2015

A Framework for anonymous background data delivery and feedback

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

A FRAMEWORK FOR ANONYMOUS BACKGROUND DATA DELIVERY AND FEEDBACK

by

MAXIM TIMCHENKO

B.S., Technion - Israel Institute of Technology, 2002
MBA, Technion - Israel Institute of Technology, 2006

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2015

Approved by

First Reader

Ari Trachtenberg, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Second Reader

David Starobinski, PhD
Professor of Electrical and Computer Engineering
Professor of Systems Engineering

Third Reader

Manuel Egele, PhD
Assistant Professor of Electrical and Computer Engineering

Acknowledgments

I would like to express my appreciation and gratitude to my advisor, Prof. Ari Trachtenberg, for his guidance and insightful discussions, ideas that contributed to this work, and support that allowed me to dedicate most of my time at Boston University to research.

I am also grateful to the rest of my thesis committee: Prof. David Starobinski and Prof. Manuel Egele for their constructive comments and insightful questions. My sincere thanks also goes to Prof. Engin Kirda, whose guest lecture at Boston University had been the origin of an idea that evolved over time into the present work.

I thank my fellow labmates at the Laboratory of Networking and Information Systems for their discussions over many fine lunches and SAGE friday nights, and for sharing their expertise and research in diverse related topics.

I would like to achnowledge the staff at the various IT departments that helped me to set up experiments, provided and tended to temperamental equipment, and provided me with statistical data that made my algorithm assumptions more realistic: James Goebel and Jesse Connell at ENG IT, and Eric Jacobsen at BU IT.

Last but not least, I would like to thank my parents for love, support, and instilling by their example a desire for knowledge and lifelong learning.

A FRAMEWORK FOR ANONYMOUS BACKGROUND DATA DELIVERY AND FEEDBACK

MAXIM TIMCHENKO

ABSTRACT

The current state of the industry's methods of collecting background data reflecting diagnostic and usage information are often opaque and require users to place a lot of trust in the entity receiving the data. For vendors, having a centralized database of potentially sensitive data is a privacy protection headache and a potential liability should a breach of that database occur. Unfortunately, high profile privacy failures are not uncommon, so many individuals and companies are understandably skeptical and choose not to contribute any information. It is a shame, since the data could be used for improving reliability, or getting stronger security, or for valuable academic research into real-world usage patterns.

We propose, implement and evaluate a framework for non-realtime anonymous data collection, aggregation for analysis, and feedback. Departing from the usual "trusted core" approach, we aim to maintain reporters' anonymity even if the centralized part of the system is compromised. We design a peer-to-peer mix network and its protocol that are tuned to the properties of background diagnostic traffic. Our system delivers data to a centralized repository while maintaining (i) source anonymity, (ii) privacy in transit, and (iii) the ability to provide analysis feedback back to the source. By removing the core's ability to identify the source of data and to track users over time, we drastically reduce its attractiveness as a potential attack target and allow vendors to make concrete and verifiable privacy and anonymity claims.

Contents

List of Figures

List of Abbreviations

CPU	.....	Central Processing Unit
DDoS	.....	Distributed Denial of Service
DNS	.....	Domain Name System
GPU	.....	Graphics Processing Unit
IND-CPA	.....	Indistinguishability Under a Chosen Plaintext Attack
IP	.....	Internet Protocol
LAN	.....	Local Area Network
NTP	.....	Network Time Protocol
PIR	.....	Private Information Retrieval
RAM	.....	Random Access Memory
TCP/IP	.....	Transmission Control Protocol / Internet Protocol
TUF-CTXT	.....	Third-Party Unforgeability of Ciphertexts
UDP	.....	User Datagram Protocol
VoIP	.....	Voice Over IP
WER	.....	Windows Error Reporting

Chapter 1

Introduction

Our computers and smart devices regularly and automatically submit a variety of diagnostic information to third parties on our behalf—crash logs, usage metrics, spam samples and so on. This activity happens in the background and is rarely given a lot of thought by many users. Yet this information may contain sensitive data, especially when combined with sender's identity: for example, a crash log that includes the version of the crashed program and the IP address of the reporter can be used to attack the reporter with a known exploit for that program and version; or a location and phone number tuple submitted by a smartphone app to its cloud backend can be used to figure out when a particular person is away from home.

How can a user ensure their sensitive information is not going to be misused? Some companies make lofty privacy protection claims to reassure skeptics, and users tend to trust large corporations to safeguard their data; however, lofty claims are difficult to verify and cyber attacks on large businesses followed by exfiltration of sensitive information occur fairly regularly, with little observable punishment to the company or recourse to the individual. One example of insecure transmission of data by a large company is the Windows Error Reporting (WER) protocol, which performed its first stage in cleartext until March 2014 [62]. Reportedly, this data was being harvested by the National Security Agency to facilitate targeted operations [36]. On Android devices, the Carrier IQ rootkit [14] provided vendor-supported phone monitoring that appears to have included plaintext logging of user keystrokes, and on iOS devices

diagnostic information appears to have been expanded to leak personal data [71].

The popularity of these attacks in the press has made apparent the urgency of researching approaches to quantify and mitigate the risk of data collection applications. Unfortunately, the problem is deceptively difficult, as even encrypted transmission of the data opens side-channels based on timing or publication that breaks an underlying protocol's collusion assumptions [28]. In the case of location data, now commonly collected by many organizations, even four low granularity spatio-temporal points can uniquely identify 95% of individuals in one large-scale human mobility study [44], significantly complicating the use of anonymization options (e.g. differential privacy) in such instances.

The resulting privacy and security concerns cause many individuals and businesses to opt out of diagnostic data collection or to avoid using services that include such collection as mandatory condition. Similar concerns contribute to the difficulty of gathering research data from individuals and smaller organizations. Larger organizations can and do establish direct partnerships and keep all of the collection and analysis activity in-house, thus bypassing many of the concerns, but also greatly limiting the researchers' ability to share and reuse the dataset.

In its typical design incarnation, a data collection system (Figure 1·1) delivers data collected by *reporters* to a *collector*, commonly using a direct TCP connection and sometimes in plaintext. The collector maintains a database of all data and performs some transformations (anonymization, filtering, and so on) and analysis on the data as needed, potentially allowing access to subsets of data for trusted third parties. The privacy and anonymity guarantees of such a setup rely on the integrity of the "trusted core" in which the collector operates. If the core is compromised, its data may be identified back to its source, from collected (non-anonymized) reports or from traffic analysis.

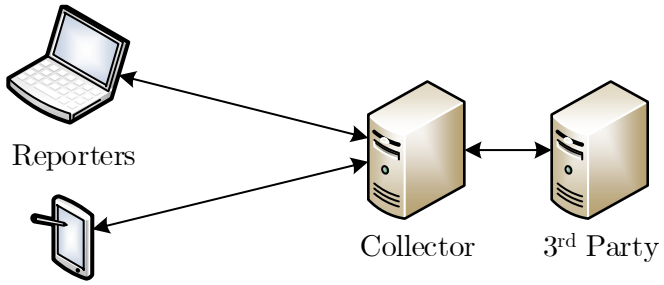

Figure 1·1: Typical high-level design of a data collection system

We thus propose a framework that explicitly removes metadata that may allow the core to track its data sources over time. We do this by shifting long-term tracking logic from the core back to the reporters and utilizing a peer-to-peer network of relays to separate a reporter from the collector. As such, our framework can sit atop emerging peer-to-peer distribution chains, like the rumored functionality for patches and updates in the Windows 10 operating system [30], to provide anonymous delivery. Indeed, the method of simplifying the core and removing any persistent source identifiers is actually suitable for a wide variety of use cases, many of which are difficult to address by a straightforward extension of existing approaches. Our approach aims to mitigate the anonymity and privacy concerns outlined above, while simultaneously providing a clear benefit and feedback (incentive to participate) back to the contributors.

We will focus on three sample use cases:

1. Identifying malicious behavior from Domain Name System (DNS) traffic—query data from individual machines is sent to a collector, aggregated, and passed through an analysis system that flags suspicious requests, which are returned to the reporter.

2. Mapping out names and versions of software packages installed on Linux machines; this follows the motif of Debian Linux's `popularity-contest`, which

attempts to focus development attention on the most popular packages in public use. In this case, feedback consists of a list of vulnerable installed packages and available updates.

3. Collection of mobile location information for traffic prediction, where the value to the user is derived from a prediction system that, by design, cannot track movements of any particular user.

We have three main contributions:

- A complete system design for anonymous delivery of diagnostic data and retrieval of feedback, for networks of limited size.
- A theoretical analysis of the design's security and communication properties and a message level simulation that provides insights into the system's behavior.
- An implementation of the design for a particular use case that can serve as an example for adoption.

The remainder of the thesis is organized as follows: the following section lists the threats we want to address and the assumptions of the design. In chapter 2, we explore existing anonymous communication networks and methods of data anonymization and point out their shortcomings with respect to our threats and assumptions. In chapter 3, we present our system design in detail, followed by the evaluation of its properties in chapter 4. The security analysis of the system against our threat model and known attacks on similar peer-to-peer systems is performed in chapter 5. Several different use cases that can be implemented using the proposed framework are presented in detail in chapter 6. Then, we discuss an implementation of a proposed design for a particular use case in chapter 7 and conclude with a discussion of future work.

1.1 Threat Model

In this section we will examine the possible goals of an adversary and analyze various attack points where an adversary may attempt to break the proposed system.

The main promise of the system is the anonymity of submitted data, so we assume an adversary wants to link between collected data and identities of Internet users, namely to be able to say which IP address has sent each data point or report. Once such link is established, the data may be used for attack targeting, undesired marketing, social engineering and other purposes.

We focus our attention on a broad range of appealing approaches available to an attacker:

Eavesdropping: observing network traffic, potentially on a global scale, and subscribing to and analyzing any public information released by the system: the anonymized data feed and the feedback feed. This is the approach taken by WER harvesting [36] and, with additional manipulation of the observed traffic, by Tor traffic analysis attacks [45].

Running nodes: since we assume our client source code and the network protocol are public, and we do not require any registration or authentication for running a client, an adversary may connect a large number of "customized" nodes to the network and analyze all data flowing through them, as well as generate new control and data traffic, analogous to the use of malicious relays for Tor [67]. An adversary may also achieve the same goal by hacking hosts with security issues and taking over relays running on those hosts.

Peer-to-peer network attacks: as our forwarding layer is a peer-to-peer network, an attacker capable of running nodes can also take advantage of a number of known attacks against peer-to-peer networks, such as Sybil attacks [25], prede-

cessor attacks [69], Eclipse attacks [58], and so on. The goal of these attacks is typically to concentrate the influence of the attacker on particular target nodes or positions within the network to achieve an outsized impact compared to the proportion of attacker-controlled nodes to the total number of nodes on the network. We discuss these attacks separately in Section 5.2.

Attacking the core: we assume that with enough effort (for example, a 0-day vulnerability or a court order), an attacker may may compromise the core and retrieve all stored information, although we assume that access is time-limited (otherwise maintaining validity of feedback becomes hopeless). Recent high profile incidents of this nature include unauthorized access to data at Anthem, the second-largest health insurer in the United States, in February 2015 and at Community Health Systems, a Tennessee-based hospital operator, in August 2014.

Attacking the analysis services: This is a variant of the core attack, where a determined attacker could compromise one or more of the analysis services receiving the anonymized and aggregated feed.

An attacker may also want to disrupt the proposed system by tampering with its functionality, integrity of data, and availability. We consider three attack paradigms of this nature:

Injecting fake data: an adversary could fake arbitrary data to be included into a report (this would require a local presence on the generating host or network), or generate reports with arbitrary information from relays controlled by the adversary.

Tampering with data in transit: an adversary could modify contents of reports travelling over the relay network in transit or via rogue relays controlled by the adversary.

Denial of service: an adversary could generate a large amount of traffic aimed at relays, the collector, or feedback servers in an attempt to exhaust resources and to prevent legitimate traffic from coming through. An adversary could also attempt to find a protocol weakness that would cause resource exhaustion at any of those points even in an absence of an overwhelming amount of traffic.

As an example of this class of attacks, a protocol issue in the Tor network allowed a resource exhaustion attack against a Tor relay, consuming 2MBps of memory on the relay while using just 92KBps of attacker's bandwidth. [37]

An adversary could attempt to devalue the system's feedback to users, causing them to lose interest in further participation in the system. This could be done by:

Impersonation: an attacker may attempt to submit a report as another user containing data that would trigger incorrect feedback to the user.

Confusing analysis systems: an attacker could try to generate data that would cause the analysis systems to miscategorize malicious activity as benign and vice versa.

