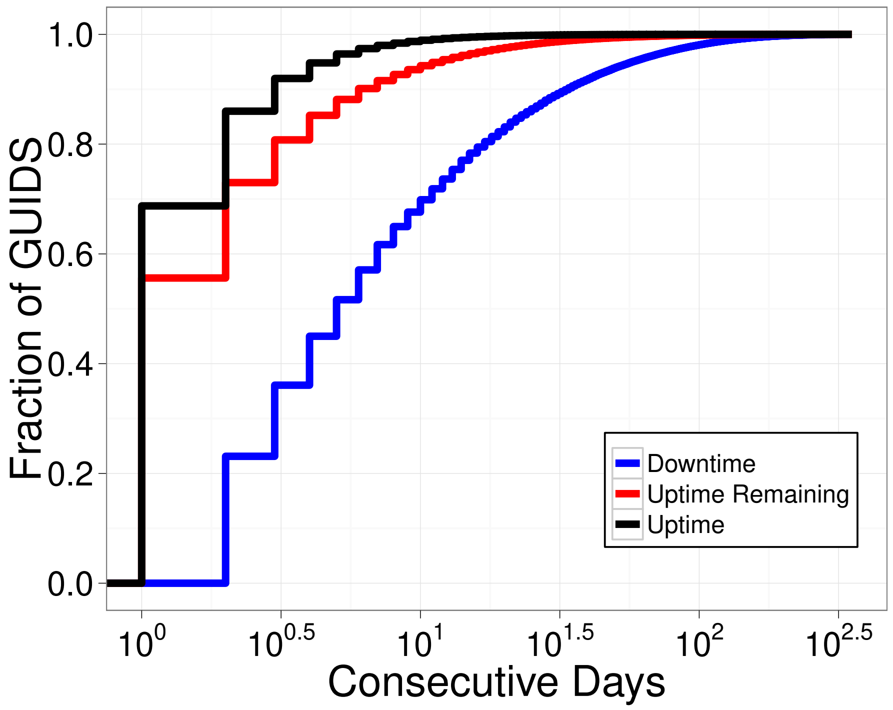


Figure 5.12: CDF of consecutive days of up- and down-time for GUIDs, as well as per-day CDF of days remaining in each GUID's consecutive days seen. Downtimes are only counted when they occurred between the first and last day a GUID was observed. $> 68\%$ of GUIDs are not seen on consecutive days, and more than $> 51\%$ of GUIDs are not observed for 5 or more days at a time.

while exhibiting high variance, uptime is on average a good indicator of the remaining uptime.

Figure 5.14 shows the correlation of session uptime to the median uptime of the next session, where sessions are defined as consecutive days where a GUID was seen. The shaded area represents the semi-interquartile range. The correlation is weak, indicating that session lengths are likely independent. This result is in contrast to what is reported by Stutzbach and Rejaie [139]. We attribute this difference mainly to the different file preferences of the users and longer observation period in our dataset.

5.11 Other Visualizations

In this subsection, we show several characterizations that expand upon figures in the main text.

We evaluate the correlation between the corpus size of a GUID and the number of shared files of interest in Figure 5.15. In summary, the growth in total number of files (FOI and non-FOI) is weakly correlated with growth in the number of FOI; the fact that many peers possess one FOI weakens the correlation accordingly.

Figure 5.16 shows the library sizes of Gnutella GUIDs over time, rather than cumulatively (as in Figure 5.4). Figure 5.16(b) shows complete libraries, rather than only known FOI as in Figure 5.16(a). While we have fewer data points for complete libraries than for only known FOI, the data we do have indicate that complete libraries tend to be much larger on average, though with greater variability.

5.12 Related Work

Ecosystems & Underground Economies. Our work is similar in theme to a body of work exploring economic characteristics of network-based ecosystems [24,52,77,87,105]. For example, the irregular use of Tor by the peers in our dataset might be explained by recent work showing that users abandon privacy for short-term benefits [1].

Content Availability in P2P Systems. A large body of related work on p2p systems investigates availability, performance, and issues related to the use of incentives [27,31, 63,97,102,111,118,161]. Unlike our work, these studies mostly focus on understanding and analyzing the unique properties of p2p networks and their users' behavior, and do not specifically target CP or separate aggressive subgroups.

CP Trafficking in P2P Systems. Prior studies of CP-related trafficking on the Internet have a limited scope. They are mostly indicative of the alarming presence of contraband rather than comprehensively quantifying how the files are being shared [72,84,121,129]. All previous work focused on CP (rather than copyright violations) is based on only CP-related search terms rather than verified content [72,73,84,132,138].

The exception is our own prior work [91], where we analyze CP-related activity on Gnutella during a five-month period with no overlap with the study in this chapter. In that work, we show that the correspondence between IP addresses and application-level identifiers is not one-to-one, and then propose proactive methods of differentiating the end hosts. In contrast, our focus in this work is on reducing availability and characterizing peer behavior.

5.13 Summary and Conclusions

The criminal trafficking of CP on p2p networks is widespread, with no easy answers for law enforcement looking to curtail it. The diversity in peers' location, the redundancy of their libraries, and the many p2p networks, coupled with limited law enforcement resources, dictate triage as a strategy. Specifically, investigators should carefully choose peers to investigate and remove from p2p networks.

We have shown that although naive approaches to triage are ineffective and optimal approaches are NP-Hard, tractable heuristics yield reasonable and useful results. Further, the use of these heuristics are complemented by our discovery of aggressive subgroups of CP traffickers, where such groups correspond to aspects of the heuristics we identified. Prioritizing enforcement in these groups is both effective and easily understandable by LE and policymakers alike.

Further, we have found no significant evidence of users attempting to hide by altering their visible file libraries: peers' libraries are largely unique, strongly implying a unique user behind each such library. Some users do use Tor, but surprisingly, most do so inconsistently, making the investigation of such users straightforward.

It is an open question as to whether network-observable behaviors, such as interest in particular types of imagery, correlate with off-line behaviors of interest to LE, such as child molestation. We leave the exploration of this interesting question to future work.

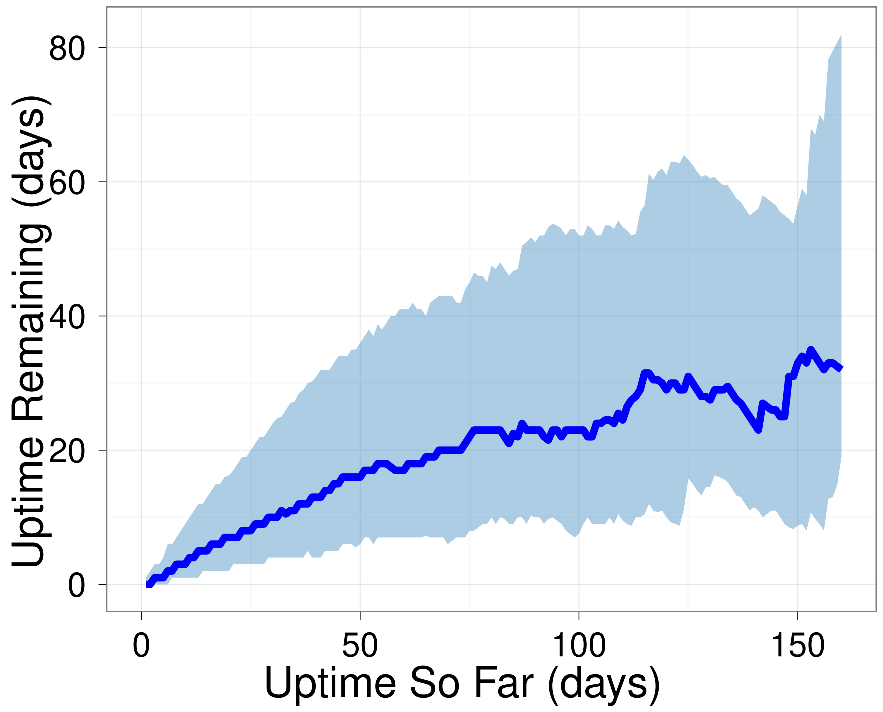


Figure 5.13: Correlation of consecutive days observed at a given point to the median number of consecutive days remaining for each GUID. The shaded area represents the semi-interquartile range. Consecutive days observed are a good predictor of future uptime.

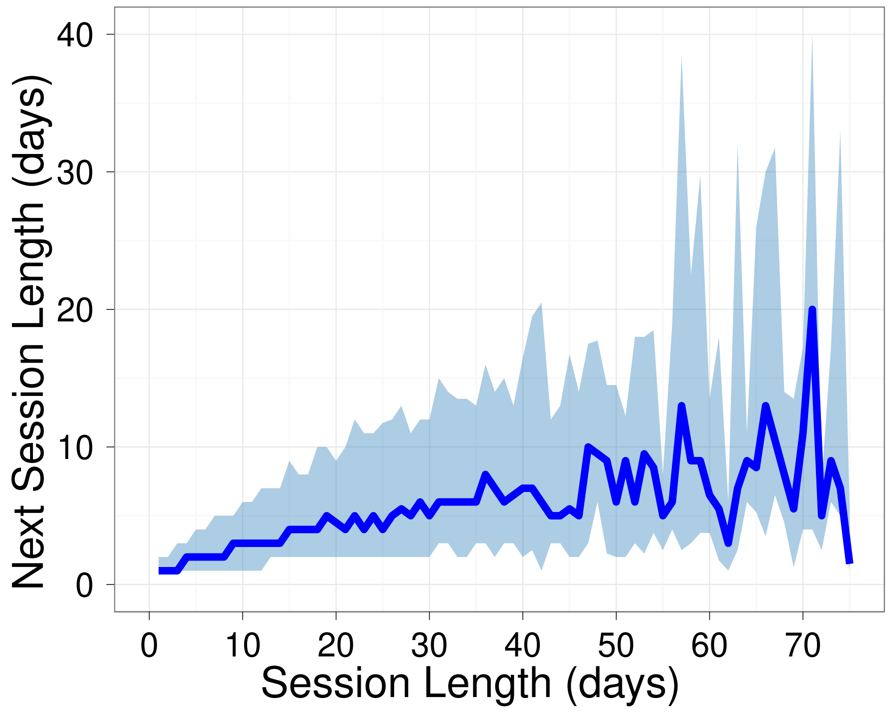


Figure 5.14: Correlation of session uptime to the median uptime of the next session, where sessions are defined as consecutive days where a GUID was seen. The shaded area represents the semi-interquartile range. The correlation is weak, indicating that session lengths are likely independent.

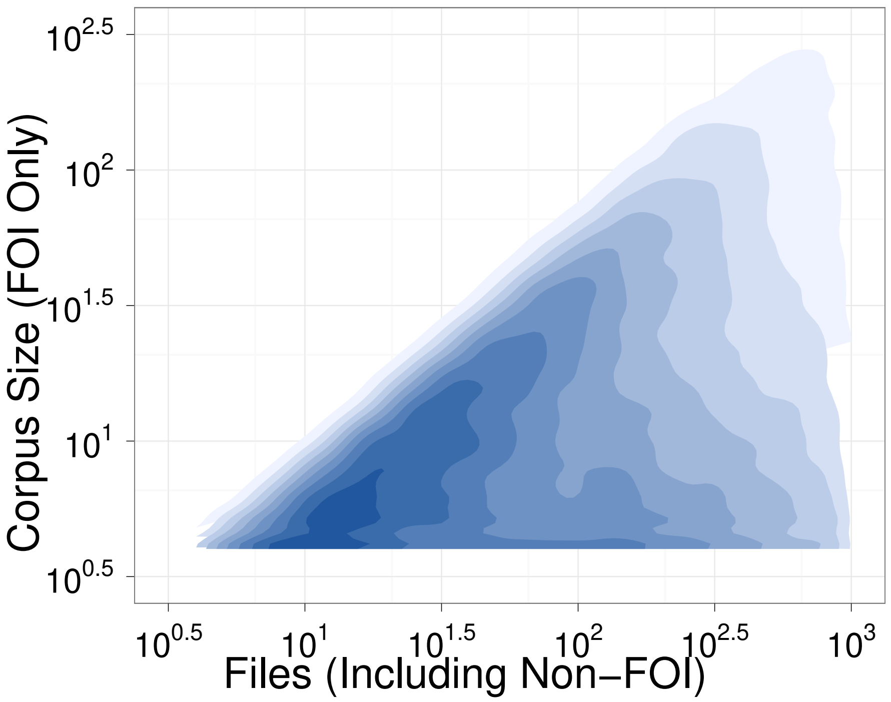
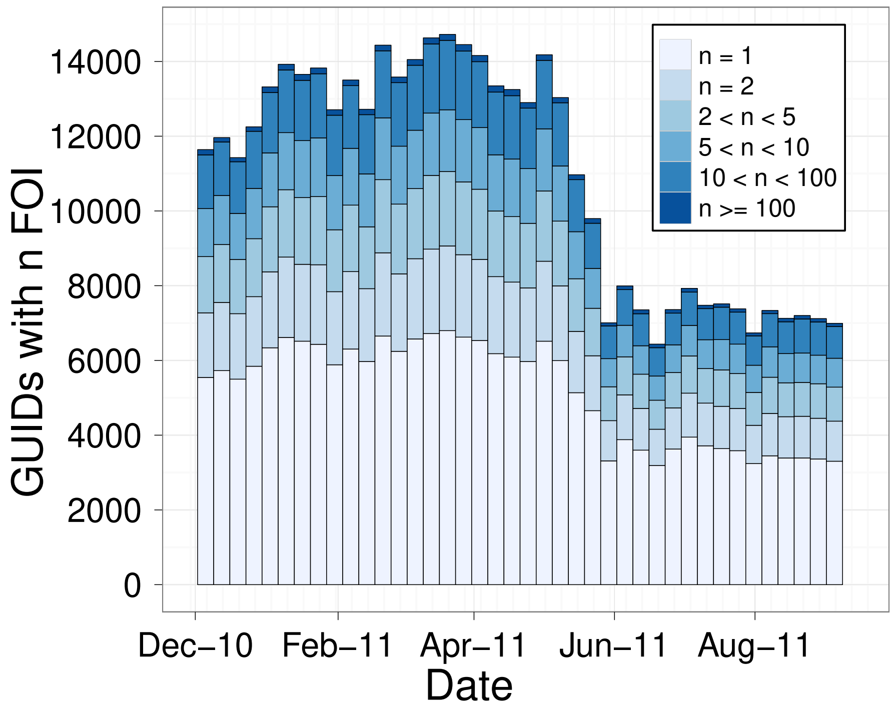
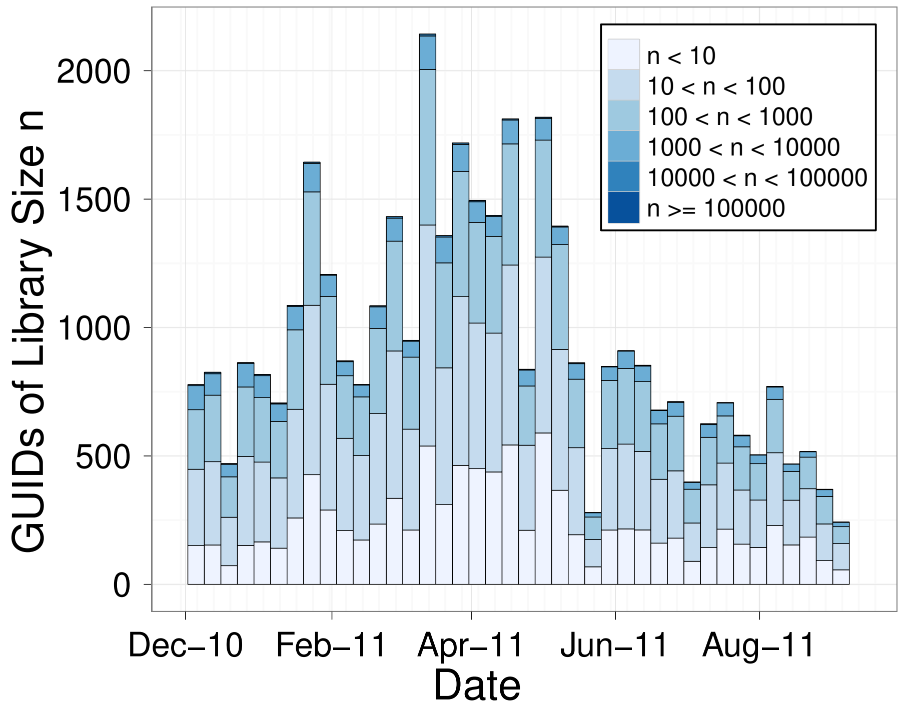