Tamper or fake feedback: an adversary could attempt to tamper with distribution of feedback data to relays or to generate fake (invalid) feedback data, in order to convey invalid analysis results back to relays.

Finally, an adversary could exploit a problem in the proposed system to achieve unrelated goals, for example:

Attack surface: since a relay would accept a connection from any other relay (or a system pretending to be one), an attacker may attempt to find and exploit a protocol parsing vulnerability that would allow it to take control of the relay process.

Attack amplifier: since UDP is used by the relay protocol, an attacker could attempt to use the relay network as a traffic multiplier that can be used in Distributed Denial of Service (DDoS) attacks by finding a protocol feature that requires a small amount of input / resources and produces a large response. For instance, In February 2014, CloudFlare have observed a largest DDoS attack to date, based on Network Time Protocol (NTP) amplification. There is a protocol implementation that allows the traffic to be multiplied by a factor of 4,670. Czyz et al. provide an extensive overview of the rise and mitigation of NTP attack amplifiers. [19]

Anonymizer: since the relay network offers some anonymizing properties and forwards arbitrary data as content of reports traveling on it, an adversary may attempt to leverage those properties by addressing its reports to a delivery address that does not correspond to the collector, thus generating traffic to an arbitrary address that cannot be easily traced back to the adversary.

1.2 Design Parameters

By considering the attributes specific to background data flows used for diagnostic and statistics, we can define different design goals than would be appropriate for a general-purpose anonymous communication system. With those modified goals, we aim to realize improvements in security properties, effectiveness/ performance and resiliency of a design thus focused, when compared to a general purpose system.

We have chosen the following design goals and constraints for our system based on our threat model:

Anonymity: a low probability that an attacker can identify the source of any *report* submitted by a reporter, or correlate reports to a common source over time.

Privacy: the contents of individual reports should only be understandable by the reporter and the collector; under normal operation, the contents should not be exposed by the collector in non-aggregated form.

Untrusted Core: anonymity and privacy should not be broken completely if the system core is compromised. It is possible that the level of anonymity will degrade depending on the nature of the report data.

Delayed Delivery: we accept delays during transport if it helps to achieve anonymity or privacy. However, we want a fixed fraction of reports to reach the collector within a specified time.

Delayed Feedback: feedback may be delayed due to aggregation, processing time of analysis services, or to improve efficiency; however, almost all feedback should be available within a fixed delay window.

Loss Tolerant: we permit the loss of a small but predictable fraction of reports in order to allow the use of stochastic algorithms. If needed, this may be mitigated with system-level error-control.

Asymmetrical Workload: Clients are expected to submit a large volume of reports to the collector, but the size of feedback (if any) to a given client is small. This suggests different algorithms may be appropriate for forward (reporting) and reverse (feedback) paths.

Arbitrary Data: Some collection systems accept specific types of data, such as numeric ranges or histograms. We prefer to have no such restrictions on the data. The nature of the data may, however, affect the amount of degradation in anonymity if the core is compromised.

Low Friction: to maximize adoption, we do not require registration or authentication from clients—anyone can download the code and add a relay to the network.

High level generic protocol: we do not attempt to provide a network-level solution over which applications can run unmodified, as Tor does, since real-time responsiveness would be required. Our solution defines a new high-level protocol that can be used to satisfy the requirements of a variety of use cases.

Chapter 2

Related Work

The literature generally clusters into three distinct groups: general purpose anonymous communication networks, diagnostic data collection systems that follow the "trusted core" design presented in the introduction, and data collection systems designed for a single specific use case that modify the data or compute a target function before results reach the collector.

The area of general anonymous communication systems has seen a lot of attention in the literature. Systems can be classified based on latency: *low-latency*, suitable for interactive World Wide Web browsing and other synchronous communication, and *high-latency*, suitable for e-mail and other forms of asynchronous communication that can tolerate a large and unpredictable delay. Edman and Yener wrote a comprehensive survey of the area in 2009 [27], including Tor [22], a well-studied representative of the low-latency group, and Mixminion [20], a representative of the high-latency group specialized for e-mail.

Both Tor and Mixminion distinguish between *clients* of the system and the anonymity-providing *core* of the system (i.e., a set of relays for Tor and a set of mixes for MixMinion), resulting in a relatively small and stable population of distinct nodes that constitute the core and a much larger population of transient clients.[1]

[1]For example, between January and September 2014 the MixMinion network had only about 10 recommended nodes on average, dropping to as low as 4 at one instance. (`http://www.noreply.org/mixminion-nodes/`) The Tor network averaged about 4000 relays marked "stable" and 1000 exit relays during the same time period, with estimated 2 million clients. (`https://metrics.torproject.org/network.html`)

While Tor is popular and well understood by the community, and could serve as a conservative option to provide a measure of sender anonymity in the forward direction and receiver anonymity for the feedback, its general-purpose nature would result in a solution that is theoretically less secure than otherwise possible. For one, Tor is just a transport layer, as opposed to the current work which offers a complete end-to-end architecture for a class of applications. Indeed, if no additional protections are taken, a trivial design using Tor would result in a system that has a clear-text, malleable last hop (between the Tor exit relay and the collector) and likely uses a persistent identifier to identify all collected data related to a particular client, enabling the adversary to use other sources of information to deanonymize clients merely by passive observation, without requiring a system core compromise. A comparable design would have to adopt most of the architectural features described in this work, including end-to-end encryption, avoiding persistent identifiers, splitting large reports into independent parts, and so on, to be able to claim a similar level of privacy protection. It would also need a separate source of trust to be able to detect and mitigate a keymapping attack.

Furthermore, Tor and other low-latency networks are vulnerable to traffic correlation attacks, and many such attacks were proposed and evaluated over time [5, 37, 45, 46]. While such attacks are unavoidable if real-time interaction with a remote host is desired, as would be the case with web browsing, in our use case the interaction is between machines and in most scenarios can be significantly delayed. Using a low-latency network in this particular case exposes users to an anonymity and privacy risk with little theoretical benefit, although there is a clear practical benefit to using a proven, high-volume anonymizing network like Tor.

Finally, the popularity and usage of Tor could work against it if a user utilizes Tor for both the proposed system and for other purposes (web browsing, BitTorrent

downloading, etc.) Since Tor reuses circuits for multiple TCP connections [63], it is possible for a client of our system to become deanonymized if the user is concurrently using another service that is either insecure (e.g. HTTP browsing) or highly susceptible to deanonymization [38].

Peer-to-peer networks such as MorphMix [55] and I2P [33] take a different approach than the core/clients model of Tor and MixMinion. In a peer-to-peer network, each node can originate, sink and forward traffic, increasing the diversity of routes but also making the routing more volatile, with the result that an attacker can more easily attach malicious forwarding nodes to the network. Both of those networks are low-latency, and therefore share the weakness of being susceptible to traffic analysis attacks.

Notably, the protocol specification of I2P (and some of the documentation) mentions a possibility of adding an intentional delay during packet processing at intermediate nodes [35]. However, the feature appears to be unimplemented at this time, so further analysis of its suitability to the task is not possible. In particular, if merely a fixed delay at each hop is implemented and the tunnel length is hard-coded for an application, an attacker could simply account for the delay while processing the collected data for traffic correlation.

Peer-to-peer networks have another property that is undesirable in the context of our application. Nodes in peer-to-peer networks such as I2P and MorphMix would likely be asked to relay unrelated traffic as well, some of which could consume non-trivial amount of bandwidth or involve illegal activities. These concerns about unrelated traffic could be a barrier for adoption of a system based on peer-to-peer transport on tightly controlled networks. In contrast, our network does not allow unrelated traffic, as discussed in section 5.6.

Proposed by Reiter and Rubin as early as 1998, Crowds [54] is a peer-to-peer,

low latency anonymous communication network with centralized membership control. Crowds was originally designed to anonymize web browsing. In the Crowds model, a client's traffic is forwarded along a randomly constructed path in the peer-to-peer network, and the response is returned immediately along the same path. During a path's construction, each segment's destination node (a "jondo" in Crowds terms) makes an independent decision on whether to extend the path further or to complete construction and forward the traffic to the destination. The idea of a random walk within a peer-to-peer network is central to our approach of forwarding reports to the collector in the proposed design.

Specialization to a particular class of problems allows us to gain significant advantages over Crowds, which uses a similar path construction algorithm. Since Crowds was designed to proxy HTTP, each Crowds jondo on a path can observe forwarded traffic in plaintext, allowing traffic association attacks with out-of-band knowledge. Our system adds a layer of end-to-end encryption, making association attacks unlikely. By making association attacks difficult we are able to avoid the design trade-off made by Crowds, where association attacks can actually reduce anonymity if the path is changed too frequently since there is a higher chance of forming a random path where all nodes belong to the attacker. Therefore Crowds only changes the random path once in a set period of time (10 minutes) while our system sends each packet along a different random walk.

AP3, or The Anonymous Peer-to-Peer Proxy [42], adopts the random walk forwarding idea from Crowds to establish a one-way path between sender and receiver. This network is notable because it offers two different algorithms: one for the forward direction, and one for the reverse direction; we make a similar distinction in the proposed system. In the reverse direction, the sender constructs another random walk in which the nodes remember their predecessor and agree to forward messages in the

reverse direction for a limited time; the response is forwarded by the receiver to the last peer in the reverse chain and then follows its way to the source. The receiver can find the last peer since the last peer publishes a sender-provided unique "channel id" into the underlying DHT substrate (AP3 is based on Pastry, which is a peer-to-peer based network providing global secure lookup and routing services) and the channel id is included into the original message.

The particular way AP3 choses relays at random (using a secure lookup in the Pastry DHT) creates additional traffic in the Pastry layer that can be observed by an attacker. The impact of this information leak has been evaluated by Mittal and Borisov and has been found to significantly (by a factor of 5) reduce the number of hosts an adversary needs to control to achieve deanonymization, both in the specific parameters case and in the asymptotical bound [43]. In addition to that attack, AP3 is presented as a general-purpose system, and therefore other known attacks can be applied to AP3 with greater chance of success since the network traffic will likely be less uniform or randomized (the predecessor attack, the neighbor selection attack, and so on). Our system avoids a potentially vulnerable composition of two different peer-to-peer layers and, by only allowing traffic with prescribed randomness and content properties, is significantly more resistant to the attacks described above.

The systems mentioned so far aim at achieving a large anonymity set. If that requirement is relaxed to anonymous communication within a smaller group, specialized cryptographic algorithms can be used to construct *group anonymity systems* such as Chaum's DC-nets [15] (for groups of tens of participants) and Dissent [18] (claiming linear scaling cost and group sizes of several thousands of participants). While DC-net is peer-to-peer in principle, Dissent achieves better scalability by separating the system into clients and servers, where the assumption for the server is that it "is run by a respected, technically competent, and administratively independent anonymity

service provider". [68] We will discuss similar requirements when we develop options for scaling the feedback provided by our system. However, even network sizes of thousands of participants are insufficient for large scale data collection: for instance, the Debian `popularity-contest` package receives over 150,000 unique submissions, and many popular applications have tens of millions of users.

Moving from general-purpose systems into the domain of systems whose primary purpose is collection of statistics and diagnostic data, we mention in passing the cluster of systems whose design follows the "trusted core" pattern. One such system, described by Calvert et al. [13], gathers extensive traffic data from home networks and mentions network security as a potential application. All systems of this kind share the risk of a privacy disaster if the "trusted core" is compromised, with the severity of the disaster depending on the types of data collected, ease of reassociation, and retention policies.

The final cluster of work in the literature relates to systems that modify data in transit. If the collector receives the modified data itself, a natural approach involves suppressing attributes, adding noise at the source, and quantifying the values to achieve k-anonymity [61]. k-anonymity is a property of the data set and it means that the information for each individual within the data set is indistinguishable from the information of at least $k - 1$ other individuals within the set. Applications of k-anonymity have been done, for example, by Zhong and Hengartner for location data [72]; a key concern of this approach is that the anonymity set may shrink rapidly with successive queries [6, 44].

On the other hand, if the collector receives only results of a computation on the data, a natural framework comes from differential privacy and distributed, privacy-preserving computations. Informally, differential privacy is a property of an analysis algorithm that guarantees that inclusion or exclusion of a single item of underlying

data has a quantified and very small (at most) impact on the output of the algorithm. Therefore, there is little risk for an individual to submit an item of data into the dataset. A survey of results in the field has been done by Dwork [26].

Examples from this group include the system of Akkus et al. [2] for web analytics based on histograms of user counts, Anonygator [53] computing histograms over general numerical data, RAPPOR [29] used by Google Chrome to discover popularity of particular user settings such as having a specific feature enabled or having a specific web page set as their home page, and PrivEx, which is used to aggregate numerical traffic statistics [28]. Voting and survey systems are usually an extension of this group, with an additional emphasis on strong user identity and allowing only a single vote/survey submission per user per event. Some of the methods used by those systems are computationally expensive: for example, one of the algorithms in PrivEx and the Zhong/Hengartner system use homomorphic encryption primitives, and Anonize (discussed next) uses non-interactive zero knowledge proofs.

A general limitation of differential privacy based systems is their inability to provide responses to each report. For many use cases where this functionality is not required, and only big trends are of interest to the aggregator, this class of systems can provide an effective solution–but low-probability or low-volume events will get lost in the differential privacy-induced noise. For example, when running RAPPOR's algorithm on a large set of strings with exponential popularity distribution, only strings of popularity of 1% and above were reliably detected [29]. In addition, many of the systems listed above are designed to operate on specific data types (histograms, numerical data, and so on) and therefore are a poor fit for highly structured data.

Anonize [34], by Hohenberger et al., while called a survey system, is an exception from the above group since it forwards unmodified data to the collector. The system features anonymous verification of user identity (only selected users may submit data)

and permits only one submission per user per survey. There is a bandwidth and CPU cost to those protections, however, even though it is "very reasonable for such schemes" [34, p. 14].

In summary, while there is plenty of research on general purpose anonymous communication networks and some research on the specific topic of anonymous data collection, we believe our system is a novel contribution to the area. Comparing the proposed system to the related work, the most obvious benefit is that our *untrusted core* requirement removes a single point of failure from the "typical" design. In addition, our *arbitrary data* design parameter permits a greater variety of use cases which have data that cannot be easily modified to provide anonymity using differential privacy techniques. Moreover, the proposed design allows generating a response to each unique sample of data submitted, offers a larger anonymity set than group-based protocols and, unlike voting/survey systems, does not require a separate registration process to obtain a rare and unique user identifier, which could become a barrier to adoption (a violation of our *low friction* design requirement).

Finally, most previous work either does not have a reverse channel (feedback to submitters) at all, or offers a symmetric, identical algorithm for both forward and reverse directions. By separating and optimizing the forward and reverse direction algorithms according to their different properties, we minimize the amount of state and path information that needs to be maintained at network nodes and improve the overall efficiency of the system.

Chapter 3

Design

Our proposed system comprises four major components:

1. A *submission* subsystem, carrying reports from their sources, called reporters or agents, to the collector;

2. A *core* that aggregates collected data and prepares a corresponding feed;

3. Some *analysis systems* (which are out of scope of this paper) that process the core's feed; and,

4. An optional *feedback* subsystem that receives analysis results, combines them, and can both confirm receipt of a report and provide the results of the report's analysis back to the originating agent. The feedback subsystem is optional, if

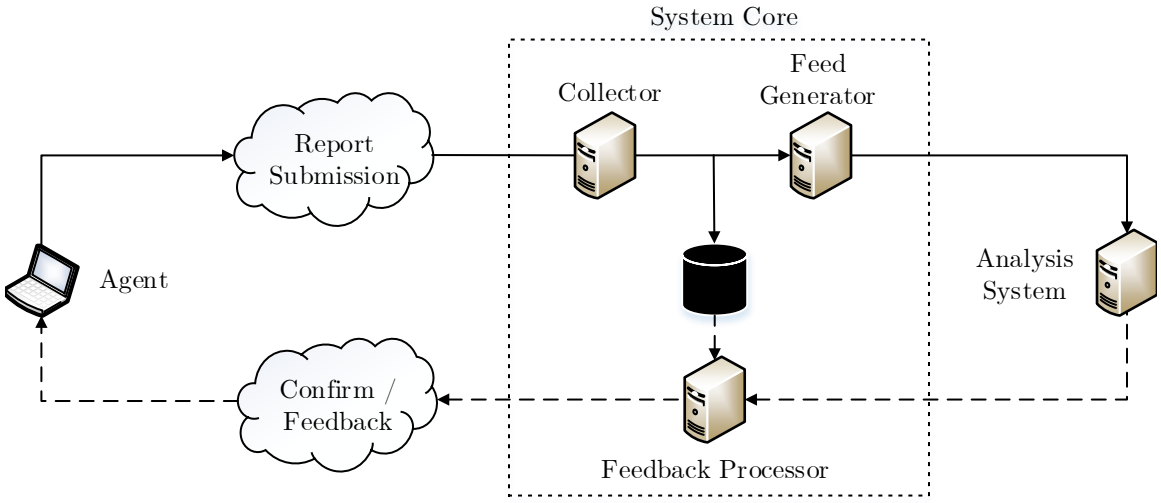

Figure 3·1: Design Overview

the goal is strictly data collection; but then it may be more difficult to convince users to volunteer data.

Since we want the design to be as generic as possible, we treat report contents as opaque anonymized blobs, and, as such, we pad all reports to a fixed, use-case specific size so that they would not be distinguishable by length. Section 3.5 expands on desired data properties, and Section 6.1.2 provides details for our DNS use case.

The following sections explore our submission and feedback subsystems in detail.

3.1 Submission: Peer-To-Peer Mix Network

The report submission subsystem is responsible for delivering reports prepared by agents to the collector, while making it as difficult as possible for an adversary to identify the source of a report or to be able to modify a report undetected. The delivery is not guaranteed, but we aim for a robust and quantifiable chance of success and, for use cases where feedback subsystem is employed, it is possible to provide delivery confirmation and make multiple attempts to deliver a report before giving up.

Each report is prepared by an agent and forwarded to the peer-to-peer node (a *relay*) associated with that agent. The data in the report is encrypted by the agent before handing it over to the relay, and only the collector can decrypt the report and read the submitted information. Typically, both the agent and the relay would be processes running on the same host, but it is possible for several agents running on a trusted LAN to share a relay. For simplicity, we will assume that both agent and relay are colocated on the same physical machine and there are no agents or relays operating "by themselves". Such a combination of an agent and a relay is called a *reporter* in previous chapters.

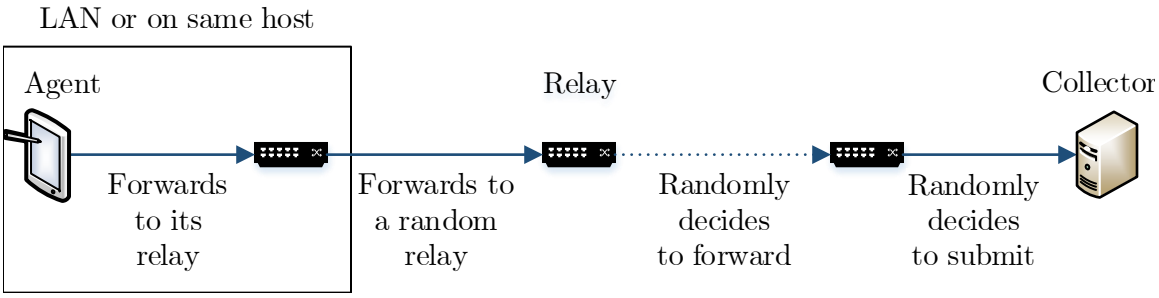

Figure 3·2: Report Submission Subsystem

A relay receiving a report will randomly choose to either submit it to the collector, or to forward the report to another relay in its peer list. An exception is the first relay: to ensure the collector does not see its identity, the relay receiving a report from its agent will always choose to forward it. Since the choice is random (and memoryless, with fixed probability), the length of the forwarding chain is in itself a random variable. Each report, therefore, takes a different path (a random walk) through the relay network.

To complicate timing analysis, reports are not processed immediately. Instead, a mixing pool is maintained by each relay. Incoming and new reports are added to the pool and are assigned a randomly distributed timeout; when the timeout expires, the chosen action is performed. The pool has limited capacity and once it is close to capacity, the relay will refuse forwarding requests in order to distribute the traffic in a uniform fashion across the network.

3.1.1 Bootstrapping

Our initial bootstrapping method is via the collector, since it is the only fixed-address service a relay knows when starting from a "clean slate". A relay can request a list of several active relay addresses from the collector when starting up and use those relays to join the network. This should only happen once when a relay is set up—once it has bootstrapped, it will remember the peers it contacted and will not need to bootstrap

again unless it has been disconnected from the network for an extended amount of time and, due to churn, all the peers in its list are unavailable.

After the network has been running for a while, it is possible for independent lists of long-lived, trusted peers to emerge. We envision an easy way to add such lists via the user interface of the relay. Similar trusted lists of peers were used, for example, by the eDonkey file sharing network: the only way to bootstrap a connection from a client to one of the network's servers was to download a server list (also called a "`server.met`" file) and open it with the eDonkey client.

Once a relay has established links to several active relays, it will request addresses of additional peers from those relays, allowing a small number of bootstrapping links to grow rapidly into a strong connection to the network.

3.1.2 Maintenance

Any peer known to a relay is placed into one of the following three sets:

Active: a relay that we have communicated with successfully in the past. An active relay can be chosen to forward reports, verify credentials of peers, request additional peers from, and so on.

Ready: a relay whose credentials have been verified, but there is no room to add it to the active set.

Potential: a relay whose credentials have been obtained from any source (bootstrap, gossip, or a new relay initiating a connection) but have not yet been verified.

As the relay attempts to forward reports generated locally and received from others, it will refresh the validity of data in the active set. If an active set peer repeatedly fails to respond to a message or consistently returns a busy status when an attempt to forward a report is made, it will be replaced by a new relay from the ready set.

A sufficiently high rate of outgoing reports allows us to keep the active set membership state refreshed without having to perform explicit periodic probes. This is desirable since the control traffic in the network is reduced. A similar approach of refreshing the active set without using dedicated commands is taken by other peer-to-peer networks as well: for example, Kad derives active set updates as a byproduct of network search operations (both of the client itself, and forwarded from other clients). [40]

Since the ready set is not used for forwarding, relays within this set could get stale. We replace old ready set entries with entries from the potential set when their identity is verified. This behavior allows new relays to enter active sets of their peers faster and begin receiving forwarded traffic sooner.

3.1.3 Reliability and Churn

When each relay is empowered to make routing decisions and avoid forwarding reports to hosts that are busy or otherwise unavailable, the overall reliability of the network improves compared to predetermined-routing systems, where a failure of a single host along the route disrupts all of the circuits routed through the host. The higher likelihood of failure is the reason why protocols such as I2P and Tor construct and maintain multiple ready-to-use routing chains ("tunnels" in I2P) at any given time.

Many general-purpose protocols are synchronous, so they tend to time out and retransmit rather quickly (or provide immediate feedback on missed pieces of data). In our delivery network, the delivery confirmation (if utilized) is only received after a significant delay (about one hour in the DNS use case) and retransmission of lost data is undesirable, so maintaining high reliability is particularly important.

The failure of a relay in our design means that all reports buffered on the relay are lost, impacting the rate of successful delivery in the system. Accounting for this introduces a tradeoff in the choice of parameters: increasing the time a report

can wait in a mixing pool of a relay improves anonymity (since more reports will be received and sent by the relay during the time) but reduces the probability of successful delivery.

In general, the lifetime of hosts in peer-to-peer networks follows a heavy-tail distribution: a host that has been up for some time is more likely to remain available [32]. However, for our application, routing reports through a static set of peers could expose a host to deanonymization based on traffic association over time. More importantly, new hosts joining the network should start receiving traffic soon to populate their mix pool, but they would not get any reports if relays preferred older known connections. For those two reasons, we introduce deliberate churn, removing a random relay from the active relays set every so often and replacing it with one from the ready set.

3.1.4 Design Alternatives

We are proposing a completely new peer-to-peer based system and protocol. We could have reused an existing peer-to-peer substrate and build our protocol on top of it; for example, this is the design choice taken by AP3–a network based on the Pastry DHT layer. However, AP3 also illustrates a potential issue in the composition approach, as described by Mittal and Borisov: "... the composition of a secure DHT lookup mechanism with an anonymous communication protocol ... should be carefully analyzed, as it is likely to introduce additional vulnerabilities." [43]. Even if the composition is done carefully, general-purpose peer-to-peer overlays often optimize for interactive applications: for example, in his survey, Wallach observes that Tapestry and Pastry "construct their overly in an Internet topology-aware manner to reduce routing delays and network utilization" [65]. In contrast, our primary goals are anonymity and privacy, while delays or network utilization are a lesser concern due to non-interactive and low bandwidth nature of our traffic. Therefore, we might intentionally choose longer, geographically and organizationally diverse links to re-

duce a chance an adversary with capability of monitoring only a particular part of the network be able to track reports' paths.