Figure 5.15: Correlation between total number of shared files (FOI and non-FOI) and files of interest only, in Gnutella. Darker areas indicate a greater density of observations. A positive correlation between the quantities is present, but the large numbers of GUIDs with few FOI weaken it.



(a) Counts of GUIDs possessing n files of interest.



(b) Counts of GUIDs with a library size of n.

Figure 5.16: Number of GUIDs with (a) n files of interest, and (b) n total files, in Gnutella. Bar width represents one week of data.

CHAPTER 6
AUTOMATED PROTOCOL EXPLORATION

Developing tools for network-based investigations, such as those described in Chapter 5, require a signification investment of knowledge, resources, and time. The tools have to be tailored to each individual network protocol, and sometimes, to each software implementation. In addition to the initial development cost, the tools may require constant maintenance to support protocol changes or extensions.

In this chapter, we take the first step toward automated development of tools for online investigations. Our work focuses on using inference-based techniques to explore the behavior of protocol implementations and search for bugs and other behaviors that an investigator (or tool) can exploit.

Unlike the techniques we presented in the previous chapter, exploiting protocols may not fit into the standard pre-warrant investigator model. As such, we frame this chapter from both a forensics and a software engineering perspective. Our techniques are applicable to both fields, as they can used by software developers to enhance the security and reliability of their products.

6.1 Introduction

Despite significant effort and resources spent on ensuring software quality, software systems often ship with bugs and security vulnerabilities. These vulnerabilities enable cybercrime, which in 2010 cost \$114 billion globally [141] and has affected one-third of U.S. households [37]. Investigators can leverage these same bugs to enhance their investigations; for instance, by de-anonymizing suspects using anonymity protocols such as Tor. However, finding violations against a software specification is time-consuming and difficult. On average it takes developers 28 days to resolve a security bug [140], and new bugs are reported faster than developers can handle them [3]. Addressing bugs is also expensive: a recent study estimates that $312 billion is spent globally per year on debugging [20].

Testing and discovering differences between a protocol specification and its implementations is especially hard in networked software for several reasons. First, networked communication among many peers introduces an inherent nondeterminism, and error states are difficult to reproduce. Second, it is uncommon for different implementations to correctly or consistently account for behavior that is undefined or unspecified in a specification.

In this chapter, we describe and evaluate APE, a system for automated testing and detection of *specification violations* in networked software. Such violations can result in unexpected behavior, incompatibility, bugs, and vulnerabilities. APE can be used in four ways: (1) to automatically find specification violations in implementations, (2) to automatically verify that a patch fixes a violation and that there are no related versions of the same problem, (3) to automatically apply known exploits to a multitude of implementations of a protocol and tweak the exploits to work on new implementations, which is particularly useful when updates only partially fix a vulnerability, and (4) to automatically generate behaviorally distinct, new tests. As a result, APE helps find specification violations, adapt existing exploits, and test implementations against existing exploits.

APE consists of three stages. In the first stage, APE uses a form of network-based fuzz testing to observe and explore a *target system*'s behavior, and infer a precise state-based model of that behavior. This exploration is iterative: APE infers a model based on some observed executions, identifies unexplored behavior, stimulates the system to execute that behavior, and iteratively refines the model using the new observations. Accordingly, this exploration can either start from scratch, or from partial models of behavior of similar systems, such as other implementations or earlier versions of the target implementation. This allows APE to both discover new exploits and to tweak existing ones to work on a new implementation. In the second stage, APE uses the model and a user-specified description of a specification violation — such as "download a file without contributing any uploaded data" — to propose a set of candidate exploits for the target system. Finally, in the third stage, APE verifies that the candidate exploits, or other exploits encoded directly by the user, are effective on a multitude of implementations. Given the large size of the search space of possible behavior and that not all vulnerabilities can be discovered efficiently, we find that a combination of automated discovery and human-driven exploit selection and specification is most effective.

Our contributions are as follows:

- APE, an approach for automatically discovering specification violations in networked software systems and for reproducibly creating exploitations of these violations (Section 6.3). APE works on compiled binaries and does not require access to the target system's source code. Instead, the user needs to provide executable methods for sending and receiving network messages, and a description that identifies when a specification has been violated.
- An alternate use of APE enables (1) easily applying a known, or an APE-discovered exploit to a multitude of implementations, and (2) tweaking exploits to discover related but distinct exploits (Section 6.3.4). This use of APE allows the developer not only to verify that a vulnerability has been patched, but also that there is no easy way to circumvent the patch to re-expose the vulnerability.
- An automated method of generating tests that are behaviorally different from an existing test suite (Section 6.3.5).
- An analysis of APE's payload modification technique applied to two versions of OpenSSL (Section 6.4). APE discovered and reproduced the Heartbleed vulnerability in one version, and verified that the vulnerability was patched in the other. Further, aided by some manual inspection of the behavioral model, APE discovered two additional specification violations in both versions of OpenSSL.

- An evaluation of the APE prototype that tests three popular implementations of BitTorrent: μTorrent, Transmission, and Azureus (Section 6.5). APE discovered five bugs and unexpected behaviors that can lead to crashes and violations of BitTorrent's tit-for-tat fairness mechanism.
- An analysis of the prototype's efficiency in practice, and an exploration of the convergence properties of the inferred behavioral models and the effects of several exploration strategies (Section 6.6).

Prior work on automated behavior exploration, test generation, and bug discovery, e.g., ProCrawl [134], has enabled automatically exploring behavior in a guided manner. Our work complements and builds on this prior research by targeting networked systems, and allowing modifying messages (as opposed to focusing exclusively on method call sequences), increasing both the search space and types of specification violations APE can discover.

6.2 Motivating Example

APE's goal is to locate and exploit specification violations in networked software. To illustrate how our proposed system works, we first develop a running example using the TRACKER protocol. TRACKER is a simple client-server protocol we designed and implemented to keep track of clients connected to a server.

The TRACKER protocol specification. Clients identify themselves to the TRACKER server using a `hello` message, at which point, the server verifies the client has permission to join the network. Whenever a client requests an update of the current set of clients, the server responds with a list of all clients that have connected since the previous update. Clients request and receive this information using `clients_request` and `clients_response` messages, respectively. Clients leave the network by sending a `goodbye` message. The protocol dictates that the server announces a client as *new* only once per session; a session begins with a `hello` and ends with a `goodbye` message. The server limits the number of sessions

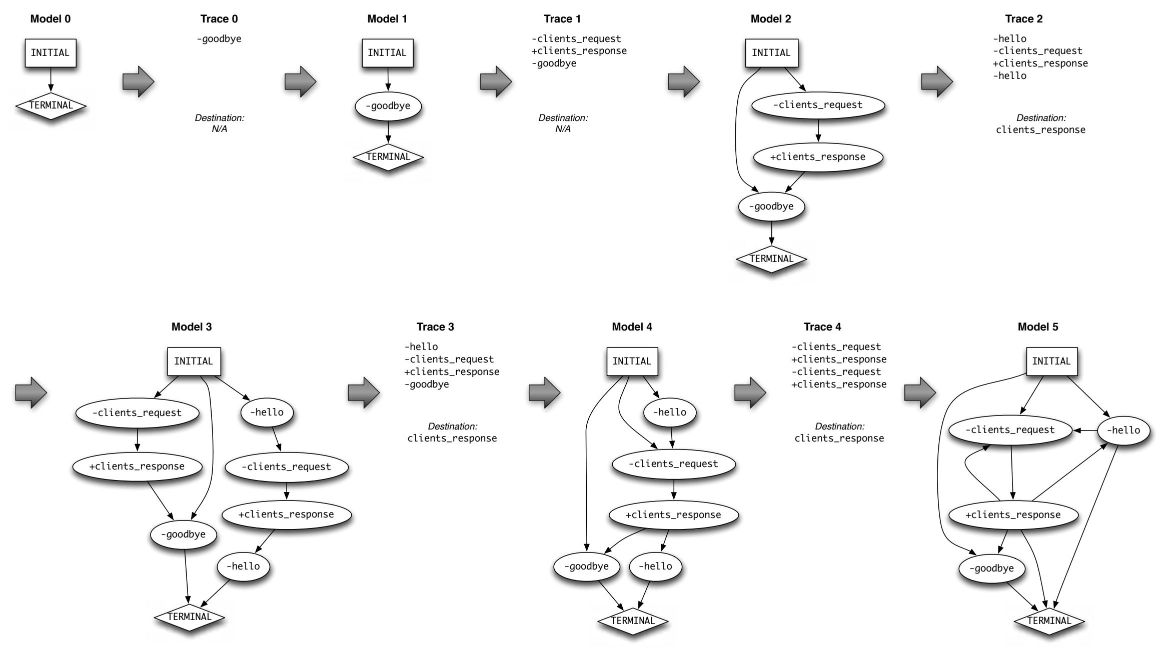


Figure 6.1: APE uses models of its target's (here TRACKER) behavior to guide exploration and generate traces of new executions. For example, using Model 3, APE explores an execution that generates Trace 3, which APE then uses (together with Traces 0, 1, and 2) to build Model 4. Messages *to* the target are prefixed with -, and messages *from* the target are prefixed with +.

per client to one per day. This means that a client should at most be announced in a `clients_response` message once per day.

Example TRACKER implementation. Consider the following code, part of our Python TRACKER server implementation.

```
def handle_message(IP, message):
    if message == ”hello” and \
        self.is_authorized(IP):
          self.connected_peers.append(IP)
          self.new_peers.append(IP)
    elif message == ”clients_request”:
        self.handle_clients_request(IP)
    elif message == ”goodbye”:
        self.connected_peers.remove(IP)
```

The `handle_message(...)` method is responsible for determining the server's response to client's messages. This method contains two bugs that allow a client to violate the intended operation of the protocol. First, lines 6–7 fail to check if the client is authorized before processing the `clients_request` message, allowing a malicious client to receive new peer information without authenticating. Second, a non-new client can trick the server into thinking it is a new peer and cause the server to erroneously broadcast the peer's presence with the next `clients_response` message. This happens because the server fails to check if the client has already connected when it handles the `hello` message in lines 2–5, and appends the IP to the list of new peers in line 5. We focus on the first bug.

Automatic analysis of TRACKER. APE can discover these specification violations automatically with knowledge of only the allowed messages and without access to the source code. Here, we give a high-level overview of how APE does so.

We refer to the implementation in which APE is attempting to discover bugs as the *target*. APE discovers specification violations through automated interaction with the target by observing how the target reacts to different messages and using these observation to learn and explore the target's behavior.

APE has three stages of operation, described here in the context of TRACKER: exploration, candidate violation discovery, and violation verification. The descriptions in this section are at a high, intuition-building level, while Section 6.3 describes the details.

During its self-guided exploration stage, APE tries to learn and refine a model of the behavior of the target. To do so, APE stimulates the target by sending it sequences of messages and observing its behavior. APE systematically chooses which messages to send, and how to alter the messages, by using a model of what it already knows about the allowed behavior, and perturbing previously attempted executions. This results in a form of fuzz testing[1]. Whenever an execution completes, APE restarts the client and begins a new execution with a different sequence of messages. APE logs all messages that are sent and received during the executions, and uses this log to refine the model of known behavior.

[1]Fuzz testing refers to a general class of techniques wherein the tester, or *fuzzer*, employs invalid or random inputs in an attempt to trigger a bug.

APE uses *Synoptic* [11], an existing model-inference tool to infer a finite state machine (FSM) model that describes the target's behavior. While APE could instead use other model-inference algorithms [9,10,38,42,58,94,96,110,126,134,156], our experience showed that Synoptic model's enforcement of temporal invariants observed during execution leads to sufficiently precise models for APE's purposes. Each *path* through the FSM model represents an execution, in terms of the sequence of messages sent and received by APE. An important property of Synoptic-inferred models is that they are predictive: The models include all observed executions *and* predicted executions, as-of-yet unobserved, but likely to be possible because they satisfy key temporal properties mined from the observed executions. This allows APE to better guide exploration along previously unobserved but likely meaningful paths. Additionally, APE probabilistically mutates these paths to create more diversity in the exploration. APE refines its model of the target's behavior by iteratively executing potentially mutated paths, collecting execution traces, and inferring models based on the observed executions. This model-based exploration is better targeted and more efficient than the alternative fuzz testing approach of sending random sequences of network messages and observing the target's behavior.

APE can both discover specification violations and generate variations of known exploits to work on new implementations. To find new exploits, APE starts its guided exploration with an empty FSM model. To tweak existing exploits, APE starts with an FSM model of behavior that includes those exploits, and guides the exploration along those exploits. In all cases, the user must provide a set of *vulnerability characteristics* that the implementation is expected to follow.

Figure 6.1 illustrates five iterations of the exploration process for discovering exploits from scratch. Model 0 shows the starting FSM model, which encodes no behavior, just two connected states labeled `INITIAL` and `TERMINAL`. The first round of exploration generates a single trace (Trace 0): sending a `goodbye` message (in the traces, sending is denoted by a `-`, and receiving by a `+`) generates no response. APE improves its understanding of TRACKER by using Synoptic to infer Model 1 from Trace 0. Next, using Model 1, APE picks a path (sending a `goodbye`) and permutes it slightly to generate a new Trace 1: Sending a `clients_request` ahead of the `goodbye` message results in receiving a `clients_response` message. Inferring an FSM model from the two observed traces together results in Model 2. Repeating this process further refines the model of TRACKER's behavior. Note that Model 5 includes not only observed behavior, but also predicted behavior, such as the trace: ⟨`-client_request`, `+client_response`, `-hello`, `-client_request`, `+client_response`, `-goodbye`⟩. These predicted paths allow APE to explore the target's behavior more efficiently than by guessing randomly or through exhaustive testing.

As APE explores and models its interactions with the target, it employs its second stage, candidate violation discovery, to analyze the model for potential specification violations. In the description of TRACKER, one vulnerability characteristic provided by the user is that `clients_response` should always be preceded by `hello`. A user may be interested in finding paths that do not satisfy this property, evincing a violation. Model 5 in Figure 6.1 includes one such path: ⟨`INITIAL`, `-clients_request`, `+clients_response`⟩. APE searches through the graph to find all such loop-free paths.

Whenever APE finds candidate violation paths, its third stage attempts to execute the candidates against the target implementation. As some paths are predicted, rather than observed, this verification step ensures the candidates are real. APE can report verified violation exploits to the user, and generate an implementation of the tester that follows that path of execution to exploit the violation.

If APE is given a test suite from which to generate the initial traces and model, it will explore the target's behavior to first generate and then verify new executions untested by the test suite.

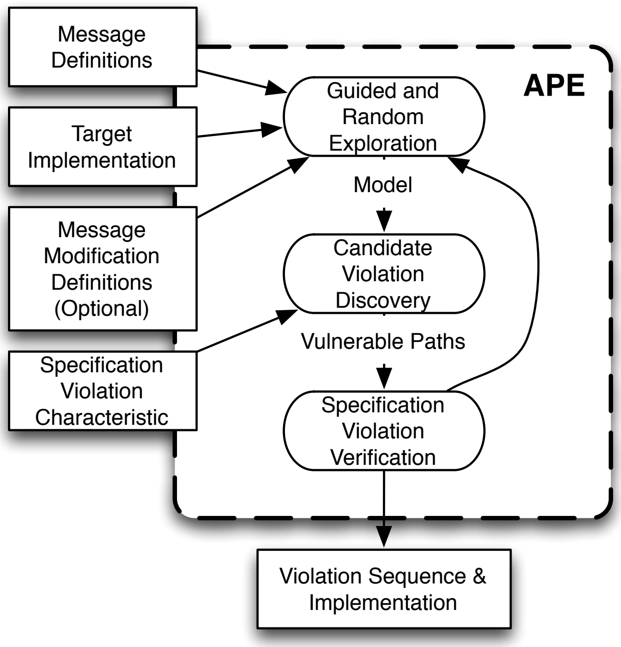

Message Definitions
Target Implementation
Message Modification Definitions (Optional)
Specification Violation Characteristic
APE
Guided and Random Exploration
Model
Candidate Violation Discovery
Vulnerable Paths
Specification Violation Verification
Violation Sequence & Implementation

Figure 6.2: A high-level overview of APE and its three stages: exploration, analysis, and verification.

6.3 APE System Design

This section provides the details of APE's operation using the TRACKER example from Section 6.2.

As illustrated in Figure 6.2, APE's goal is to find specification violations in a software system that implements a given protocol. APE has three required and one optional inputs: (1) a binary of the target software system implementation; (2) a description of the messages the system may send as part of its protocol; (3) a set of descriptions for identifying candidate violations; and (4) optionally, a set of descriptions for modifying legal messages. APE does not require access to the target system source code. APE operates in three phases. First, APE *explores* the behavior of the target system. Second, APE *analyzes* the model to find likely system executions that fit the violation descriptions. Third, APE follows the model to attempt to execute each of the violations, *verifying* that they are real.

We begin by precisely specifying the inputs to APE (Section 6.3.1). We then describe each of APE's three phases of operation in detail (Sections 6.3.2–6.3.4), demonstrating each step on the TRACKER protocol. Finally, we describe how APE can generate tests (Section 6.3.5).

6.3.1 APE Inputs

APE requires three inputs: (1) a binary of the target system; (2) a description of the legal messages; and (3) a characteristic of the specification violation. Optionally, APE accepts a fourth input: (4) a description of the ways in which the legal messages can be modified.

Target implementation binary. APE requires access to a binary of the target system and the ability to start and stop its execution. For TRACKER, this means the command to start and to kill the TRACKER server process.

Message definitions. APE also requires a description of the messages that can be sent to and received from the target. The description of the messages must be sufficient for APE to recognize incoming messages from the target, and to send outgoing messages to the target. In our implementation, the description consists of several executable methods: for each message that can be sent, a method that sends the message; for each message that can be received, a method that parses the received messages and returns the event type; and optionally, one method for each receivable message that will be called when that message type is received.

For TRACKER, the message description consists of five methods shown in Figure 6.3: three for sending the different message types; one for handling the received `clients_response` event; and one for translating the received payloads into event types. For TRACKER, this last method is trivial because we can only receive a single type of message, `clients_response`. To test a more complicated system, such as BitTorrent (described in Section 6.5), a more complex set of methods may be required. Our BitTorrent executable message description is structured similarly to the description for TRACKER, but includes the logic and state necessary to minimally participate in the BitTorrent protocol.

```
class TesterProtocol(ExploreProtocol):
  def send_hello(self):
    self.transport.write(
      struct.pack('!I', 5) + 'hello')

  def send_clients_request(self):
    self.transport.write(
      struct.pack('!I', 15) +
        'clients_request')

  def send_goodbye(self):
    self.transport.write(
      struct.pack('!I', 7) + 'goodbye')

  def handle_clients_response(self, data):
    pass

  def get_message_type(self, message):
    #Use to determine message type
```

Figure 6.3: A TRACKER message types description written in Python. It has three methods for sendable messages, one method for a receivable message, and one method to parse received messages into their event types. The `struct.pack` is a standard python function, and `transport.write` is provided by the Twisted [144] networking framework.

Specification violation characteristic. As a final required input, APE needs a way to recognize when executions exploit a specification violation. APE uses the violations description to identify which paths in the model have the potential to exploit the violation. We present violation descriptions more fully when discussing candidate violation discovery in Section 6.3.3.

Message modifications. Optionally, APE accepts a description of how messages may be modified. Without such a description, APE can discover specification violations by altering the *order* of legal network messages. With this optional input, APE can also discover violations by altering the *content* (often called *payload*) of the messages. In our implementation, this input consists of several executable methods: for each sent message that can be modified that describes how to generate such messages.

We wrote methods that generate varied message content by specifying the fields of the message, which fields can be modified, and how the fields can be modified. For example, Section 6.4 describes our OpenSSL `heartbeat` [66] message generator. The `payload` field is a random-sized series of random bits. And the `payload_length` field is chosen uniformly at randomly to be either the size of the payload field, a random integer larger than the size of the payload field, or a random integer smaller than the size of the payload field. Specifying a finite number of values from which each field can be selected randomly is another approach that could be implemented.

6.3.2 Exploration

The goal of the exploration stage is to interact with the target implementation and learn how it responds to sequences of messages. The product of this stage is an FSM model of the implementation's behavior. The FSM has a state for each message that can be sent or received, and two special `INITIAL` and `TERMINAL` states; the paths from `INITIAL` to `TERMINAL` represent potential executions.

The exploration stage executes the target implementation under varying conditions, and collects traces of the network messages sent and received during execution. APE then uses a model-inference technique to automatically infer an FSM from those execution traces. The FSM model is a generalization of the observed system behavior; it includes paths that represent all the observed behavior, but it also includes more paths that the model-inference algorithm deemed similar enough to be likely possible (although unobserved) executions.

One way to generate traces is to randomly stimulate target implementation behavior by sending random sequences of messages, and observing how the implementation responds. However, this unguided search through the space of behavior is inefficient. For example, if a system requires a handshake before any interesting interaction can begin, most randomly chosen sequences will not contribute meaningfully to behavior exploration as they will not begin with the handshake.

To mitigate these problems, APE uses guided exploration (explained in Section 6.3.2.3). It starts with known behavior and perturbs it slightly to increase the chances that attempted executions are successful. The goal is to add meaningful information to its knowledge of the target's behavior with each generated trace. However, when APE starts with no knowledge about the behavior, it is forced to use random exploration (explained next) until it learns a little behavior and can switch to guided exploration.

6.3.2.1 Random exploration

The base case for APE is to have no knowledge of a target implementation's behavior. This case is represented by a model with only two states, `INITIAL` and `TERMINAL`, as shown in Model 0 of Figure 6.1. In this case, there is no known behavior to perturb, so APE commences random exploration.

During random exploration, APE does not attempt to follow any particular path or reach any destination, but instead sends messages that have not been previously sent from the current model state. To determine its current location in the model, APE examines the last k received messages. APE finds all instances of this sequence of messages as a path in the model; the set of states each of which is a last state in at least one of these paths (labeled with the most recently received event) is the set of possible current states. It then chooses a message uniformly at random from the set of messages not yet sent from the set of current possible states. If that set of messages is empty, APE selects a message uniformly at random from among all messages. If APE fails to find any such paths, and thus any candidate states, it repeats the process using a value of $k-1$. If there are no valid candidate states for $k = 1$ then APE selects a message to send uniformly at random from among all messages. In practice, we found that $k = 5$ provided an adequate balance between performance and accuracy.

Example. As explained above, the two-state initial model for TRACKER does not have any received message events, so APE starts Trace 1 by sending a single random message, in this example, a `goodbye` message. Not surprisingly, the TRACKER server does not reply to the `goodbye` message and the trace terminates after a timeout (Trace 0 in Figure 6.1). APE then uses Synoptic (see Section 6.3.2.2) to infer Model 1 from Trace 0. Model 1 contains just a single path from `INITIAL` to `TERMINAL` via `-goodbye`.

Model 1 still does not have any valid destination events, so it once again APE enters random exploration mode. This time, it sends a `clients_request` message, waits (using a timeout) for a possible response, receives a `clients_response` message, and then sends a `goodbye` message (Trace 1). APE now uses Synoptic to infer Model 2 from Traces 0 and 1. Now that the model has a valid destination event, receiving a `clients_response`, it will use guided exploration. We next describe model inference, and then guided exploration.

6.3.2.2 FSM Model Inference

APE uses the observed execution traces to infer a predictive FSM model. There are many existing techniques for such model inference [9–11,38,42,58,94,96,110,126,134,156], and it is not the focus of this work to improve on them. Instead, APE uses *Synoptic* because of its precise predictive properties and previous use for *manual* software debugging [11]. Other model-inference techniques would likely also support APE well, perhaps presenting certain trade-offs in exploration efficiency and effectiveness.