3.2 Feedback: a PIR Problem

We look at several forms of potential feedback: delivery acknowledgement, binary feedback for a particular report, and a free-form response.

To maintain anonymity of the reporter, a relay cannot simply ask the collector for feedback for a report with a specific identity, as our goal is to keep the anonymity set of the reporter as large as possible. The general scenario where a server learns nothing about the collector (in an information-theoretic or computational sense) is known as the *Private Information Retrieval* (PIR) problem [17].

A trivial way to implement a PIR scheme is to package the feedback for all clients together and make it available for downloading by relays. This method gives no useful information about what each client sends, but it limits the volume of feedback that can be provided for a particular time interval, thus limiting the network size. Representing the feedback in a compact form and splitting it into multiple layers, where each successive layer is downloaded by a progressively smaller number of relays, helps to reduce the average size of feedback downloaded by a single relay.

3.2.1 Delivery Confirmation

Delivery Confirmation is a binary type of feedback–a report has either been delivered in time or it has been not. We do not require this feedback to be completely accurate, since our design parameters specify that losing a small percentage of the collected diagnostic information is acceptable. In our design, delivery feedback does double duty: it tells the originating relay that a report has been delivered and provides all relays along the report's path an indication that can be used to keep reputation scores for other relays.

We use a Bloom filter to store a set of hashes of delivered reports. The hashes are computed over the encrypted data, which is seen by intermediate relays as well. This lets us use the filter for reputation tracking: if a report seen by a relay has been delivered, then the next hop the report has been forwarded to can have its reputation score increased.

A Bloom filter [10] is a compact structure that stores set membership data and allows querying the set with a tunable probability of a false positive (saying an item belongs to the set when it actually does not). Ordinary Bloom filters have a predetermined size derived from the desired false positive probability and the expected number of elements. Scalable variants have been developed for cases where the expected number of elements is not certain. [3]

To deliver the feedback, the filter and its parameters are serialized to a file, cryptographically protected, and published on the feedback server. Relays download the file and query the filter for hashes of reports they have originated or forwarded. Employing a peer-to-peer protocol, like BitTorrent, to distribute the feedback is possible to reduce the load on the feedback server.

3.2.2 Binary Feedback

General binary feedback can use the same mechanism as delivery confirmation feedback. Considering it separately (instead of just having a combined delivery / binary feedback filter) provides two benefits. First, the feedback is private: unlike delivery confirmation, which is based on a hash that can be computed by any relay on the report's path, binary feedback is based on a secret value stored within the report and known only to the originating relay and the collector. Second, the binary feedback data should be downloaded only if a relay originated a report during the epoch (delivery feedback can be downloaded every epoch, to update reputation). Since this

data is downloaded less often, and we could desire better accuracy, in terms of the chance of a false positive, it could use different parameters for its Bloom filter.

3.2.3 Arbitrary Feedback

In this case as well, we can select for maximum privacy by making the client download the entire feedback set. Depending on relative sizes of resulting files and chance of generation of detailed feedback, it might be preferable to have a small file to be downloaded always or to have binary feedback that indicates whether checking the arbitrary feedback file is necessary.

For each report, its feedback is encrypted by the collector using a key found in the report, making the contents private. An index in the beginning of the file can indicate where the feedback for a particular report can be found.

3.2.4 Design Alternatives for Large Networks

The feedback privacy requirements correspond to the properties of a private information retrieval (PIR) protocol. For a single source of information (a single feedback database), several protocols achieving computational privacy are available. The main drawback of these schemes is the large computational cost for the server. There were a few attempts to reduce the cost, for example by using GPU computations [41]. Even the reduced cost does not scale, however. Borrowing some parameters from the DNS scenario, for 10^6 reports (30,000 relays) a trivial download requires around 3.8Gbps of bandwidth and is within the I/O capacity of one server; a GPU-accelerated PIR would require around 50 mid-tier "System 2" servers from Melchor et al. [41, p.8].

There are other PIR protocols that offer privacy guarantees when the query is distributed among a number of cooperating databases; the protocol guarantees that as long as fewer than a certain number of databases cooperate, none of the databases can discover which data block has been retrieved. We believe this type of protocol

holds promise for a better feedback distribution alternative. A number of servers belonging to multiple geographically distributed organizations can be deployed; each of those servers can obtain a full copy of the feedback data using high-bandwidth connections, and answer queries from relays using one of the existing PIR protocols. An organization can be compensated for hosting a feedback server, or the benefit can be reciprocal: if A hosts a feedback server for B's system and vice versa, both of their systems' privacy properties are strengthened. Implementing a PIR replacement for the trivial feedback method is one of our priorities for future work.

3.3 Timing

If feedback is not desired, there are no particular demands on timing of individual reports—the agent submits a report, and with some fairly high probability the report will arrive at the collector within a certain time. Aggregation and analysis on the collector do not impact the agent or the relay in any way.

The situation is different, however, if we desire any kind of feedback, even a simple one such as an indication of whether the report has been received. Figure 3·3 shows the principal stages in lifetime of a report:

Report Epoch: This identifies a period of time during which a report was generated. To preserve anonymity the exact time is not used. The longer this period is, the larger the resulting anonymity set becomes, but the amount of collected data per-period and feedback size grow as well.

Delivery Window: Since a number of relays a report goes through and the resulting delay are random, a longer delivery window improves the probability a report will be received in time. Having a short delivery window results in a faster round-trip time until feedback is received.

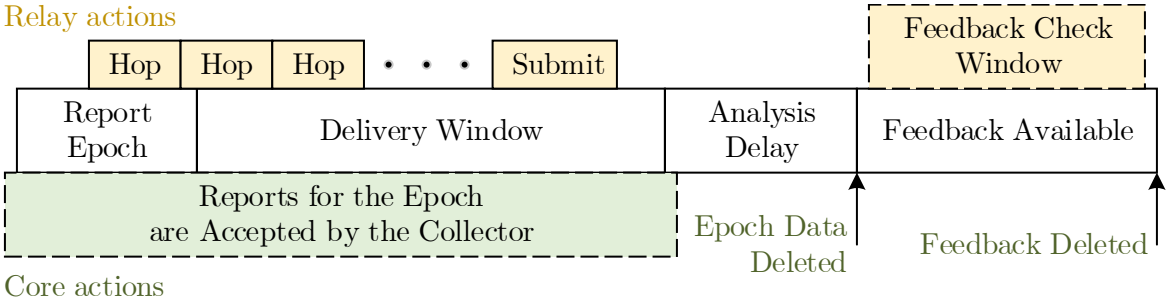

Figure 3·3: Timing of a single reporting epoch

Hop: Represents the average length of time a report spends waiting at a relay before being forwarded.

Report Acceptance Window: The collector accepts reports for a given epoch starting from a little before the beginning of the epoch until slightly past the end of the delivery window. The margins account for the clock drift among different relays. If a collector receives a report for an epoch outside of its acceptance window, the report is discarded.

Analysis Delay: This stage accounts for aggregation of the collected data for the epoch, forwarding of the data to analysis systems, processing of the data by analysis systems, receiving results, and preparing the feedback data. At the end of this stage, all the collected data for the epoch can be discarded.

Feedback Availability Window: During this stage, the feedback for all the reports in the epoch is available for downloading. A relay will choose a random time within this interval to check for feedback, if it needs to. Once this stage ends, all the feedback data for the epoch can be discarded.

The duration of nearly all the stages (except the Analysis Delay) can be freely adjusted to meet the specifications of the particular use case (for example, the desired

probability a report will be successfully delivered) and to fit within the resource limits on any element of the system.

3.4 Cryptography

For report submission, we require two layers of cryptographic protection, with the inner layer applied between an agent and the collector, and the outer layer applied between each pair of relays. Both between pairs of relays and between an agent and the collector, we require a cryptosystem that outputs messages indistinguishable under chosen-plaintext attack (IND-CPA secure) to make sure neither a compromised relay nor a network observer can learn anything about report contents. We also require third-person unforgeability of ciphertexts (TUF-CTXT) to detect modifications to reports made by relays or in transit. Without this feature, a change to a report might show up in anonymized/aggregated data and allow tracking, similar to the "tagging" attack for Tor [22]. The same security properties would also protect privacy and integrity of control traffic between relays in the peer-to-peer network. For any feedback that is distributed publicly, encryption is not possible but we would still like to prevent tampering by requiring TUF-CTXT.

To prevent replay attacks between pairs of relays, we can use the strictly increasing nonce approach. The same approach will not work between a relay and the collector, since unpredictable delivery times make out-of-order delivery likely. We generate new keys instead: an agent would generate a new key pair for each report, with the public part of it also serving as a single use identifier. This ensures two reports from the same source cannot be tied together. Since each report contains its epoch, the overhead of tracking all keys seen for a given epoch is bounded.

An agent retrieves the current collector's public key from the collector when starting up. The agent will verify the received key with other peers to prevent keymapping.

The collector's keys have a set expiry time and are regularly rotated to provide forward secrecy to reports from previous epochs. Relays generate new key pairs whenever their state is reset (typically, when their host is restarted).

3.4.1 Choice of the Cryptographic Library

We have chosen to use *Sodium*, a 2013 fork of *NaCl* [21]. The improvements include cross-compilation support and a simpler build system, as well as a public and active development process. The library offers the same primitives: curve25519 for private key encryption, salsa20 for symmetric encryption, and poly1305 for message authentication.

The library includes all the necessary primitives: public-key authenticated encryption for relay to collector communication, public key signature scheme for shared feedback, and symmetric key encryption that can be an option for communication between relays. Simple APIs make it easier to use the library correctly.

3.5 Data Collection

There are several common patterns for diagnostic data generation. From the perspective of the proposed system, the important parameters are the amount of data for a single sample and the distribution of samples over time. The simplest case is fixed-size, fixed-interval data such as periodic measurements of indoor temperature or of location—the predictability of this case makes choosing the system parameters easy. It is possible, and recommended, to group small samples to improve efficiency. DNS traffic samples are variable in size and come at a variable frequency while still being roughly stable on the global scale.

Identical data can often be structured in different ways. For example, a list of installed software on a new Debian Linux instance comprises 1700 packages, and its trivial (text) representation takes about 200 KBytes. If we wanted to collect that data

once per day, we could put the list into one report and choose a low p_{sub} and a high t_{dw} to ensure each relay has a number of waiting reports in its mix pool. However, a low p_{sub} increases network traffic and a high t_{dw} increases the chance a report will be lost due to nodes going offline. Furthermore, a full list of packages is likely to have fairly low anonymity; like contents of a person's bookshelf, the list would reflect one's occupation, interests, and preferences. The list also changes relatively slowly over time for a given machine.

If the same data is split among smaller reports containing information on just a few packages each, the same host could generate 85 reports of 20 packages each a day. With 80 times as many reports traveling through the network, we can increase p_{sub} and reduce t_{dw} while retaining a healthy pool size on relays and improving the anonymity properties of the data. In addition, much smaller reports would reduce the memory requirements for the mixing pool by at least an order of magnitude.

There are some cases of diagnostic data generation that are not optimal for the proposed system, namely those where the size of a unit of data is very large or where the distribution of samples can vary greatly network-wide. One such example is crash reports. A stable version for some software package can have very few crashes, perhaps one per day per 100 installs; a new alpha version could jump to over 50 per 100. If network parameters are fixed in advance, it might be challenging to find a balance between good anonymity, low resource limits, and a potential for network-wide "gridlock" as a significant fraction of peers fills up their mixing pools and refuses to forward additional reports.

From the anonymity perspective, it is very important to avoid collecting any information that unambiguously reveals the report creator's identity, such as IP addresses or unique serial numbers. There should also be no persistent identifier that links different reports from the same origin together, as it could significantly reduce the

system's resistance to deanonymization attacks [72]. Instead, if such linkage is necessary for the system's functionality, an attempt should be made to provide the needed information in the feedback and perform the linkage on the sender side, instead of in a centralized fashion in the system core (which we assume can be compromised given sufficient effort).

One example of this approach is given in the DNS use case (see Chapter 6): we would like to alert the user once the number of detections from their data crosses a threshold but, instead of keeping a per-user tally in the system core, we return a number of detections per report and the counting and decision making is being made at the source. In this way, we avoid receiving and storing a user-unique identifier and make data deanonymization more difficult.

As discussed in Section 5, under a threat model where system core compromise is possible, a system that collects and stores end-user data has to entertain a possibility of an attacker break-in and a deanonymization attempt using external knowledge. For each use case, possible sources of external knowledge and their correlation to the submitted data should be considered. While the proposed system takes steps to make reassociation difficult (aggressively discarding data and breaking submissions into small reports), it does not provide a guarantee that such an attack will fail. If a guarantee of that kind is required based on the sensitivity of the data or on applicable law, submitting a differentially private result of some function of the data should be considered. This is likely to result in a reduction of utility of the system, however, and the choice between the two approaches is a design tradeoff that needs to be made for each particular use case.

Chapter 4

Evaluation

This section evaluates the theoretical and practical performance of the proposed system, based on a message level simulation. The analysis of the system's resistance to the specified threats follows.

4.1 Theoretical Analysis

We first formalize the description in the previous sections to a set of parameters and algorithms.

The decision on whether to submit or forward a report is made by sampling a random variable $S = \text{Bernoulli}(p_{sub})$ and submitting if $s = 1$, forwarding otherwise. The time a report will remain on a relay is obtained by sampling a random variable $T = Exp(1/t_{hop})$.

For a simplified analysis, we assume that communication between relays is instantaneous. This assumption is good as long as t_{hop} is much larger than actual communication time.

Since S is Bernoulli, the number of times a report is forwarded, N_F, is a geometric random variable. It follows that the mean number of times a report is forwarded before submission is $E[N_F] = 1 + 1/p_{sub}$ and, with the instantaneous communication assumption, that the mean delay between report creation and arrival at the collector is $\frac{t_{hop}(1+p_{sub})}{p_{sub}}$.

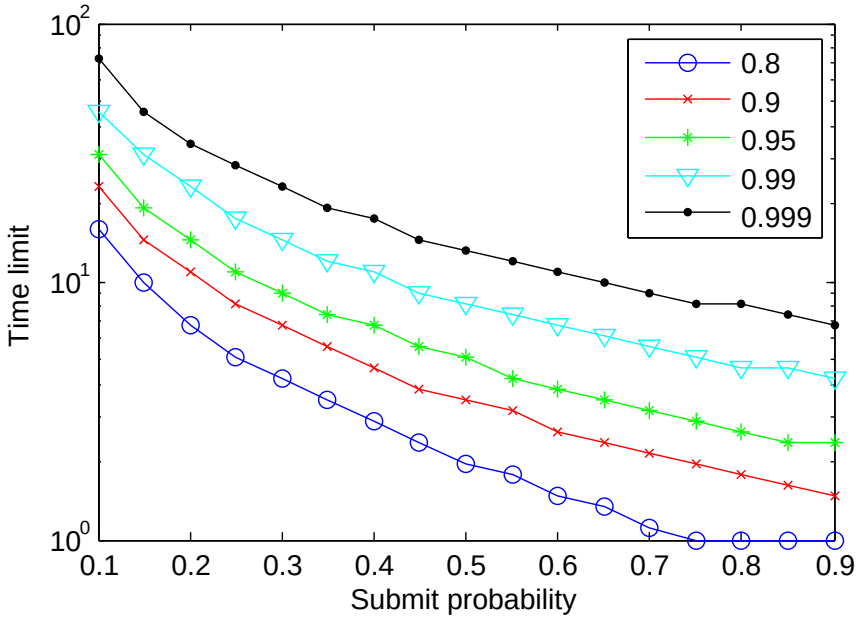

Figure 4·1: Delivery window size necessary to reach a particular delivery probability for a given p_{sub}, with $t_{hop} = 1$ and $t_{ep} = 3$.

We compute the probability a report is delivered on time within the simplified model of instant communication and perfect reliability. Assume a uniform distribution of reports within an epoch: $T_{REP} = \text{Unif}(0, t_{ep})$ where t_{ep} is the length of an epoch. Let t_{dw} be the delivery window (a fixed parameter) and D the total delay of the report, then $D = \sum_{h=1}^{N_F+1} T_h$ and, since T_h are i.i.d exponential variables,

$$D|N_F = \text{Erlang}(N_F + 1, t_{hop}).$$

$$Pr(D < t_{dw} + t_{ep} - T_{REP}) = \sum_{n_f=1}^{\infty} Pr(D < t_{dw} + t_{ep} - T_{REP}|N_f = n_f)Pr(N_f = n_f) =$$

$$\sum_{n_f=1}^{\infty} \frac{1}{t_{ep}} \left(\int_0^{t_{ep}} Pr(D < t_{dw} + t_{ep} - t_{rep}|N_f = n_f, T_{REP} = t_{rep})dt_{rep} \right) Pr(N_f = n_f) =$$

$$\frac{p_{sub}}{t_{ep}} \sum_{n_f=1}^{\infty} (1 - p_{sub})^{n_f - 1} \int_0^{t_{ep}} \frac{\gamma(n_f + 1, t_{hop}(t_{dw} + t))}{n_f!} dt \tag{4.1}$$

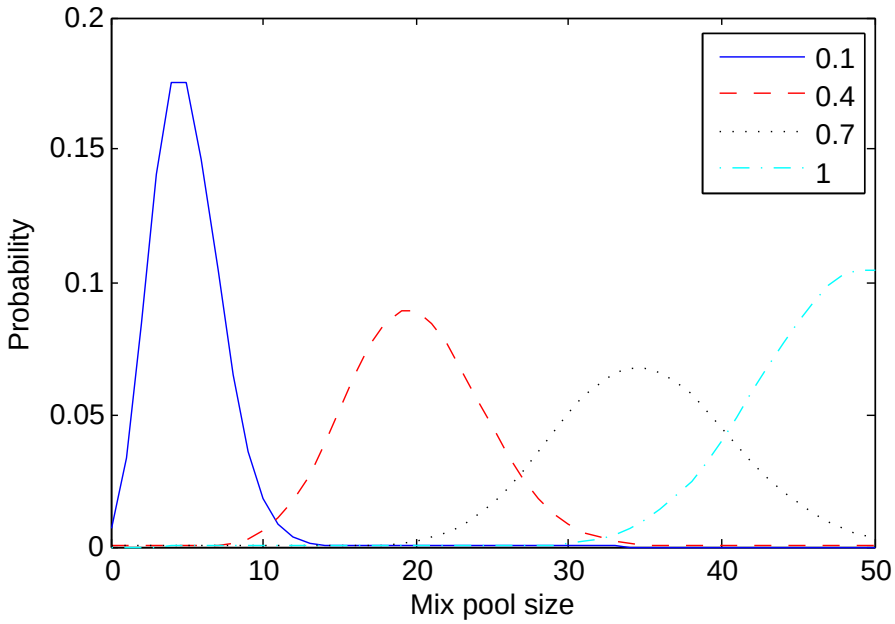

Figure 4·2: Queue probability distribution for some values of t_{hop}, with $p_{sub} = 0.25$, $\lambda = 10$, and $N = 50$.

Figure 4·1 shows the results of numerical computation of the delivery probability for a range of p_{sub} and t_{dw}.

The amount of traffic each relay receives can be obtained by symmetry. There are two sources of traffic: generated by the relay's agent (with an average rate λ) and forwarded to the relay by other relays, at rate x. There are two sinks: reports being submitted ($p_{sub}x$) and reports being forwarded out ($(1 - p_{sub})x + \lambda$). In a stable network, a relay forwards out as much as it receives, namely $x = (1 - p_{sub})x + \lambda$, therefore $x = \lambda/p_{sub}$. It follows that the collector handles the same amount of reporting traffic as it would in a trivial system, but each relay has to handle $2/p_{sub}+1$ times as much reporting traffic (total over both directions).

In general, the creation of reports by an agent does not follow an easy-to-analyze distribution. However, if we assume that arrivals from the agent and from other relays are Poisson (at a rate of $\lambda\frac{p_{sub}+1}{p_{sub}}$), then we obtain a standard M/M/m/m queue. This allows us to optimize the network parameters, in particular t_{hop}, for optimal utilization

of a given queue size (which is limited by resources on the relay). In Figure 4·2 we can see that too low a t_{hop} negatively impacts anonymity (there are only a few reports waiting at the relay), but setting t_{hop} too high can raise the blocking probability to an unacceptable level.

4.1.1 Feedback

The theoretical analysis of the feedback mechanism follows in a straightforward way from the properties of Bloom filters, and we know that filter size (and therefore feedback size) scales linearly with the number of individual reports submitted during a reporting epoch. This is generally true for arbitrary feedback as well, since we can assume a constant fraction of reports require detailed feedback and that the size of arbitrary feedback is constant.

In their analysis, Broder and Mitzenmacher show the minimum size of a Bloom filter, m, to be in the following relationship to the number of entries n and probability of error ϵ [12]:

$$m \geq n \frac{\log_2 1/\epsilon}{\ln 2}$$

By plotting this relationship for a range of filter sizes, error probabilities, and permissible number of filter entries, we obtain Figure 4·3. We estimate that, for different applications, reasonable feedback sizes will be exceeded when the network produces between 10^5 to 10^7 reports per epoch, corresponding to roughly 10^4 to $5 * 10^6$ individual relays. The figure also shows that varying the error rate does not impact the capacity of the filter significantly. These estimates provide the baseline for evaluation of better-scaling (PIR-based) feedback solutions and can assist in locating a switch-over point, meaning the network size where supporting a more complicated feedback system becomes worthwhile.

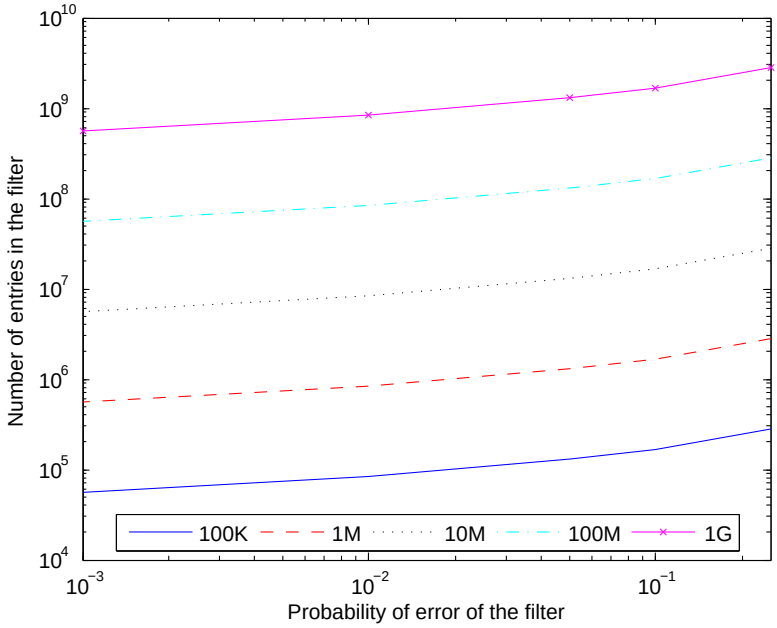

Figure 4·3: Expected maximum number of entries that maintain a given error rate in a Bloom filter for several selected Bloom filter sizes

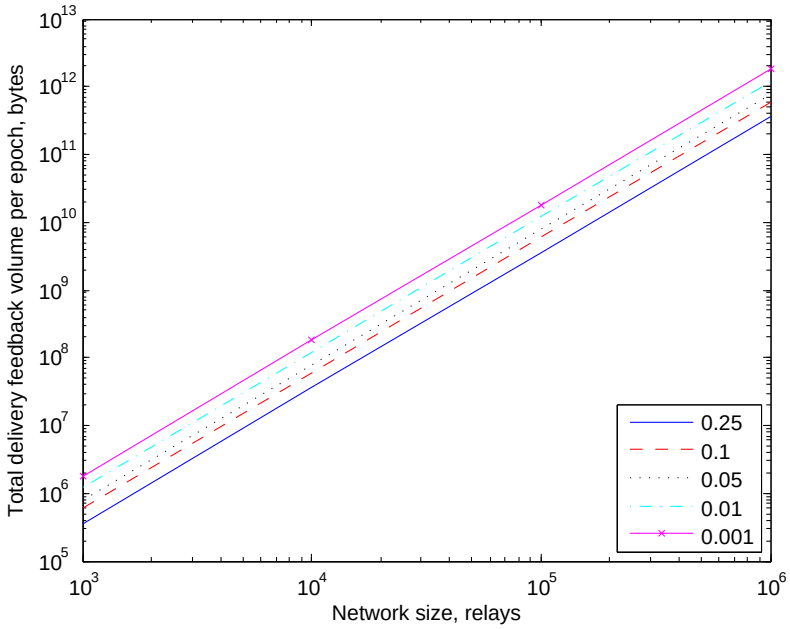

Figure 4·4: Total delivery confirmation feedback traffic with growing network size, for a range of error rates

Another kind of limitation on network size is illustrated in Figure 4·4. In the simplest implementation, all relays that would like to obtain feedback download it from the system core, resulting in a quadratic growth of the total volume of feedback traffic from the core. Taking some assumptions (for example, an epoch length of 15 minutes and report generation rate of 30 reports/epoch for the DNS use case), we can see that at network size of $1.5 * 10^4$ relays the sustained traffic at the core will be around 10 MBps.