Synoptic infers a model by mining a set of temporal properties present in the observed execution, such as `hello` always eventually followed by `goodbye`. Synoptic then builds a concise model of the observed traces, and uses model checking and counterexample-guided abstraction refinement (CEGAR) [34] to eliminate predicted paths that do not satisfy the mined temporal properties. The end result is a precise and concise model that includes all observed executions and predicts unobserved executions deemed likely because they satisfy the mined temporal properties.

Representing unmodified messages in models is straightforward: each message is represented by its name. However, incorporating messages modified via the user-specified message modification methods is more complex as the modifications of the same message type can either be treated as identical or distinct. APE allows for two ways to incorporate messages with modified payloads into the model: (1) APE can create a unique name for each modified message instance (e.g., modified `hello` messages may be named `hello_1`, `hello_5`, and `hello_16`). This approach's main drawback is that Synoptic's predictive power is reduced because it sees similar (but slightly different) messages as completely different, which can affect the efficiency of APE's exploration. However, the primary benefit of this approach is that once APE finds a candidate violation in a model (see Section 6.3.3), the model contains sufficient information to verify it (see Section 6.3.4). (2) APE can abstract all modified messages with a single message name (e.g., `hello_modified`). This approach enables Synoptic's predictive power, but requires, once a candidate violation path is identified, retracing the execution traces to identify which executions led to this path, and how to recreate the executions' modified messages. While APE supports both approaches, the evaluation in Section 6.4 will follow the former.

Throughout exploration, APE periodically updates its FSM model using all of the traces collected up to that point. The frequency of the model updating is configurable; for exposition, for the TRACKER example, APE updates the model after every execution trace.

6.3.2.3 Guided exploration

APE's guided exploration starts with an FSM model of its interactions with the target's behavior. Guided exploration explores along the paths already in the model, but introduces deviations from those paths to discover new behavior. Because Synoptic's models are predictive — they include likely possible but unobserved behavior — both paths in the model and deviations from those paths can produce evidence of new, unobserved behavior.

Choosing a destination. To guide its exploration, APE first selects a *destination event*, uniformly at random, from among all message types that appear as *received* messages in the model. In other words, APE's goal is to coerce the target into responding with a specific message. APE does not choose its destination from among *send* message types because sending a message to the target implementation can be achieved simply by sending it, without exercising much of the target's behavior.

For TRACKER, `clients_response` is the only receivable message, and thus it is the only possible destination. However, more complex target implementations are able to send more message types, and present more exploration destination possibilities.

To avoid falling into local minima, with a low probability, APE forgoes guided exploration and enters random exploration mode. This probability is inversely proportional to the number of distinct event labels in the current model.

Choosing a path. After choosing a destination, APE randomly picks one of the paths from `INITIAL` to one of the states that represent the selected received message type. For example, in Model 3 in Figure 6.1, there are two possible paths to states labeled with `+clients_response`: (1) `-clients_request` and (2) `-hello`, `-clients_request`. APE's guided exploration's goal is to guide the target to the destination state by inducing an execution along the chosen path (possibly with some deviations).

Sending messages. An execution consists of a sequence of a sets of sent events and received events. For example, in Trace 2, APE sends `hello` and `clients_request` messages, receives a `clients_response`, and sends a `hello`. Using a path, APE identifies the first set of messages it needs to send to follow that path. For each message in that set, APE may perturb the message with a small probability. APE may randomly skip the message entirely. If the message is not skipped, then APE may substitute a different message, chosen uniformly at randomly from the set of all possible send messages. In practice, for the systems and models we have used, we find that a probability of $^1/_{11}$ works well for both skipping and substitution. Once APE chooses and sends the message, it waits up to 0.5 seconds for the target's response. If there is no unexpected response, APE moves on to the next set of send messages in the path, following the same perturbation procedure until it has traversed the entire path and received the destination message. APE then switches to random exploration mode to finish the current execution and generate a new trace.

Determining a location in the model. APE may not always receive the expected set of messages on the path it is following. This outcome can happen because APE perturbed the path, or because the path was predicted inaccurately by Synoptic. Whenever APE receives an unexpected message, it attempts to reconcile that message with the current model. To do this, APE tries to determine its current state in the model using the last *k* received messages, just as it did in its random exploration mode (recall Section 6.3.2.1). If that process cannot find a suitable state, APE switches into random exploration mode to finish the current execution and generate a new trace.

Example. Since Model 3 contains received events, APE picks `clients_response` as the destination event for the new execution (Trace 3). It selects the `-hello`, `-clients_request`, `+clients_response` path and sends the first two messages, by chance, without perturbing them. APE receives the expected `+clients_response` and APE switches to random exploration mode, choosing to send a `-hello` message. The target then closes the connection. APE then uses Synoptic to infer Model 4 from Traces 0–3.

6.3.2.4 Exploration with message modifications

If APE is given the optional description of how message content may be modified, APE uses a two-phase approach to exploration. First, it uses standard guided exploration (Section 6.3.2.3), without modifying any messages, to build an initial model of the target system's behavior. Next, it switches to the message-modifying phase: every time it sends a message of a particular type, it uses the message modification method to generate a new message of that type and send it. (Note that the message modification method may choose, at times, to generate an unmodified message.)

6.3.3 Candidate Violation Discovery

After a user-specified number of trace generation and model refinement iterations, APE takes the current model and searches for specification violations. APE uses the violation characteristic (recall Section 6.3.1) to identify candidate violations.

Specification violation characteristic. A violation characteristic is a function whose input is an FSM model and output is a set of paths through the model. The intent is for the paths expose a violation. For example, for TRACKER, this characteristic could be a function that returns all paths that allow a peer to receive new peer information without authenticating, which exposes the first bug in our TRACKER implementation (recall Section 6.2). However, to simplify the use of APE, we allow for a highly simplified definition of the violation characteristic: The user needs only to describe two conditions, (1) an ordered sequence of states that must present in the path, not necessarily contiguously, and (2) a set of states that must not be on the path. For example, for TRACKER, the characteristic specifies that the violation-exposing path contains `INITIAL` and `clients_response`, but does not contain `hello`.

Enumerating candidate paths. Given a model and a set of candidate violation characteristics, APE programmatically finds candidate paths using a straightforward graph search algorithm. The current APE implementation removes the set of states that must never occur on a candidate path from the model, along with all edges to or from those states. Then, APE uses depth-first search to find all (loop-free) paths between sequential pairs of states that must occur in the violation-exposing path. Finally, APE combines the paths between the sequential pairs of states to produce complete candidate violation-exposing paths.

For example, for TRACKER, APE uses Model 5 from Figure 6.1 to first, remove the `-hello` state and adjacent edges from the model, and second, find the only loop-free path that contains `INITIAL` and a `-clients_response`.

6.3.4 Specification Violation Verification

APE uses the target system binary to verify the candidate violation-exposing paths. It follows the path to interact with the binary, much in the same way as it does during exploration (recall Section 6.3.2), except without perturbing the path.

For example, in Section 6.3.3, APE identified `INITIAL`, `-clients_request`, `+clients_response` as a candidate violation-exposing path, so it will start TRACKER, send a `clients_request` message, and then wait to receive the `clients_response` message. In this example, the buggy TRACKER server will respond with `clients_response`, verifying the violation. In this case, APE reports the success as a discovered violation.

Because Synoptic can make mistakes, not all candidate paths can be verified. APE tries multiple times to verify the path, in case of nondeterminism. If after verification, APE fails to produce at least one verified violation-exposing path, it returns to the exploration stage, stimulating and learning about more of the target system's behavior before attempting again to find violations. Since the executions observed during the verification stage are particularly relevant to the violations, APE includes them in the traces uses to infer the behavioral models.

Note that while APE uses violation verification to test automatically discovered candidate violations, it can just as well test other candidates. For example, it could test a violation that was present in an earlier version of an implementation, or in other implementations of the same protocol. By using exploration, APE can start with a specification violation from another version or implementation, and explore if the target implementation is vulnerable to variants of those violations.

Limitations. Synoptic does not support negative examples when building the model, although some recent model-inference research suggests how it could [9]. With such support, APE could explicitly eliminate candidate violations verified as impossible from the models to further improve the model's precision.

As APE executes, it may find violations. However, it is not guaranteed to do so; for example, some target system implementations may not have specification violations of the kind described by the violation characteristic. In this case, the more APE explores without finding a violation, the higher the user's confidence that the system does not have violations of this particular type.

6.3.5 Test Generation

While generating candidate violations, APE explores the target implementation's behavior and discovers new, previously unobserved executions. This can be used to automatically generate white-box tests — tests based on the implementation, as opposed to a specification or requirements — in two different ways.

Greenfield test generation. Using APE as in Section 6.3.2, to explore the behavior of the target system and infer a behavioral model of its legal executions, can lead to test generation. All paths in a model are either observed or likely predicted executions of the system. Further, every distinct path represents somewhat distinct behavior: the more two paths are different, the more different are the two executions those paths represent. Generating paths on the model from `INITIAL` to `TERMINAL`, (e.g., via a random walk starting from `INITIAL`) generates potential tests that can then be verified similarly to the way APE verifies violations in Section 6.3.4.

Augmenting an existing test suite. If a system already has a test suite, APE can execute those tests, record the traces of those executions, use Synoptic to generate a model of those traces, and use that model as a starting point for exploration (Section 6.3.2). Exploration will produce new executions and APE can check if those executions are accepted by the model built from the existing test executions, or if they represent new, previously untested behavior. That is, whenever a newly discovered execution is not in the language of the original model (because it requires additional transitions or states), this execution represents behavior, or combination of behavior that was not covered by the test suite, and is thus worth considering adding to the test suite. Again, newly discovered executions can be verified similarly to the way APE verifies violations in Section 6.3.4.

6.4 Using APE to Analyze OpenSSL

In this section, we detail the application of APE to OpenSSL's implementation of the TLS Heartbeat Extension Protocol. We configured APE to generate OpenSSL messages with

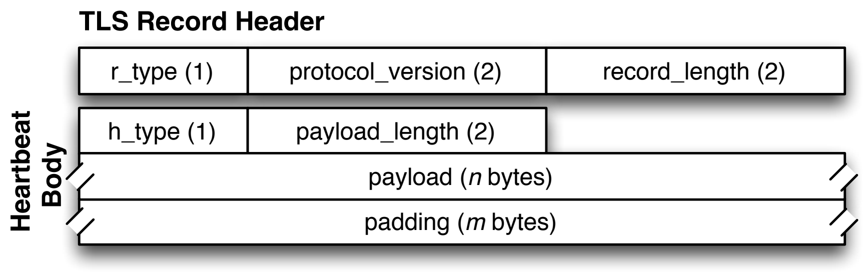


Figure 6.4: The layout of a TLS heartbeat message.

varying content, and we found that it was effective in detecting a known buffer-overflow vulnerability. In doing so, APE also identified two deviations from the protocol specification. We have reported all these bugs to the respective projects' developers.

6.4.1 The Heartbleed Bug

Researchers recently disclosed a bug in OpenSSL's implementation of the heartbeat extension protocol for TLS that allows an attacker unauthorized access to private keys [66]. The heartbeat extension protocol[2] provides a mechanism for clients to maintain a connection to the server by sending a heartbeat message, requesting a response that echoes the heartbeat. The so-called *heartbleed* bug allows an attacker to specify a small payload but request a larger payload to be echoed back. The vulnerable OpenSSL servers, without checking the bound condition, then respond with internal, private memory state.

Applied to two versions of OpenSSL, APE was reliably and reproducibly able to discover the heartbleed vulnerability in one version, and verify the vulnerability was patched in the other version. Further, aided by some manual inspection of the behavioral model, APE discovered two additional specification violations in both versions of OpenSSL.

[2]The heartbeat extension protocol is defined in RFC 6520, `https://tools.ietf.org/html/rfc6520`.

6.4.2 Testing the Heartbeat Protocol

We applied APE to the TLS heartbeat protocol in two different Ubuntu 12.04 servers running Apache with OpenSSL. One server was vulnerable to the heartbleed bug, and the other server was patched against the bug. APE used the the payload modification option described in Section 6.3.1 to test these implementations.

Figure 6.4 shows the layout of a heartbeat request message. The heartbeat protocol is layered on top of the TLS record protocol. The record header (the TLS protocol refers to messages as records) consists of three fields making up 5 bytes: `record_type`, `protocol_version`, and `record_length`. A `record_type` of `0x15` denotes a heartbeat message. The `record_length` is a 2-byte integer specifying the length of the remaining fields of the heartbeat message: `heartbeat_type`, `payload_length`, an arbitrary `payload`, and `padding`. The padding is random data intended to be ignored by the server.

As described in Section 6.3.1, we configured APE to modify four of these fields: `record_length`, `payload_length`, `payload`, and `padding`. The `payload` and `padding` fields are each a randomly sized series of random bits. The `payload_length` field is uniformly randomly chosen to be either the size of the payload field, a random integer larger than the size of the `payload` field, or a random integer smaller than the size of the `payload` field. Similarly, the `record_length` field is uniformly at randomly chosen to either be the size of the entire record, a random integer larger than that size, or a random integer smaller than that size.

As described in Section 6.3.2.4, APE first uses exploration to create a basic model of operation for the heartbeat protocol without modifying the messages. Figure 6.5 shows the model of the TLS heartbeat message behavior that is the result of this exploration. For clarity, the displayed model abstracts away multiple TLS messages used for the initial handshake into the `-hello` message. After exploring this initial model, APE uses the message modification method to modify the `-heartbeat` message, again, as described in

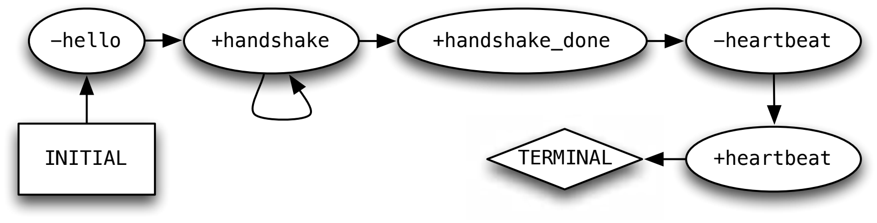


Figure 6.5: A model inferred by APE using guided exploration without message modification. After inferring this model, APE then explored the effects of modifying the heartbeat message.

Section 6.3.2.4. APE generated a total of 100 traces per server, with message modification, only inferring a single model after generating all the executions.

When exploring the vulnerable target server implementation, 22 of the 100 traces included a heartbeat received from the server. Because we used the APE option to represent each modified sent message, each of these 22 traces is represented as a distinct path in the model. Of these 22 paths, 8 included a heartbeat message that triggered the heartbleed vulnerability; i.e., our violation characteristic was a heartbeat response included more bytes than sum of the payload and padding of the original message.

For our patched target server, only 8 of the 100 paths lead to heartbeat events, and all of those correspond to valid heartbeat messages. In other words, the patched server, as expected, did not appear vulnerable to the heartbleed bug.

APE-assisted, manual specification violation discovery. APE was able to automatically discover the heartbleed vulnerability in the vulnerable server when asked to look for executions that return more data than in made available in the original heartbeat message. Further, it aided the manual discovery of two other specification violations.

We manually examined the final model APE inferred of the two servers and found two specification violations that all occurred in both servers. First, both servers failed to respond to properly formed heartbeat messages with payloads smaller than 4,073 bytes. The protocol specification imposes no such restriction. Second, while the protocol specifies that the server must be silently discard the heartbeat message if the total length of the message is greater than 2^{14} bytes, APE observed multiple instances of both servers responding to such messages.

6.5 Analyzing BitTorrent

In this section, we describe the application of APE to several implementations of the BitTorrent protocol. We chose BitTorrent because of its popularity, and because the protocol is implemented by many clients, which allows us to evaluate APE's effectiveness in testing different implementations of the same protocol. APE found one bug and two unexpected uses of the specification [16] in three popular BitTorrent clients: μTorrent, Azureus, and Transmission. Further, manual exploration of the APE-discovered bug led us to discover two more bugs, one that causes the client to crash, and one that artificially inflates the client's upload rate.

6.5.1 The BitTorrent Protocol

BitTorrent is a peer-to-peer file sharing protocol. To share files, a peer creates a small metadata *torrent* file that contains information on finding other peers, file size, and a list of hashes of parts of the file for integrity checking. Peers' BitTorrent clients, after discovering each other via *trackers* that maintain a list of peers' IPs, share files by requesting *pieces* of the file, and responding to others' requests. Each piece's integrity can be verified using the torrent's hashes. The piece request messages contain information of about the index, offset, and length of the requested pieces. Pieces are usually downloaded in *chunks* of 16KB, so it may take multiple request messages to download a single piece.

Reciprocation is a very important aspect of the BitTorrent protocol. While implementation are free to use whatever algorithm to promote reciprocation, in general, a client is more likely to trade with peers that the client deems to be contributing. As such, peers can send a `choke` message to other peers whom they choose not to communicate with, and may send an `unchoke` message to resume communication. To enable new peers, who do not yet have any data to share, to download data, BitTorrent clients will periodically send `unchoke` messages to random peers in a process called *optimistic unchoking*. This allows new peers to send several piece requests and begin downloading pieces even without prior reciprocation.

6.5.2 Applying APE to BitTorrent Clients

We tested three popular BitTorrent clients running on Ubuntu 12.04: μTorrent, Azureus, and Transmission[3]. We did not modify these target clients in any way. We did wrap each with a simple script to clean up after each execution of the client; this step was necessary to ensure that target client did not carry over any state from a previous execution, such as downloaded pieces or peer lists.

To create BitTorrent message definitions for APE to use, we modified an existing open source implementation, AutonomoTorrent [4], adding logging, additional message types (`bitfield_all`, `bitfield_some`, and `bitfield_none` messages, described below), and hooks for APE to use to send BitTorrent's nine message types [16]. We chose Autonomo-Torrent, instead of developing message-sending methods from scratch, to take advantage of its code for tasks such as getting peer lists from the tracker, checking piece hash values, and file IO. We observe that APE users may often choose to reuse existing protocol client implementations to ease the task of writing methods that send protocol messages. Further, we note that we were able to use this single message-sending method implementation for all three BitTorrent targets, and could use it to test any other BitTorrent client.

During normal operation, peers use the `bitfield` message to advertise which pieces they have already downloaded. For APE, we found it useful to split the bitfield into three different message types to indicate if the peer is advertising all, some, or none of the pieces of the file. BitTorrent peers will display different behavior depending on the bitfield value. For example, if APE advertises itself as having all the pieces by sending a `bitfield_all` message, then the other peers will never unchoke it.

[3]Azureus 4.3.0.6-5, Transmission 2.51-0-ubuntu-1.3, and uTorrent Server v3.0 build 27079

During exploration, we configured APE to start the target implementation with half of the torrent already downloaded. This allowed APE to both download and share pieces. We used a custom torrent file and private tracker to ensure that each exploration run consisted of only two peers: the APE tester and the target.

We did not use APE's optional message modification capability for BitTorrent.

6.5.3 Avoiding Reciprocation

We were interested in searching for specification violations in BitTorrent clients that allow a peer to avoid reciprocation, downloading data without uploading. We thus encoded a violation characteristic to check for the peer downloading pieces without uploading.

APE discovered and verified around 100 execution paths across the three implementations that satisfied this characteristic, meaning it discovered multiple ways of violating this specification. We manually examined all these paths to categorize the distinct strategies for avoiding reciprocation, although many could quickly be categorized as very similar. As an example, one strategy exploited control messages to alter the peer's behavior. We now enumerate these three strategies:

1. The choke/unchoke cycle. During normal operation, if the client receives and ignores a `request` message from a peer, that peer sends a `choke` message and no longer responds to messages from the client. However, APE discovered a bug that allows a peer to ignore `requests` without repercussions. Whenever receiving a `request` message, if the client responds by sending a `choke`, followed immediately by an `unchoke` message, (in lieu of the requested piece), the peer sends a new `request` and continues responding to the client's requests. We refer to this as the choke/unchoke cycle. All three tested implementations exhibit this vulnerability.

The next two strategies do not explicitly violate the specification, but result in an unexpected ability to avoid reciprocation.

2. The new guy on the block. The client can pretend to be new to the network by sending a `bitfield_none` messages, even when it actually has data [95]. This falsehood causes other peers to share data without expecting to receive any data in return. Further, APE discovered that to remain a freeloading client, it must never send a `have` message claiming to have the data other peers need.

3. The parrot. An extension of the above strategy is for the client to report in the `bitfield` message only those pieces that the other peers already have.

After APE identified the choke/unchoke cycle, we used APE to apply the exploit of this violation — sending one `choke` and one `unchoke` message in response to each received `request` message — to each of the three target BitTorrent implementations. For all three implementations, this exploit caused a drastic increase in both the total number of `requests` (thousands per minute), and the number of distinct `requests`. In other words, other peers often they requested different pieces. This could be used, for example, to selectively respond to peers only to specific piece `requests`. Over a five minute timespan, Azureus, μTorrent, and Transmission sent 172, 593, and 420 distinct piece requests, respectively. Without sending `choke` and `unchoke` messages, those numbers dropped to 4, 5, and 15 distinct piece requests, respectively.

This exploration led us to uncover two other related bugs in the μTorrent implementation. First, when running in resource-constrained environments (a virtual machine with less than 512MB of RAM), μTorrent would periodically crash when receiving many `choke` and `unchoke` messages. Second, μTorrent appeared to include the `choke` and `unchoke` messages when calculating the peer's download rate, evidenced by the μTorrent's UI. Neither of these bugs were discovered directly by APE, but APE's discovering led directly to manually finding these bugs.

6.6 Evaluation of APE Performance

To find vulnerabilities in target implementations, APE must explore the very large state space of the target's possible behavior. This section evaluates APE's efficiency in exploring that behavior to find vulnerabilities.

Our evaluation focuses on the TRACKER server. While TRACKER is small, its size limits the randomness of the search process and allows us to reliably measure how quickly APE finds vulnerabilities. We ran 30 trials of APE discovering vulnerabilities in TRACKER. Each trial consisted of 100 iterations; each iteration used guided exploration to generate one new trace, and then inferred a new Synoptic model of the target's behavior using all observed traces. We use these trials to answer three research questions.

RQ1: How quickly does APE find vulnerabilities in TRACKER?

Recall that the buggy TRACKER server from Section 6.2 has two vulnerabilities. First, an attacker can elicit a `+clients_response` by sending a `-clients_request` even without previously sending a `-hello`. Second, an attacker can cause the server to erroneously announce its presence multiple times by sending multiple `-hello` messages. To answer RQ1, we examined each of the 3,000 models, generated over the 30 trials, to find at which point in the exploration process, the model encoded each of the two vulnerabilities.

The first vulnerability is simple for APE to find since it only needs to generate a model with a path that starts with `-clients_request`. In our experiments, each of the 30 trials found this vulnerability within the first three exploration traces. In 10 of the trials, the very first trace revealed the vulnerability. This result is consistent with the random exploration procedure, which dominates APE's guided exploration when the model is empty or very small. The procedure picks, at random, a message to send from the three sendable, previously unsent messages. One third of the time (10/30) APE picks the `-clients_request` messages, and in all instances, after three tries, APE has tried all three messages.

The second vulnerability, causing the target to erroneously send multiple `+clients_response` messages, is harder to find. In our experiments, APE found, and verified this vulnerability in each of the trials after exploring at most six traces. In some cases, the guided exploration generated a trace with multiple `+clients_response` messages. In other cases, model inference *predicted* that such behavior is allowed.

Running APE on TRACKER took no longer than a few minutes on average to discover both vulnerabilities.

RQ2: How quickly does guided exploration learn most of the target's behavior?

Critical to APE's success is being able to explore its target's behavior quickly. While it likely takes a long time to explore the behavior exhaustively, APE's guided exploration is able to explore *most* of the behavior after relatively few iterations of trace generation and model inference. Using model size (the sum of the number of states and transitions) as an estimate of the measure of how much of the target's behavior has been explored, Figure 6.6 shows that the mean amount of behavior APE has not learnt diminishes exponentially with time. This finding is not obvious as it implies that (1) the TRACKER server implementation's behavior measure is bounded, and (2) APE can learn it quickly. Other exploration strategies could learn slowly, forcing more time to be spent in the guided exploration stage, and resulting in poor scaling to larger targets. Note that a threat to validity of this result is that it is possible that some behavior cannot be discovered by APE, so the claim that APE discovers the behavior quickly only applies to that behavior that APE can discover.

RQ3: How does the predictability of the model change as APE explores the target's behavior?

The models APE infers from observed executions are predictive, in that they describe both the observed behavior and likely unobserved behavior that satisfies a set of observed executions' temporal properties [11]. The model inference can be inaccurate if the temporal properties that hold for the observed executions do not accurately describe the target system. Thus, for the predictive ability of the model to be accurate, guided exploration must quickly eliminate spurious temporal properties that are observed only because of lack of diversity of the observed traces.

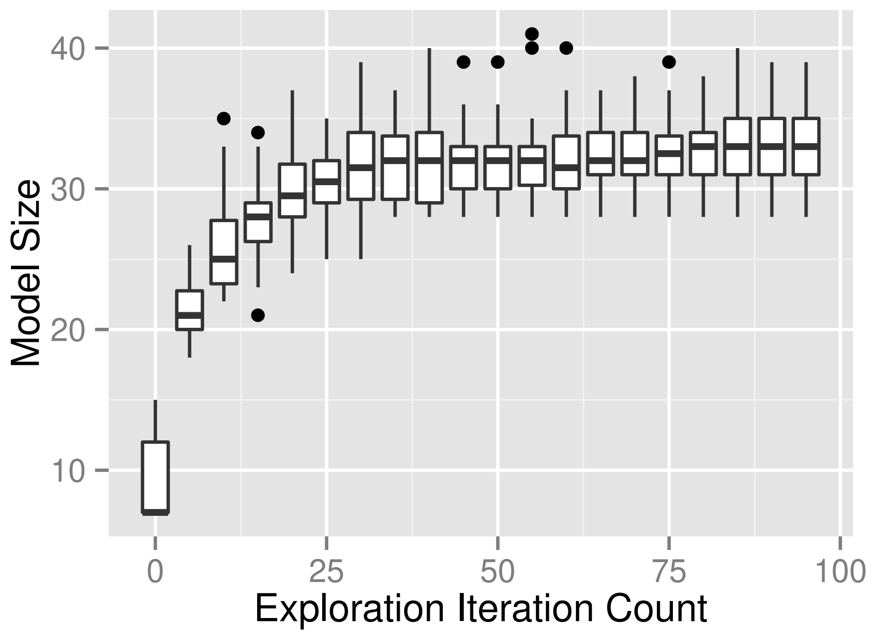


Figure 6.6: The size of the model (the sum of the number of states and transitions) approximates the amount of the target's behavior learnt by APE. APE's guided exploration learns the behavior and approaches the limit quickly.

Figure 6.7 shows that the mean number of mined temporal properties after each trace diminishes quickly. This means that after relatively few iterations of trace generation and model inference, the model's predictive ability is accurate.