Therefore, even in relatively modest networks it becomes important to utilize the peer-to-peer network for spreading the feedback. If every node participates in delivery of feedback, the scaling reverts to linear instead of quadratic. In this case, while the total network traffic remains the same, the system core needs only to seed the feedback files into the peer-to-peer network, and does not have to bear the full bandwidth cost of delivering the data to each relay that requests it.

4.2 Perfect Reliability Simulation

To evaluate the performance of the system and the behavior of the proposed algorithms at scale, we have developed a message-level simulator of the proposed architecture. To speed up the simulation, we omit the serialization of message objects to binary streams and remove cryptographic protection for all the communication. No actual data is simulated since the behavior of the system at this level of detail does not depend on report contents.

The network is bootstrapped with a set of 10 relays, similar to how we would bring up an actual implementation. Then additional relays are joined using a configurable algorithm, currently at one relay per second until the network size is reached. Each relay runs the peer-to-peer forwarding protocol including peer discovery and conges-

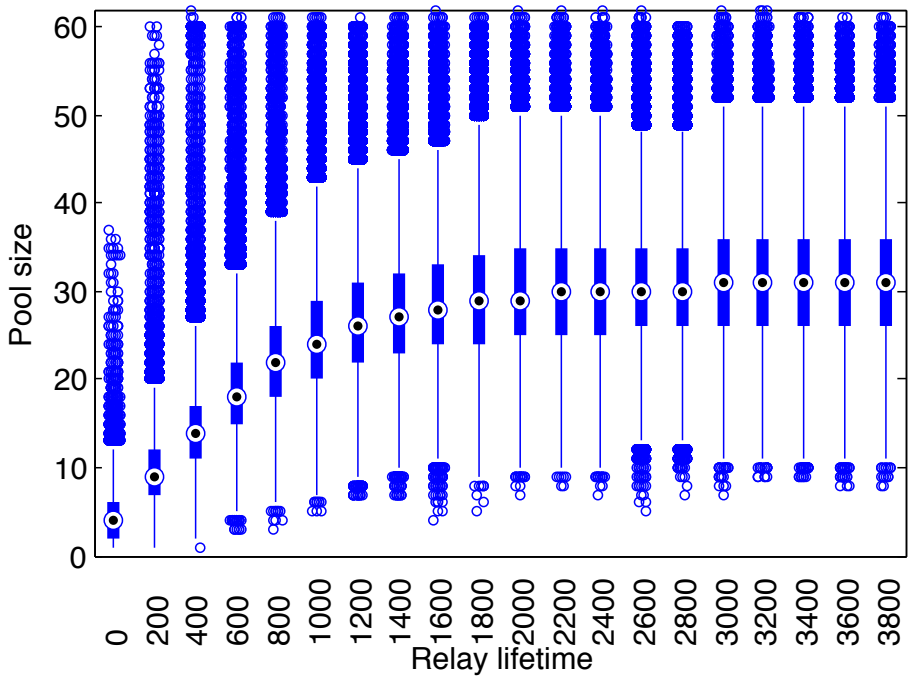

Figure 4·5: Pool size distribution over relay lifetime.

tion management. The reports are generated using a Poisson distribution using the same parameters across all relays.

The simulation is based on SimPy, a discrete event simulation framework written in Python. Running a 5,000-node network simulation for 3 hours of simulated time takes 17 minutes (on an Intel Core i7-2600 with 8 GB RAM) and produces an extensive log of the network's state. During that time, about 1.4 million reports are created. With a goal of 99% successful deliveries and 99% delivery feedback correctness, the simulation results were 99.17% and 99.13% correspondingly over 10 runs. A lost report has been incorrectly seen as delivered by a relay with a probability of 0.000162% (on average over 10 runs), matching the expected performance of the Bloom filter. This shows that our simulation matches well with the theoretical expectations of the system's behavior.

Figure 4·5 shows that a new relay is integrated into the network fairly quickly:

while over the first few minutes there is some chance the relay's mixing pool is still empty, by the 10-minute mark all relays had at least a few reports in their pool. By the 20-minute mark the distribution of mixing pool utilization looks almost the same as the steady-state distribution, which can be seen towards the right edge of the figure.

4.2.1 Implementation Details

The simulation has been designed to be suitable for modeling both a generic, theoretical scenario with algorithmic generation of reports, nodes, and node uptimes and a scenario based on real-world data, such as a collection of DNS traffic over a network of significant size (from which information about nodes and timing/contents of reports generated by those nodes can be derived directly, and node uptimes can be inferred).

To that end, there are three major groups of entities that constitute the simulation, besides the simulation framework itself: the major entities containing the logic of the system (relay, receiver including network bootstrapping, and feedback); the data types (a report, a delivery confirmation containing a Bloom filter, and other minor structures such as a list of relay addresses used for bootstrapping); and scenario- and use-case specific components, including report generation (contents and timing) for each relay, a reliability model for relays in the network, and a class capturing most of the configurable timing and numerical settings (for example, buffer sizes) for other entities.

The overall object graph constructed by the simulation is shown in Figure 4·6. A single simulation- and use-case-specific main file (`sim.py`) instantiates the necessary objects, wires them together, and kicks off the simulation framework to run for the specified length of simulated time.

We attempt to capture all of the use-case-specific parameters (epoch timing and forward/delivery probability) into a single class, `ApplicationParameters`, that can

be replaced to simulate a different use case. In the theoretical simulation it is called `DNSApplicationParameters`.

The SimPy environment is represented by `env` and is needed by any object that needs access to the simulated time, to the framework's functionality of launching additional event-driven processes, and to other environment properties. It may be possible to abstract these environment primitives, which are necessary for any event-driven application, into a more general interface that would allow using the same source files (for example, the Relay functionality) for both the simulation and an actual implementation of the system. However, the differences in syntax between SimPy and other popular event-driven frameworks (for example, *Twisted*[1] or the newer *asyncio*[2], a standard library in Python 3.4) are large enough to make the required adapters non-trivial. We leave this desirable portability/compatibility modification for future work.

The data collection process is modeled by `ReportGenerator`. There is one instance of this class per relay, allowing for either uniform behavior of relays (as is the case in this simulation, where each relay generates reports using a Poisson distribution with identical parameters), a non-uniform behavior where parameters may vary between relays, or a real-world-data based generation where each instance processes a stream of actual data belonging to a particular host and generates reports according to volume and timing of the data. Each generated report is passed to the corresponding `Relay`.

The `Relay` class implements the combined logic of a relay and an agent in the network, performing report forwarding and submission (and peer-to-peer network bootstrapping and maintenance) as the relay portion and doing report generation and feedback verification as the agent portion. A `Relay` interacts with other relays by obtaining their objects through the `Network` class for every interaction, and with

[1] `http://twistedmatrix.com/trac/wiki/FrequentlyAskedQuestions`
[2] `https://docs.python.org/3/library/asyncio.html`

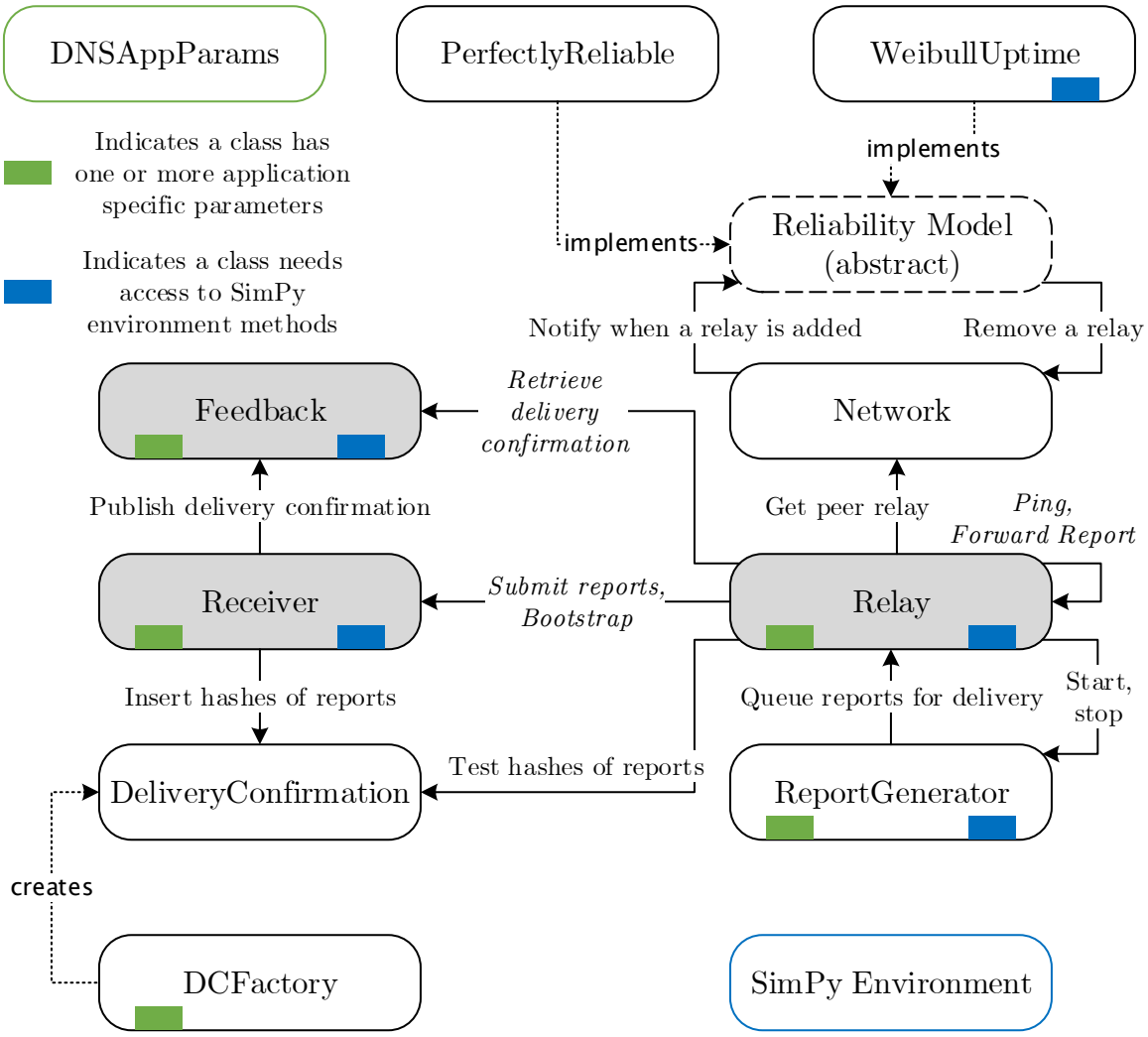

Figure 4·6: Object graph and interactions in the Python simulation

In this figure, dark-shaded classes indicate the core classes that implement most of the use-case independent logic of the proposed system. Unshaded classes represent support, utility and framework classes. Data types (`Report`, etc.) are omitted for clarity.

Connections between classes are method calls (for example, `Receiver` calls the `insert` method of `DeliveryConfirmation` to add a confirmation for a report). A method call in italic, such as *msg_submit*, simulates a message being passed on the network from one entity to another, in this case from a relay to the receiver.

`Receiver` and `Feedback` directly. This reflects the assumption that while individual relays might "come and go" from the peer-to-peer network, the relay-facing parts of the system core (report submission and feedback checking) are continuously available.

A relay in the simulation contains the following event- and time-driven processes:

Relay set maintenance: responsible for keeping the sets of known relays up to date. If the list is short, new relays would be solicited via bootstrapping from the receiver or by requesting peers from another known relay. Occasionally, a relay would be removed to induce churn in the network.

Bootstrap: this is a sub-process of relay set maintenance that interacts with the receiver to bootstrap the connection to the network. Since in the simulation the receiver never gets "too busy" (unresponsive) or unavailable, the main purpose of this sub-process at this time is to manage the bootstrapping of the earliest peers, which will not receive a valid bootstrap response until at least several other peers contact the receiver and a set time period elapses for the receiver to generate the first valid bootstrap set of active peers.