An astute reader will notice that in the first iterations, the number of temporal properties increases, whereas it decreases monotonically afterwards. This phenomenon is due to Synoptic's property filtering; the inference ignores properties between messages it has never observed co-occur in a trace. At first, new traces increase the number of messages that have been observed co-occurring, increasing false (and valid) temporal properties. Further exploration eliminates the false properties.

While sets of observed executions with poor diversity often satisfy many temporal properties, systems typically have few true temporal properties [11]. For TRACKER, the final inferred models satisfied only a single temporal property: `-clients_request` is always followed by `+clients_response`.

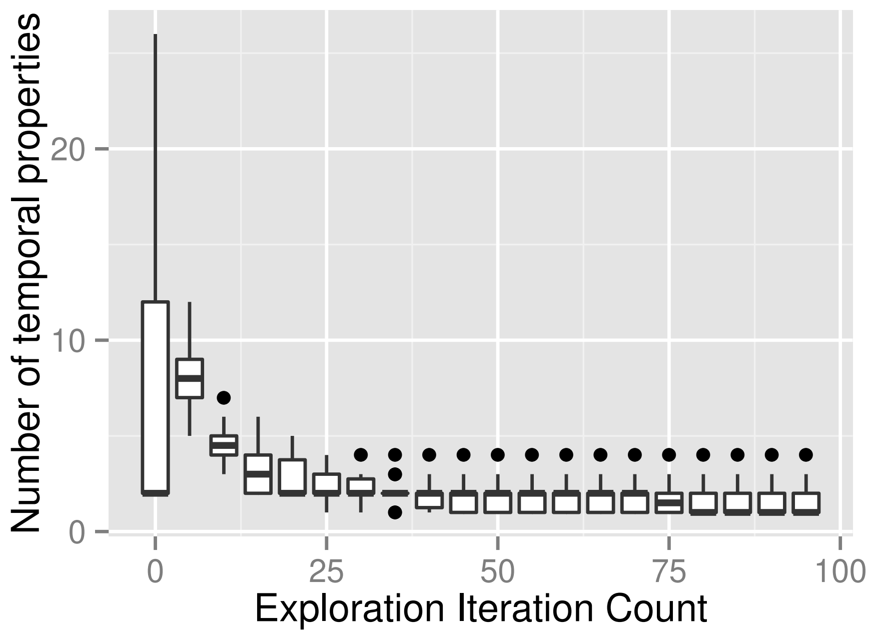


Figure 6.7: The accuracy of the model's predictive ability depends on the guided exploration's ability to diversify observed executions and eliminate false temporal properties. APE accomplishes this and quickly eliminates temporal properties that are artifacts of small sets of explored executions.

6.7 Related Work

The closest research to APE are ProCrawl [134], a system for inferring behavioral models of web applications, and MACE [29], a system for vulnerability detection. ProCrawl also explores a system's behavior, and like APE, it guides the exploration based on model-inference. However, unlike APE, ProCrawl does not target networked systems and does not allow modifying messages. Instead, it focuses on method call sequences, and exploring orders of those sequences to explore unexplored behavior and generate tests. Meanwhile MACE uses symbolic and concrete execution, guided by model inference, to discover vulnerabilities. MACE's use of symbolic and concrete executions to explore a program's state space is complementary to APE's dynamic approach. Related prior work [35,60,71] had relied on user-specified descriptions of an input abstraction function, whereas MACE does not, but still requires an output abstraction function for the output. These are similar to APE's message definitions.

Related research on model inference is complementary to our work, and APE can directly benefit from improvements in model inference. APE uses predictive model inference to abstract the observed executions into a concise and predictive model. In general, inferring the most concise model from observed examples is NP-complete [2,34,59], and most model-inference algorithms approximate a solution [9–11,14,25,38,42,58,88,94,96,126]. APE uses Synoptic [11], which relies on mining temporal properties to abstract the observed executions and balance running time against the conciseness of the final model. Synoptic is nondeterministic, which adds to the nondeterminism already inherent in APE, although deterministic, temporal-property-based model inference [9] could be used instead.

While, in practice, we found Synoptic to work quite well, APE can benefit directly from improvements to the model-inference algorithms. Further, other model-inference algorithms may produce more precise models in particular domains. For example, Li et al. [88] introduce a temporal-property-based inference specific to reactive systems, although they do not directly produce the kind of FSM model APE needs. Other approaches that do produce FSM-based models can be directly substituted for Synoptic. Among these are the kTails [14], the GK-Tails [96], and the RPNI [25] algorithms.

APE's payload modification is similar to protocol fuzz testing [5,60,71], but APE guides exploration, instead of randomly searching through the state space. APE also aims to automate more of the exploration than, e.g., SNOOZE [5] and Gorbunov et al. [60].

Protocol reverse engineering [22,23,40] automates inferring message format, which can be used to automate APE even further. Prospex [35] combines reverse engineering message format and fuzz testing and can generate test inputs, but unlike APE, is not aimed at automatically discovering specification violations.

Cho et al. [30] propose a method for inferring complete protocol state machines with respect to some given subset of the input alphabet. Note, that enumerating the entire input alphabet from experiments alone is currently an unsolved problem. They adapt L^*, an active inference approach, to operate in online environment. They check the resulting model using random sampling. Cho et al. also demonstrate how the inferred model can be analyzed to find weaknesses or derive other interesting properties of the implementation. For example, they compare the FSMs for two different bot C&C servers and find that they both communicate to the same SMTP server making this a weak common link. They accomplish this by associating messages with the server IP address and projecting one FSM onto the alphabet of the other. Like other inference techniques, their work assumes alphabet abstraction functions and resettability.

Gatling [85] finds performance attacks in large scale distributed networks. Like APE, Gatling simulates behavior on the network with peers, although using multiple peers is APE's future feature at this time. While Gatling allows changing message fields and dropping messages, APE is more general in its message modifications. Meanwhile some systems focus on proving specific protocol properties, such as authentication and authorization properties [70]. By contrast, APE is more general and can discover any specification violation that can be described by a violation characteristic method.

6.8 Summary and Conclusions

We have presented APE, a technique for automatically discovering and verifying specification violations in networked software. APE explores the very large space of behavior by dynamically inferring precise models of behavior, stimulating unobserved behavior likely to lead to violations, and iterating by refining the behavior models with the new, stimulated behavior. In evaluating APE, we verify the known Heartbleed bug in OpenSSL, and find seven other specification violations or unexpected behaviors in OpenSSL and three popular BitTorrent clients. Our prototype implementation, and its evaluation show great promise for using model inference, together with fuzz testing, to automatically find bugs, verify bug patches, identify related exploits of known bugs, and augment and generate test suites.

CHAPTER 7
SUMMARY

Motivated by the emerging trends in digital forensics of scale, diversity, and online environments, we developed inference-based techniques for extracting information from a diverse set of evidence sources. We evaluated our work using a U.S. legal foundation and in the context of two common investigative scenarios: mobile phone triage and network-based investigations.

For phone triage, we show that probabilistic parsing is effective for extracting records from a variety of phone models, including those that were not previously examined. Our primary insight is that while specific encodings and formats are different between phones, the underlying semantic structure is often preserved. In other words, a call log record is typically a name, number, and time stamp stored in close proximity. Further, we employed a hash-based filtering approach to quickly reduce the amount of raw data that needs to be examined.

When applied to smart phones, we find that our triage techniques are useful, but not sufficient for handling the thousands of inferred results. To address this issue, we designed a technique for ranking results using a small amount of investigator feedback. We find that by leveraging a few manually labeled true positives, we can create a strong ranking of the most relevant results.

Our phone techniques do not make any assumptions about the operating or file system, allowing them to be applied directly to new models and, potentially, other embedded devices.

For network-based investigations we first quantified and characterized the extent of CP trafficking on p2p networks. We have shown that the majority of files are shared by a relatively small portion of the population, but these peers are spread across many different countries making it infeasible to arrest them all. We also find that the anonymous communication platform, Tor, does not pose a significant problem for investigators in practice — most peers seen on Tor were later observed connecting from a non-Tor, and presumably trackable, IP address.

Finally, we took the first step toward the automated development of tools for investigating network protocols. Our work focused on efficiently exploring the behavior of protocol implementations to search for bugs and other behaviors that investigators can exploit to further an investigation.

Collectively, these projects all share the common theme of forensic triage: quickly collecting information from diverse sources to *(i)* generate leads, *(ii)* prioritize resources, and *(iii)* generally assist investigators in the early stages of an investigation.

BIBLIOGRAPHY

[1] Acquisti, Alessandro, and Grossklags, Jens. Privacy and rationality in individual decision making. *IEEE Security & Privacy 3* (January 2005), 26–33.

[2] Angluin, Dana. Finding Patterns Common to a Set of Strings. *Journal of Computer and System Sciences 21*, 1 (1980), 46–62.

[3] Anvik, John, Hiew, Lyndon, and Murphy, Gail C. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering* (Shanghai, China, 2006), pp. 361–370.

[4] AutonomoTorrent. `http://github.com/abhijeeth/AutonomoTorrent`.

[5] Banks, G., Cova, M., Felmetsger, V., Almeroth, K., Kemmerer, R., and Vigna, G. Snooze: toward a stateful network protocol fuzzer. *Information Security* (2006), 343–358.

[6] BBC News. 'Miserable failure' links to Bush. `http://news.bbc.co.uk/2/hi/americas/3298443.stm`, December 7 2003.

[7] Beebe, Nicole Lang, Clark, Jan Guynes, Dietrich, Glenn B, Ko, Myung S, and Ko, Daijin. Post-retrieval search hit clustering to improve information retrieval effectiveness: Two digital forensics case studies. *Decision Support Systems 51*, 4 (2011), 732–744.

[8] Berger, Yigael, Wool, Avishai, and Yeredor, Arie. Dictionary Attacks Using Keyboard Acoustic Emanations. In *Proc. ACM CCS* (2006), pp. 245–254.

[9] Beschastnikh, Ivan, Brun, Yuriy, Abrahamson, Jenny, Ernst, Michael D., and Krishnamurthy, Arvind. Unifying fsm-inference algorithms through declarative specification. In *Proceedings of the 35th International Conference on Software Engineering (ICSE)* (San Francisco, CA, USA, May 2013), pp. 252–261.

[10] Beschastnikh, Ivan, Brun, Yuriy, Ernst, Michael D., and Krishnamurthy, Arvind. Inferring models of concurrent systems from logs of their behavior with csight. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)* (Hyderabad, India, June 2014).

[11] Beschastnikh, Ivan, Brun, Yuriy, Schneider, Sigurd, Sloan, Michael, and Ernst, Michael D. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proceedings of the 8th Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE11)* (Szeged, Hungary, September 2011), pp. 267–277.

[12] Beverly, Robert, Garfinkel, Simson, and Cardwell, Greg. Forensic Carving of Network Packets and Associated Data Structures. In *Proc. DFRWS Digital Forensics Research Conference* (Aug. 2011), pp. 78–89.

[13] Bianchini, Monica, Gori, Marco, and Scarselli, Franco. Inside PageRank. *Trans. Inter. Tech. 5*, 1 (2005), 92–128.

[14] Biermann, A. W., and Feldman, J. A. On the Synthesis of Finite-State Machines from Samples of Their Behavior. *IEEE TC 21*, 6 (1972), 592–597.

[15] Bishop, Matt. *Computer Security: Art and Science*. Addison Wesley Professional, 2003.

[16] The bittorrent protocol specification. `http://www.bittorrent.org/beps/bep_0003.html`, 2012.

[17] Bond, Mike. Leaving Room for the Bad Guys. In *Proc. Financial Cryptography* (2007), p. 1.

[18] Bourke, Michael L, Fragomeli, Lance, Detar, Paul J, Sullivan, Michael A, Meyle, Edward, and O'Riordan, Mark. The use of tactical polygraph with sex offenders. *Journal of Sexual Aggression*, ahead-of-print (2014), 1–14.

[19] Bratus, Sergey, Lembree, Ashlyn, and Shubina, Anna. Software on the Witness Stand: What Should It Take for Us to Trust It? In *Proc. Intl. Conf. Trust and Trustworthy Computing* (2010), pp. 396–416.

[20] Britton, Tom, Jeng, Lisa, Carver, Graham, Cheak, Paul, and Katzenellenbogen, Tomer. Reversible debugging software. Tech. rep., University of Cambridge, Judge Business School, 2013.

[21] Burch, Andrea M., Durose, Matthew R., and Walsh, Kelly A. *Census of Publicly Funded Forensic Crime Laboratories, 2009*. No. NCJ 238252. Bureau of Justice Statistics, August 2012.

[22] Caballero, J., Poosankam, P., Kreibich, C., and Song, D. Dispatcher: Enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 621–634.

[23] Caballero, J., Yin, H., Liang, Z., and Song, D. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), ACM, pp. 317–329.

[24] Caballero, Juan, Grier, Chris, Kreibich, Christian, and Paxson, Vern. Measuring pay-per-install: the commoditization of malware distribution. In *Proc. USENIX Security* (Aug. 2011), pp. 1–13.

[25] Carrasco, Rafael C., and Oncina, José. Learning Stochastic Regular Grammars by Means of a State Merging Method. In *The 2nd International Colloquium on Grammatical Inference and Applications* (1994).

[26] Carrier, Brian, and Shields., Clay. The Session Token Protocol for Forensics and Traceback. *ACM TISSEC 7*, 3 (2007), 333–362.

[27] Cha, Meeyoung, Kwak, Haewoon, Rodriguez, Pablo, Ahn, Yong-Yeol, and Moon, Sue. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *Proc. ACM IMC* (2007), pp. 1–14.