Feedback checking: responsible for obtaining delivery feedback on reports submitted by the relay. This simulation does not implement checking of forwarded reports' delivery feedback for the purpose of reputation tracking.

Pool maintenance: responsible for managing reports in the relay's mixing pool. The process sets the time-out for each report, decides on whether to forward or to submit a report, and performs the chosen action when the time-out expires. The process needs to handle cases where the peer chosen for forwarding is no longer available.

The `Relay` class introduces the convention of simulating communication between two entities in the network. Recalling that the simulation trades off the accuracy

of including encryption and wire-level messages for simulation speed, `msg_` methods (for example, `msg_report`) implement and abstract such communication. The caller (originator of the communication; in this case, a relay that is forwarding a report) calls a `msg_` method of the destination relay. All of the message contents from the caller are passed as parameters to the method; all of the reply contents are returned from the method. In current simulation, Python structures are passed directly; however, an extension that implements encoding and decoding of messages down to wire-level would be straightforward within this framework, impacting only the specific methods of the caller and the callee.

The `Network` class participates in the simulation of communication between two entities in the network. Currently, the class is only dealing with the entities whose availability on the network can be limited (relays); the communication between relays and the receiver, for example, bypasses the `Network` class. The class controls the availability by using a plug-in reliability model that can remove a relay from the network or reconnect it back, but is not responsible for the initial creation of relays. Two such models are provided: `PerfectlyReliable` is a stub implementation that does not model reliability (all entities are always connected). `WeibullUptime` gives each relay a lifetime drawn from a Weibull probability distribution; once the lifetime expires, the relay is removed from the network and deleted from the simulation. Other models can be plugged in and, since a reliability model receives the identity of each relay, it can use per-relay parameters or even real-world data to perform accurate and diverse lifetime simulations, including cases where a relay is allowed to leave and rejoin the simulation at a later time.

As a dynamic, reflection-capable language, Python allows an interesting extension to the `Network` class that is not currently implemented. By adding the name of the `msg_` method to a communication request, it would be possible to sever the current

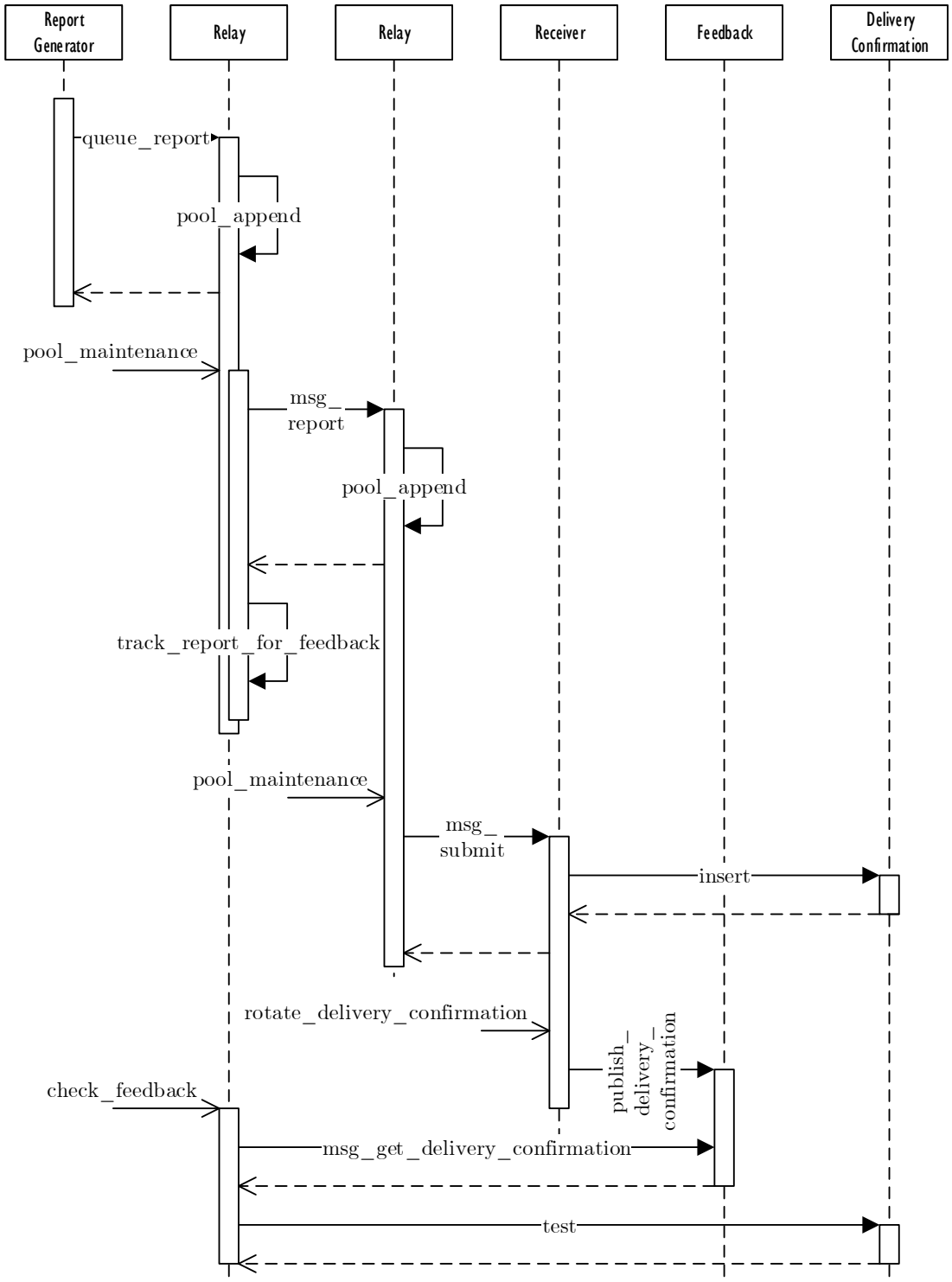

Figure 4·7: Path of a report through the Python simulation

direct connection between two communicating entities, removing the conceptual distinction between inter-relay communication and communication to the receiver and the feedback server. This would allow modeling the availability of both the receiver and the feedback server, simplify the simulation's object graph, and improve the separation between the entities. The separation would be improved since the only permitted interaction would be mediated by the `Network` class instance, instead of the current implementation where the entity initiating the communication receives a handle to the other entity and could inspect its internal state.

The `Receiver` class collects reports submitted by relays and allows relays to bootstrap their connection to the network by returning a collection of other known relays. The class is not responsible for feedback processing; this is the task performed by the `Feedback` class. The communication between the two is done by passing a `DeliveryConfirmation` instance from the receiver to the feedback server each time an epoch ends. The receiver adds each received message to the delivery confirmation instance, simulating actual processing of the message, and the feedback server responds to relay requests to retrieve feedback (delivery confirmation only, in this case) as long as the requests are within the requested epoch's feedback checking window.

Figure 4·7 shows the flow of the simulation during processing of a single report.

4.3 Modelled Reliability Simulation

In real peer-to-peer networks, nodes will join and depart the network over time. To evaluate the resulting effect on network maintenance and delivery rate (if a relay is removed from the network, all the reports waiting in its mixing pool are lost), we have added a reliability model to the simulation. Unfortunately, a quick review of the literature shows the optimal model for a relay's reliability over time is likely to be use-case-specific.

There is a body of research looking into characteristics of peer-to-peer file sharing networks. The models suggested from peer-to-peer networks would significantly underestimate uptime if applied to our system, however, due to difference in user behavior: while our system is intended to be run constantly in the background, peer-to-peer file sharing network clients are often shut down by their users soon after the requested content is downloaded. This behavior has been observed by Steiner et al. for the KAD network: "Given that KAD is predominantly used to download copyright-protected content, the users probably stay connected the least possible time required to download the requested content," [60, p.1381] and by Pouwelse et al. for BitTorrent when analyzing the uptime of peers once a download is finished: "The sharp drop ... indicates that the majority of users disconnect from the system within a few hours after the download is finished." According to the data, 83% of peers disconnect from the network within an hour of finishing a download. After 10 hours the number of peers still online decreased to 3.1%, and after 100 hours to 0.34%. [51, p.20]

Another set of previous work analyses "server-class" machines that are intended to remain functional and connected to the network for long periods of time, with occasional outages. One such system that has been analyzed extensively is PlanetLab [49], a global overlay network of machines that can be used by researchers as a testbed for distributed systems. Analyzing PlanetLab, a collection of DNS servers, and a collection of web servers, Yalagandula et al. concluded that "neither [exponential nor Pareto] distribution fits the data perfectly, but the exponential distribution fit is better." [70] However, looking at a different set of PlanetLab traces, Verespej tested the fit of data to the exponential distribution using a Chi-Square metric and found it a poor fit; even after excluding one anomalous bucket of data and aiming at a significance level of 5%, "the check failed by a large margin." [64, p.17]

Arguably the most appropriate comparison comes from several works analyzing behavior of nodes on the Skype peer-to-peer VoIP network. A study by Guha and Daswani finds that the typical usage pattern of the Skype client is similar to a background service and unlike that of a file sharing peer-to-peer client: "Skype users regularly run the client during normal working hours and close it in the evening", perhaps when the computer itself is turned off [32]. Focusing only on the network's supernodes, the authors suggested using a Weibull or Pareto distribution of session times and modelling node arrival as a variable-rate Poisson process to account for the diurnal patterns in arrival and departure of nodes. The diurnal arrival / departure regularity is much more pronounced among regular nodes on the Skype network than among supernodes, whose population is relatively stable and with a low churn rate.

The Weibull distribution surfaces again in a study by Nurmi, Brevik and Wolski examining the fit of various models of machine availability to three sets of experimental data [48]. While the authors achieve a slightly better fit with hyperexponential distributions, they note that a Weibull distribution is easier to estimate (only two parameters) and achieves good fit with two of the three data sets (certainly better than either exponential or Pareto on all of the data sets).

Lacking ground truth data for either of our use cases, we have adopted the Long data set model from Nurmi, Brevik and Wolski. It is a Weibull model with a mean lifetime of 14 days and a median of 5.3 days. In 10 iterations of a 10-day-long simulation for a 1,000-node network with immediate replacement of failed nodes, an average of 55% of 2,239 relays failed, raising the report loss probability to 1.92%.

Chapter 5

Threat Analysis

In this section, we consider the proposed system's security properties, in light of potential threats to anonymity, disruption, validity of feedback and unrelated goals.

5.1 Sender Anonymity for Report Submission

Encryption of traffic between relays means an attacker that is limited to eavesdropping has to break the underlying algorithm to be able to distinguish between two reports. An attacker can tell how many reports were received and sent, and (based on the average mixing pool size) keep a good estimate on how many reports were originated at the relay, but not to find out with certainty which ones.

Since we do not employ cover traffic, an eavesdropping attacker can see report forwarding events. Therefore, for an attacker with network-wide listening capability, it is possible to produce a list of potential senders based on back-tracking from the submitter of the report through the network. However, due to intentional and randomized delay at each hop, a classic timing attack is unlikely to succeed. The number of potential senders (the anonymity set of a report) in this case depends on the network parameters. A rough indicator can be obtained by raising the average mixing pool size by the average number of hops a report takes—for an example based on the simulation in section 4.2, there can be 160,000 potential senders given $p_{sub} = 0.33$ and observed average mixing pool size of 20 reports.

An attacker running relays gains the ability to see encrypted reports without the relay-to-relay encryption layer as they pass through relays controlled by the attacker; in this case the attacker can also assign a higher probability a report came from the relay forwarding it: p_{sub}, according to the theoretical analysis. However, the submitted data cannot be recovered, as that would require breaking the encryption algorithm used between an agent and the collector. Indistinguishability of reports based on contents and available metadata reduces the likelihood of success for the predecessor attack (as described below in Section 5.2).

An attacker compromising the core gains the ability to see the submitted data for the active epochs pre-aggregation. However, since there is neither a "return address" nor a persistent source identifier in the report metadata, and the relay submitting the report is not the originator, the attacker has to rely on properties of the report data itself. In our design, deanonymization based on accumulated data is hindered by two design choices: a preference for splitting data into small, independent reports that have smaller likelihood of uniquely identifiable contents; and aggressively discarding non-aggregated data as soon as possible. Indeed, for some use cases (such as the package popularity use case), incoming data can be processed and immediately discarded, as the generation of feedback does not require consulting a third-party analysis service. Nevertheless, as in any system that accepts and stores unmodified user data, we cannot provide a guarantee user data cannot be reassociated using external knowledge; if this kind of guarantee is required, then a differential-privacy approach for obfuscating the submitted data has to be employed, with the concomitant reduction in utility.