[28] Cheng, Alice, and Friedman, Eric. Sybil-proof Reputation Mechanisms. In *Proc. Wkshp on Econ of P2P Systems* (August 2005), pp. 128–132.

[29] Cho, C.Y., Babic, D., Poosankam, P., Chen, K.Z., Wu, E.X.J., and Song, D. Mace: Model-inference-assisted concolic exploration for protocol and vulnerability discovery. In *USENIX Security* (2011), vol. 11.

[30] Cho, C.Y., Shin, E.C.R., Song, D., et al. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security* (2010), ACM, pp. 426–439.

[31] Christin, Nicolas, Weigend, Andreas, and Chuang, John. Content availability, pollution and poisoning in file sharing peer-to-peer networks. In *Proc. ACM Electronic Commerce (EC)* (2005), pp. 68–77.

[32] Chu, J.C., Labonte, K.S., and Levine, B. Availability and locality measurements of peer-to-peer file systems. *Proc. SPIE 4868* (2002), 310.

[33] Clark, David D., and Landau, Susan. The Problem isn't Attribution: It's Multi-Stage Attacks. In *Proc. ACM Re-Architecting the Internet Workshop (ReARCH)* (2010), pp. 11:1–11:6.

[34] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. Counterexample-guided Abstraction Refinement. In *Computer Aided Verification* (2000), pp. 154–169.

[35] Comparetti, P.M., Wondracek, G., Kruegel, C., and Kirda, E. Prospex: Protocol specification extraction. In *Security and Privacy, 2009 30th IEEE Symposium on* (2009), IEEE, pp. 110–125.

[36] Computer Crime and Intellectual Property Section, U.S. Department of Justice. Retention Periods of Major Cellular Service Providers. `http://dgsearch.no-ip.biz/rnrfiles/retention.pdf`, August 2010.

[37] Consumer Reports. Online exposure. Social networks, mobile phones, and scams can threaten your security. *Consumer Reports Magazine* (June 2011).

[38] Cook, Jonathan E., and Wolf, Alexander L. Discovering models of software processes from event-based data. *ACM ToSEM 7*, 3 (1998), 215–249.

[39] Cozzie, Anthony, Stratton, Frank, Xue, Hui, and King, Samuel T. Digging For Data Structures. In *Proc. USENIX OSDI Symposium* (Dec 2008), pp. 255–266.

[40] Cui, W., Kannan, J., and Wang, H.J. Discoverer: Automatic protocol reverse engineering from network traces. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium* (2007), pp. 1–14.

[41] Cui, Weidong, Peinado, Marcus, Chen, Karl, Wang, Helen J., and Irun-Briz, Luis. Tupni: automatic reverse engineering of input formats. In *Proc. ACM CCS* (2008), pp. 391–402.

[42] Dallmeier, Valentin, Lindig, Christian, Wasylkowski, Andrzej, and Zeller, Andreas. Mining object behavior with ADABU. In *International Workshop on Dynamic Analysis (WODA)* (2006).

[43] Dinaburg, Artem, Royal, Paul, Sharif, Monirul, and Lee, Wenke. Ether: Malware Analysis via Hardware Virtualization Extensions. In *Proc. ACM CCS* (Oct 2008), pp. 51–62.

[44] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The Second-Generation Onion Router. In *Proc. USENIX Security Symposium* (Aug. 2004), pp. 303–320.

[45] Douceur, J. The Sybil Attack. In *Proc. Intl Wkshp on Peer-to-Peer Systems (IPTPS)* (Mar. 2002).

[46] Eisenbarth, Thomas, Kasper, Timo, Moradi, Amir, Paar, Christof, Salmasizadeh, Mahmoud, and Shalmani, Mohammad. On the Power of Power Analysis in the Real World. In *Proc. CRYPTO* (Aug. 2008), pp. 203–220.

[47] Fairbanks, Kevin D. An analysis of ext4 for digital forensics. In *Proc. DFRWS Digital Forensics Research Conference* (2012).

[48] Federal Rules of Civil Procedure. Rule 26, Duty to Disclose; General Provisions Regarding Discovery. `http://www.law.cornell.edu/rules/frcp/rule_26`.

[49] Fisher, Kathleen, Walker, David, and Zhu, Kenny Q. LearnPADS: automatic tool generation from ad hoc data. In *Proc. ACM SIGMOD* (2008), pp. 1299–1302.

[50] Fisher, Kathleen, Walker, David, Zhu, Kenny Q., and White, Peter. From dirt to shovels: fully automatic tool generation from ad hoc data. In *Proc. ACM POPL* (2008), pp. 421–434.

[51] Foster, Kristina. *Using Distinct Sectors in Media Sampling and Full Media Analysis to Detect Presence of Documents from a Corpus*. PhD thesis, Monterey, California. Naval Postgraduate School, 2012.

[52] Franklin, Jason, Paxson, Vern, Perrig, Adrian, and Savage, Stefan. An inquiry into the nature and causes of the wealth of internet miscreants. In *Proc. ACM CCS* (Oct. 2007), pp. 375–388.

[53] Fu, Xinwen, Graham, Bryan, Bettati, Riccardo, and Zhao, Wei. Active Traffic Analysis Attacks and Countermeasures. In *Proc. Intl. Conf. on Computer Networks and Mobile Computing* (Oct. 2003), pp. 31–39.

[54] Garfinkel, Simson. Digital Forensics Research: The Next 10 Years. In *Proc. DFRWS Annual Forensics Research Conference* (Aug 2010).

[55] Garfinkel, Simson, Nelson, Alex, White, Douglas, and Roussev, Vassil. Using purpose-built functions and block hashes to enable small block and sub-file forensics. In *Proc. DFRWS Annual Forensics Research Conference* (Aug 2010), pp. 13–23.

[56] Garfinkel, Simson L. Carving contiguous and fragmented files with fast object validation. *Digital Investigation 4, Supplement*, 0 (2007), 2 – 12.

[57] Garfinkel, Simson L. Digital media triage with bulk data analysis and bulk_extractor. *Computers & Security 32* (2013), 56–72.

[58] Ghezzi, Carlo, Pezzè, Mauro, Sama, Michele, and Tamburrelli, Giordano. Mining behavior models from user-intensive web applications. In *Proceedings of the International Conference on Software Engineering (ICSE)* (2014).

[59] Gold, E. Mark. Language Identification in the Limit. *Information and Control 10*, 5 (1967), 447–474.

[60] Gorbunov, S., and Rosenbloom, A. Autofuzz: Automated network protocol fuzzing framework. *IJCSNS 10*, 8 (2010), 239.

[61] Greenwald, Glenn. Government harassing and intimidating Bradley Manning supporters. `http://www.salon.com/news/wikileaks/index.html?story=/opinion/greenwald/2010/11/09/manning`, November 2010.

[62] Guevin, Jennifer. Michigan police refute claims of data-collection wrongdoing. `http://news.cnet.com/8301-1009_3-20055961-83.html`, April 2011.

[63] Gummadi, Krishna, Dunn, Richard, Saroiu, Stefan, Gribble, Steven, Levy, Henry, and Zahorjan, John. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. *SIGOPS Oper. Syst. Rev. 37* (Dec. 2003), 314–329.

[64] Halderman, J. Alex, Schoen, Seth D., Heninger, Nadia, Clarkson, William, Paul, William, Calandrino, Joseph A., Feldman, Ariel J., Appelbaum, Jacob, and Felten, Edward W. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM 52* (May 2009), 91–98.

[65] Hawley, Kip. TSA's Take on the Atlantic Article. `http://www.tsa.gov/blog/2008/10/tsas-take-on-atlantic-article.html`, Oct 21 2008.

[66] CVE-2014-0160. `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160`, 2014.

[67] Hoog, Andrew. *Android forensics: investigation, analysis and mobile security for Google Android*. Elsevier, 2011.

[68] Hopcroft, J. E., Motwani, Rajeev, and Ullman, J. D. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2000.

[69] Howard, Ty. Don't Cache Out Your Case. *Berkeley Technology Law Journal 19* (Fall 2004), 1157–1575.

[70] Hsu, Y., and Lee, D. Authentication and authorization protocol security property analysis with trace inclusion transformation and online minimization. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on* (2010), IEEE, pp. 164–173.

[71] Hsu, Y., Shu, G., and Lee, D. A model-based approach to security flaw detection of network protocol implementations. In *IEEE International Conference on Network Protocols. ICNP 2008.* (2008), IEEE, pp. 114–123.

[72] Hughes, D., Walkerdine, J., Coulson, G., and Gibson, S. Peer-to-peer: is deviant behavior the norm on P2P file-sharing networks? *IEEE Distributed Systems Online 7*, 2 (Feb. 2006).

[73] Hughes, Danny, Rayson, Paul, Walkerdine, James, Lee, Kevin, Greenwood, Phil, Rashid, Awais, May-Chahal, Corinne, and Brennan, Margaret. Supporting law enforcement in digital communities through natural language analysis. In *Proc. Intl Workshop Computational Forensics* (Berlin, Heidelberg, 2008), pp. 122–134.

[74] Järvelin, Kalervo, and Kekäläinen, Jaana. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS) 20*, 4 (2002), 422–446.

[75] Jonkers, Kevin. The forensic use of mobile phone flasher boxes. *Digital Investigation 6*, 3–4 (May 2010), 168–178.

[76] Judish, Nathan, et al. Searching and Seizing Computers and Obtaining Electronic Evidence in Criminal Investigations. US Dept. of Justice, 2009.

[77] Kanich, Chris, Weavery, Nicholas, McCoy, Damon, Halvorson, Tristan, Kreibichy, Christian, Levchenko, Kirill, Paxson, Vern, Voelker, Geoffrey, and Savage, Stefan. Show me the money: characterizing spam-advertised revenue. In *Proc. USENIX Security* (Aug 2011).

[78] Karp, Richard, and Rabin, Michael. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev. 31*, 2 (March 1987), 249–260.

[79] Kerr, Orin. Ex ante regulation of computer search and seizure. *Virginia Law Review 96*, 6 (October 2010), 1241–1293.

[80] Kerr, Orin S.˙ *Computer Crime Law*, 2nd ed. West (Thomson Reuters), October 2009.

[81] Kocher, P., Jaffe, J., and Jun, B. Differential Power Analysis: Leaking Secrets. In *Proc. CRYPTO* (1999), pp. 388–397.

[82] Kohno, Tadayoshi, Broido, Andre, and Claffy, Kimberly C. Remote physical device fingerprinting. *IEEE Trans. Dependable Sec. Comput. 2*, 2 (2005), 93–108.

[83] Krishnan, Srinivas, Snow, Kevin Z., and Monrose, Fabian. Trail of Bytes: Efficient Support for Forensic Analysis. In *Proc. ACM CCS* (Oct. 2010), pp. 50–60.

[84] Latapy, M., Magnien, C., and Fournier, R. Quantifying paedophile queries in a large p2p system. In *Prof. IEEE INFOCOM* (April 2011), pp. 401–405.

[85] Lee, H., Seibert, J., Killian, C., and Nita-Rotaru, C. Gatling: Automatic attack discovery in large-scale distributed systems. In *Network & Distributed System Security Symposium. NDSS 2012.* (2012), USENIX.

[86] Legal Information Institute. Uniform commercial code locator. `http://www.law.cornell.edu/uniform/evidence.html`, March 2003.

[87] Levchenko, Kirill, Pitsillidis, Andreas, Chachra, Neha, Enright, Brandon, Félegyházi, Márk, Grier, Chris, Halvorson, Tristan, Kanich, Chris, Kreibich, Christian, Liu, He, McCoy, Damon, Weaver, Nicholas, Paxson, Vern, Voelker, Geoffrey, and Savage, Stefan. Click trajectories: End-to-end analysis of the spam value chain. In *Proc. IEEE Symp. Security & Privacy* (Nov 2011), pp. 431–446.

[88] Li, Wenchao, Dworkin, Lili, and Seshia, Sanjit A. Mining assumptions for synthesis. In *9th International Conference on Formal Methods and Models for Codesign (MEMOCODE)* (July 2011).

[89] Liberatore, Marc, Erdely, Robert, Kerle, Thomas, Levine, Brian Neil, and Shields, Clay. Forensic Investigation of Peer-to-Peer File Sharing Networks. In *Proc. DFRWS Annual Digital Forensics Research Conference* (August 2010).

[90] Liberatore, Marc, and Levine, Brian Neil. Inferring the Source of Encrypted HTTP Connections. In *Proc. ACM CCS* (Oct. 2006), pp. 255–263.

[91] Liberatore, Marc, Levine, Brian Neil, and Shields, Clay. Strengthening Forensic Investigations of Child Pornography on P2P Networks. In *Proc. ACM Conference on Future Networking Technologies (CoNEXT)* (November 2010).

[92] Lin, Zhiqiang, Rhee, Junghwan, Wu, Chao, Zhang, Xiangyu, and Xu, Dongyan. Dimsum: Discovering semantic data of interest from un-mappable memory with confidence. In *Proceedings of the 19th ISOC Network and Distributed System Security Symposium, San Diego, CA* (2012).

[93] Ling, Zhen, Luo, Junzhou, Yu, Wei, Fu, Xinwen, Xuan, Dong, and Jia, Weijia. A new Cell Counter Based Attack Against TOR. In *Proc. ACM CCS* (Oct. 2009), pp. 578–589.

[94] Lo, David, Mariani, Leonardo, and Pezzè, Mauro. Automatic Steering of Behavioral Model Inference. In *ESEC/FSE* (2009).

[95] Locher, T., Moor, P., Schmid, S., and Wattenhofer, R. Free riding in BitTorrent is cheap. In *Workshop on Hot Topics in Networks* (November 2006).

[96] Lorenzoli, Davide, Mariani, Leonardo, and Pezzè, Mauro. Automatic Generation of Software Behavioral Models. In *ICSE* (2008).

[97] Lou, Xiaosong, and Hwang, Kai. Collusive Piracy Prevention in P2P Content Delivery Networks. *IEEE Transactions on Computers 58*, 7 (July 2009), 970–983.

[98] Luk, R.W.P., and Damper, R.I. Inference of letter-phoneme correspondences by delimiting and dynamic time warping techniques. In *IEEE Intl Conf on Acoustics, Speech, and Signal Processing* (Mar. 1992), vol. 2, pp. 61–64.

[99] Manils, Pere, Chaabane, Abdelberi, Le Blond, Stevens, Kaafar, Mohamed, Castelluccia, Claude, Legout, Arnaud, and Dabbous, Walid. Compromising Tor Anonymity Exploiting P2P Information Leakage. In *Proc. HotPets* (July 2010), (See also `http://blog.torproject.org/blog/bittorrent-over-tor-isnt-good-idea`).

[100] Marturana, Fabio, Me, Gianluigi, Berte, Rosamaria, and Tacconi, Simone. A quantitative approach to triaging in mobile forensics. In *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on* (2011), IEEE, pp. 582–588.

[101] Marturana, Fabio, and Tacconi, Simone. A machine learning-based triage methodology for automated categorization of digital media. *Digital Investigation 10*, 2 (2013), 193–204.

[102] Menasche, Daniel, Rocha, Antonio, Li, Bin, Towsley, Don, and Venkataramani, Arun. Content availability and bundling in swarming systems. In *Proc. ACM CoNext* (2009), pp. 121–132.

[103] Milanesi, Roberta Cozza Carolina, Nguyen, Tuong Huy, Lu, CK, Zimmermann, Annette, Sato, Atsuro, Vergne, Hugues De La, and Gupta, Anshul. Market Share Analysis: Mobile Devices, Worldwide, 4Q10 and 2010. `http://www.gartner.com/DisplayDocument?id=1542114`; see also `http://tinyurl.com/4zandx4`, Feb 2011.

[104] Mislan, Richard P., Casey, Eoghan, and Kessler, Gary C. The growing need for on-scene triage of mobile devices. *Digital Investigation 6*, 3-4 (2010), 112–124.

[105] Motoyama, Marti, McCoy, Damon, Levchenko, Kirill, Savage, Stefan, and Voelker, Geoffrey. Dirty jobs: the role of freelance labor in web service abuse. In *Proc. USENIX Security* (Aug 2011).

[106] Murdoch, Steven. Hot or Not: Revealing Hidden Services by their Clock Skew. In *Proc. ACM CCS* (Oct. 2006), pp. 27–36.

[107] National Gang Intelligence Center. National Gang Threat Assessment 2009. Tech. Rep. Document ID: 2009-M0335-001, US Dept. of Justice, `http://www.usdoj.gov/ndic/pubs32/32146`, Jan 2009.

[108] National Research Council. *Strengthening Forensic Science in the United States: A Path Forward.* The National Academies Press, February 2009.

[109] Novak, R. Side-channel Attack on Substitution Blocks. In *Proc. Applied Cryptography and Network Security* (Oct. 2003), pp. 307—318.

[110] Ohmann, Tony, Thai, Kevin, Beschastnikh, Ivan, and Brun, Yuriy. Mining precise performance-aware behavioral models from existing instrumentation. In *Proceedings of the New Ideas and Emerging Results Track at the International Conference on Software Engineering (ICSE14)* (Hyderabad, India, June 2014).

[111] Otto, John, Sánchez, Mario, Choffnes, David, Bustamante, Fabián, and Siganos, Georgos. On blind mice and the elephant: understanding the network impact of a large distributed system. In *Proc. ACM Sigcomm* (Aug 2011), pp. 110–121.

[112] Pal, A., and Memon, N. The evolution of file carving. *Signal Processing Magazine, IEEE 26*, 2 (March 2009), 59 –71.

[113] Pal, Anandabrata, Sencar, Husrev T., and Memon, Nasir. Detecting file fragmentation point using sequential hypothesis testing. *Digital Investigation 5*, S1 (2008), 2–13.

[114] Park, Jungheum, Chung, Hyunji, and Lee, Sangjin. Forensic analysis techniques for fragmented flash memory pages in smartphones. *Digital Investigation 9*, 2 (2012), 109–118.

[115] Payne, B., Carbone, M., and Lee, W. Secure and Flexible Monitoring of Virtual Machines. In *Proc. Computer Security Applications Conference (ACSAC)* (Dec. 2007).

[116] Peterson, Joseph L., and Hickman, Matthew J. *Census of Publicly Funded Forensic Crime Laboratories, 2002*. No. NCJ 207205. Bureau of Justice Statistics, February 2005.

[117] Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., and Venkataramani, A. Do Incentives Build Robustness in BitTorrent? In *Proc. USENIX NSDI Symposium* (April 2007), pp. 1–14.

[118] Piatek, Michael, Isdal, Tomas, Anderson, Thomas, Krishnamurthy, Arvind, and Venkataramani, Arun. Do incentives build robustness in bit torrent. In *Proc. USENIX NSDI* (Apr 2007).

[119] Piatek, Michael, Kohno, Tadayoshi, and Krishnamurthy, Arvind. Challenges and Directions for Monitoring P2P File Sharing Networks. In *Proc. USENIX HotSec* (July 2008).

[120] Ponec, Miroslav, Giura, Paul, Brönnimann, Hervé, and Wein, Joel. Highly Efficient Techniques for Network Forensics. In *Proc. ACM CCS* (Oct. 2007), pp. 150–160.

[121] Prichard, Jeremy, Watters, Paul, and Spiranovic, Caroline. Internet subcultures and pathways to the use of child pornography. *Computer Law and Security Review 27*, 6 (2011), 585–600.

[122] Prusty, Swagatika, Levine, Brian Neil, and Liberatore, Marc. Forensic Investigation of the OneSwarm Anonymous Filesharing System. In *Proc. ACM Conference on Computer & Communications Security (CCS)* (October 2011), p. 13.

[123] Quick, Darren, and Alzaabi, Mohammed. Forensic analysis of the android file system yaffs2.

[124] Reardon, J., Capkun, S., and Basin, D. Data node encrypted file system: Efficient secure deletion of flash memory. In *Proc. USENIX Security Symposium* (2012).

[125] Reardon, J., Marforio, C., Capkun, S., and Basin, D. Secure deletion on log-structured file systems. In *Proc. ASIACCS* (2012).

[126] Reiss, Steven P., and Renieris, Manos. Encoding Program Executions. In *ICSE* (2001).

[127] Rhea, Sean, Geels, Dennis, Roscoe, Timothy, and Kubiatowicz, John. Handling churn in a DHT. In *Proc. USENIX ATEC* (June 2004).

[128] Richard III, Golden G, and Roussev, Vassil. Scalpel: A frugal, high performance file carver. In *Proc. DFRWS Digital Forensics Research Conference* (2005).

[129] Ropelato, J. Internet pornography statistics. `http://internet-filter-review.toptenreviews.com/internet-pornography-statistics.html`, 2007.

[130] Rosenthal, Lee H. A Few Thoughts on Electronic Discovery after December 1, 2006. *Yale Law Journal Online* (November 2006). http://www.yalelawjournal.org/the-yale-law-journal-pocket-part/procedure/a-few-thoughts-on-electronic-discovery-after-december-1,-2006/.

[131] Russell, Stuart, and Norvig, Peter. *Artificial Intelligence: A Modern Approach*, 2nd edition ed. Prentice-Hall, Englewood Cliffs, NJ, 2003.

[132] Rutgaizer, Moshe, Shavitt, Yuval, Vertman, Omer, and Zilberman, Noa. Detecting pedophile activity in bittorrent networks. In *Proc. PAM Conf.* (Vienna, Austria, March 2012).

[133] Savage, Stefan, Wetherall, David, Karlin, Anna, and Anderson, Tom. Practical Network Support for IP Traceback. In *Proc. ACM SIGCOMM* (Aug., 2000), pp. 295–306.

[134] Schur, Matthias, Roth, Andreas, and Zeller, Andreas. Mining behavior models from enterprise web applications. In *Proceedings of the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE)* (2013).

[135] Sekar, Vyas, Xie, Yinglian, Maltz, David, Reiter, Michael, and Zhang, Hui. Toward a Framework for Internet Forensic Analysis. In *Proc. HotNets Workshop* (Nov. 2004).

[136] Shields, Clay, Frieder, Ophir, and Maloof, Mark. A System for the Proactive, Continuous, and Efficient Collection of Digital Forensic Evidence. In *Proc. DFRWS Annual Forensics Research Conference* (Aug 2011).

[137] Spreitzenbarth, Michael, Schmitt, Sven, and Zimmermann, Christian. Reverse engineering of the android file system (yaffs2). Tech. Rep. CS-2011-06, Technische Fakultät -ohne weitere Spezifikation-, 2011.

[138] Steel, Chad. Child pornography in peer-to-peer networks. *Child Abuse & Neglect 33*, 8 (2009), 560–568.

[139] Stutzbach, Daniel, and Rejaie, Reza. Understanding churn in peer-to-peer networks. In *Proc. ACM IMC* (Aug 2006), pp. 189–202.

[140] Symantec. Symantec Internet security threat report. Trends for January 06–June 06. http://eval.symantec.com/mktginfo/enterprise/white_papers/ent-whitepaper_symantec_internet_security_threat_report_x_09_2006.en-us.pdf, September 2006.

[141] Symantec. Symantec Internet security threat report. Trends for 2010. https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf, April 2011.

[142] Tridgell, Andrew. *Efficient Algorithms for Sorting and Synchronization*. PhD thesis, Australian National University, Feb 1999.

[143] Tuttle, John, Walls, Robert J., Learned-Miller, Erik, and Levine, Brian Neil. Reverse Engineering for Mobile Systems Forensics with Ares. In *Proc. ACM Workshop on Insider Threats* (October 2010).

[144] Python — Twisted. `http://twistedmatrix.com/`.

[145] United States Sentencing Commission. Public hearing on federal child pornography crimes. `http://www.ussc.gov/Legislative_and_Public_Affairs/Public_Hearings_and_Meetings/20120215-16/Agenda_15.htm`, February 15, 2012.

[146] U.S. Dept. of Justice. The National Strategy for Child Exploitation Prevention and Interdiction: A Report to Congress. `http://www.projectsafechildhood.gov/docs/natstrategyreport.pdf` pages 19–22, August 2010.

[147] Vidas, Timothy, Zhang, Chengye, and Christin, Nicolas. Toward a general collection methodology for android devices.

[148] Vinterbo, Staal. A Note on the Hardness of the k-Ambiguity Problem. Tech. Rep. DSG-TR-2002-006, Decision Systems Group, Harvard Medical School, June 2002.

[149] Viterbi, A. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory 13*, 2 (April 1967), 260–269.

[150] Votipka, D., Vidas, T., and Christin, N. Passe-partout: A general collection methodology for android devices. *Information Forensics and Security, IEEE Transactions on 8*, 12 (Dec 2013), 1937–1946.

[151] Vuagnoux, Martin, and Pasini, Sylvain. Compromising Electromagnetic Emanations of Wired and Wireless Keyboards. In *Proc. USENIX Security Symposium* (Aug 2009), pp. 1–16.

[152] Walls, Robert J., Learned-Miller, Erik, and Levine, Brian Neil. Forensic Triage for Mobile Phones with DEC0DE. In *Proc. USENIX Security Symposium* (August 2011).

[153] Wang, Xinyuan, Chen, Shiping, and Jajodia, Sushil. Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In *Proc. ACM CCS* (November 2005), pp. 81–91.

[154] Wang, Xinyuan, Reeves, Douglas S., and Wu, Shyhtsun Felix. Inter-Packet Delay Based Correlation for Tracing Encrypted Connections through Stepping Stones. In *Proc. ESORICS* (Oct. 2002), pp. 244–263.

[155] Wei, M., Grupp, L. M., Spada, F. M., and Swanson, S. Reliably erasing data from flash-based solid state drives. In *Proc. USENIX Conference on File and Storage Technologies* (2011).

[156] Whaley, John, Martin, Michael C, and Lam, Monica S. Automatic extraction of object-oriented component interfaces. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)* (2002).

[157] Wolak, Janis, Finkelhor, David, and Mitchell, Kimberly. Child-Pornography Possessors Arrested in Internet-Related Crimes: Findings From the NJOV Study. Tech. rep., National Center for Missing & Exploited Children, 2005.

[158] Xi, Qian, Fisher, Kathleen, Walker, David, and Zhu, Kenny. Ad hoc data and the token ambiguity problem. In *Proc. Intl Symp Practical Aspects of Declarative Languages* (2009), pp. 91–106.

[159] Xie, Yinglian, Yu, Fang, and Abadi, Martin. De-anonymizing the Internet Using Unreliable IDs. In *Proc. ACM SIGCOMM* (Aug. 2009), pp. 75–86.

[160] Yu, Wei, Fu, Xinwen, Graham, Steve, Xuan, Dong, and Zhao, Wei. DSSS-Based Flow Marking Technique for Invisible Traceback. In *Proc. IEEE Symp. Security & Privacy* (May 2007), pp. 18–32.

[161] Zhang, Chao, Dhungel, P., Wu, Di, and Ross, K.W. Unraveling the BitTorrent Ecosystem. *IEEE Transactions on Parallel and Distributed Systems 22*, 7 (July 2011), 1164–1177.