The improvement over many existing systems in this scenario is twofold. First, there is much less data for the attacker to obtain and analyze immediately following a successful compromise. While a conventionally designed system might contain a

historical database of responses associated with a particular user, our system only holds non-aggregated reports for several epochs at most, without an obvious user identifier for each report. Second, splitting of submitted data into small, independent reports reduces the impact of deanonymizing markers to only positively associating small pieces of data instead of all data coming from the particular user. This greatly increases the complexity of the task for the attacker: compare, for example, the difficulty of identifying a book based on a particular page with a highlighted word to the difficulty of doing the same identification after the book has been passed through a document shredder: the pieces with highlighting are removed from the surrounding context of their page, which has to be painstakingly reconstructed. The increased difficulty is associated with the increased time the attacker must spend actively collecting submitted data from the system core, raising the likelihood of detection and ejection from the system before enough data has been obtained.

Compromising the core also exposes the clients to a key-mapping attack. Since we would like to rotate the collector encryption key pair regularly to provide some measure of forward secrecy, the collector's public key cannot be hardcoded and must be periodically retrieved. However, if the collector issues a different key for every relay, it can discover the source from the key that was used to encrypt it. A similar attack has been investigated in the design of the Tarzan peer-to-peer network [31]. To prevent such an attack, an agent will also query other relays to retrieve the same information indirectly, and will only accept it as true if a majority of the responses is identical. We note that a strictly client-server model (as a natural, not peer-to-peer implementation over Tor would be) does not offer any protection against a potentially malicious server and requires an additional source of trust to detect and mitigate this attack.

Attacking the analysis services is unlikely to be helpful in breaking anonymity.

From the information-theoretic perspective, any information that can be gleaned from the analysis service can also be recovered by looking at the anonymized data feed. If the feed is not public, since source information is removed and the data is aggregated, placing a marker into one item of collected data would not help associating other pieces of data belonging to the same source.

Looking at combinations of attacks described above, the option that appears most promising to the attacker would be a combination of core compromise and running multiple cooperating relays. While such attack would no longer be passive, and would have to be maintained over time due to regular key rotation at the core, this scenario makes the predecessor attack possible. We discuss the predecessor attack in more detail below, while noting that even this combination does not deterministically reveal the source of each report; rather, according to properties of the predecessor attack, the probability of correctly identifying a source would grow slowly over time as data is accumulated by the attacker.

5.2 Peer-to-Peer Network Attacks

This section focuses on well-known attacks against peer-to-peer networks. The unifying goal of most of these attacks is to achieve an outside influence on the network compared to the number of nodes under attacker control; either locally (for example, by filling up a routing table of a particular node with malicious collaborating peers so that any communication from that node becomes available to the attacker) or globally. We specifically address the Sybil attack, the predecessor attack, and the eclipse attack.

5.2.1 Predecessor Attack

The *predecessor attack* [69] is a passive attack on the source anonymity of a recurring conversation in the network. The Crowds introduction described a version of the

attack for a single conversation and a particular path: a node on the path can assign to its predecessor along the path a probability of being the originator that is higher than the probability for any other node (since, given no side-channel information, any other node is equally likely to be the originator). However, as long as the forwarding probability is high enough and the number of attackers low enough, the previous relay is less likely to be the originator than it is to be merely a forwarder [54, Theorem 5.2]. In Crowds terms, given our assumptions on the network size and fraction of nodes controlled by the adversary, we can choose the network parameters such that the predecessor is "probably innocent" with a comfortable margin.

The predecessor attack is a generalization of the above process. If a conversation is recurring and identifiable, and paths in the network change over time, the attacker will have multiple chances to identify the originator. In particular, if the path length is fixed, then eventually a path will be formed where the attacker controlled nodes occupy all the positions except the origin; in that case, the source is positively identified. Even if the path length is potentially unlimited, Wright et al. show that the source can eventually be identified with a high certainty.

The predecessor attack is effective in Crowds and Tor because unique conversation identifiers are easy to obtain from unencrypted data: in Crowds, each jondo (relay) can see verbatim traffic that is being forwarded, and therefore can collect "login names, user IDs, web cookies, and email addresses" which can all serve as unique user identifiers [69]. In Tor, the route from the exit relay to the receiver is not encrypted, so an attacker can recover identifying data by controlling the exit relay and use a timing attack to link the conversation to the originator. Wright et al. state that when timing attacks are not possible, initiators have a substantial advantage against the attack.

In our system, there are no legacy plaintext protocols, nor ample metadata. Since

report contents are encrypted end to end, recovering any data identifying a sender from an encrypted report requires the attacker to either break the encryption or have a persistent core compromise that provides the current secret keys over the duration of the attack. However, a persistent core compromise means the attack is no longer passive, negating one of the chief advantages of the predecessor attack; and, since the submitted data is fragmented over many small reports, it is likely that the attacker will only be able to positively associate a fraction of them, therefore increasing the number of "rounds" (observed associated transmissions) and time needed for a successful attack.

The unencrypted part of the report offers no help to the attacker since each report uses a different random identifier, so linking two reports from the same source based only on the encrypted metadata should not be feasible. Thus, lacking a persistent core compromise, one of the two major assumptions of the predecessor attack does not hold: the unencrypted information available to the attacker is not helpful in uniquely identifying the sender.

The other two distinguishing factors in the attack are the receiver identity and timing. In our network, all reports have the same destination, so identification based on packet destination is not possible; timing attacks are made difficult by unidirectional nature of the traffic (from relays to the receiver), by randomization of report generation times, and by intentional delays in other relays' mixing pools.

We conclude that a predecessor attack is unlikely to be successful unless the attacker is also in possession of current decryption keys from the system core for the duration of the attack, and if report contents contain information that allows unique source identification (which might be more or less likely, depending on specifics of the use case). No other peer-to-peer network from the referenced papers requires the responder to be compromised for the attack to be successful, and for all other

referenced networks the attack can be completely passive.

5.2.2 Sybil Attack

The *sybil attack* is an attack against the assumption that a malicious entity cannot join more than a certain percentage of nodes to the network. If it is easy for an entity to create multiple distinct identities and join them to the network, it can achieve an outsized presence within the network and subsequently mount further attacks on the network's functionality or anonymity of its participants. Douceur shows that networks without a centralized certification authority, in general, cannot prevent entities from presenting multiple identities in a large-scale distributed system [25].

Our approach to the sybil attack has three components. While we do not claim to solve the general problem, we aim to make it more difficult for a low-capacity attacker to connect a large number of nodes to a single peer or to run nodes that do not perform useful work (delivery of reports) correctly. Furthermore, since the sybil attack is an enabler of other attacks, we look at other attacks that require a large number of nodes and either address them specifically (e.g. Eclipse attack) or show that they are not relevant for our network due to its intentionally limited functionality (for example, seeking specific centrally assigned node identifiers to concentrate attacker-controlled nodes in a particular area of the network).

Our first obstacle on the attacker's path is a preference of each relay to build a diverse network of peers. Specifically, we look at IP addresses of the potential candidates and prefer relays that are as far away as possible (treating the IP space range as a numerical circle of size 2^{32}) from existing relays. This ensures that, while an attacker can connect a large number of peers to the network from a single IP address or from a small range of addresses, relays will seek to have more than one such malicious peer in their active list, as long as they can establish connectivity to other benign relays and successfully request additional peers from them.

Our second check on the attacker's behavior takes form of local reputation tracking. Once a new relay A is accepted into an active list by a particular relay R, it will regularly be forwarded reports. Since delivery confirmation is computed from information available to each relay on the path, R can check whether reports forwarded to A showed up in delivery confirmations. Since R needs to check delivery confirmation for reports originated from R, no additional network traffic is caused by this check. If the report does show up, A's reputation score is increased. There is no penalty for not delivering a report, since we do not know what is the cause (and, in fact, if we did penalize non-delivery, an attacker could drop reports from particular hosts to attacker-controlled relays in order to hurt their reputation).

The reputation score is taken into account when a relay chooses which peers from its active list to return in response to a peer request from another relay. Relays with low reputation will not be selected, making it less likely that a relay that is present on the network but does not perform the core function of forwarding reports will be receiving new connections and traffic. This check can be viewed as imposing a bandwidth resource test on every network participant, but the test is computed as part of the normal functioning of the network and not as an explicit challenge or audit.

Some other approaches to addressing the Sybil attacks are [39]:

Trusted authority: The only approach that can completely prevent Sybil attacks, this method relies on having a unique entity proof that can be presented. Our survey of literature found no easy method that has properties of both global applicability and simple automatic verification of credentials; existing successful processes are either manual (such as verifying a government-provided ID before endorsing someone's public key in a PGP web of trust) or rely on scarce resources (such possession of a hardware token with a certificate). Either of these

options would conflict with our *low friction* design goal; however, in some scenarios this approach can be useful in the proposed system as well. Considering a scenario with a low number (perhaps hundreds) of predetermined participants, producing signed keys for each participant is certainly feasible if the overhead of doing so is judged to be preferable to a chance of a Sybil attack on the network.

Resource testing: This approach checks for computing, storage, network and other resources with the goal of raising the attacker's cost of mounting a successful attack. Our relays prioritize peers with heterogenous IP addresses to force an attacker to diversify its network presence. Since we want the system to be applicable to a variety of use cases, including those with low computing capacity (embedded) or battery-powered hosts, we avoid explicit computing, storage, or network bandwidth checks.

Recurring costs and fees: this variant of resource testing, if implemented, would violate the *low friction* design goal.

Trusted devices: we see trusted devices as either a version of trusted authority, if the process of acquiring those devices actually validates the one-entity-per-device constraint, or a mere resource constraint if a device is expensive. In either case, no low friction, globally available and applicable option for trusted devices currently exists.

Reputation systems: Cheng and Friedman have demonstrated that no symmetric reputation mechanism is sybil-proof [16]. Systems that would be effective against sybil attacks require either trusted peers that manage reputation, or local computation (every node builds its own reputation database from its interaction with other nodes). We adopt the latter approach and compute a local reputation score for neighboring relays, as described above.

5.2.3 Neighbor Selection (Neighbor Table Pollution) Attack

The neighbor selection attack happens when the attacker focuses its attention on a single node in a peer-to-peer network and attempts to replace all of the peers in its routing table with nodes controlled by the attacker. In the presence of a Sybil attack, neighbor table pollution becomes easier as attacker can raise many malicious nodes with ease.

Our system has three sources for new peer information for a relay: bootstrapping, peer queries by the relay (pulls), and new peers contacting the relay (pushes). Among those sources, pushes are the least trustworthy (since we have no previous reputation information on an existing peer, and selection of good peers from bootstrapping can be done with the receiver's network-wide visibility). However, having this method available is very useful for sending some reports through the newly joined peers to begin masking the reports they are generating. Bortnikov et al. make an additional point in favor of keeping at least some push-generated new relays in the active set: while a pull from a benign peer could contain some malicious entries, a push from a benign peer is always benign [11].

We can reconcile the two conflicting goals of choosing the most trusted peers and allowing new peers to join the network by remembering the manner in which we obtained a peer's contact information, and limiting the number of new peers to a fraction of the active set. Once a new peer has been present in the active set for a while and its reputation has accumulated enough, it can be included in responses to peer queries from other peers, but its status within the peer it originally contacted will not change for as long as it remains within that peer's active set. A relay will also never send a peer query to a peer that contacted it first, preventing the malicious peer from returning a list full of malicious peers in a response to such a query.

In combination with the selection preference of building a diverse network de-

scribed above, this policy should make it significantly more difficult for an adversary to completely take over a particular peer's active set. However, due to our requirements for induced churn and preference for fluctuation in active set membership for every relay, we think it would be difficult to eliminate the possibility of a successful attack by long-lived, well-behaved peers controlled by the attacker completely.

5.2.4 Eclipse Attack

The eclipse attack is, in effect, a neighbor selection attack aimed simultaneously at each node in the network, with the goal of filling all of the nodes' neighbor tables with peers controlled by the attacker and in this way separate ("eclipse") the correct nodes from each other. The outcome of a successful eclipse attack is a partitioned network where the attacker controls the traffic between the parts and has a global, mostly complete view of the network's communications [58]. Singh et al. show that in presence of a Sybil attack, a small number of entities can mount an eclipse attack, and that the attack is possible even in presence of an effective defense to Sybil attack.

Singh et al. propose performing distributed, anonymized audits to verify each node's in-degree (number of peers that route traffic to the node), on the assumption that a small number of cooperating peers attempting to eclipse a large network would have a much higher number of active connections than average. We will use a similar idea to reduce the chance of a successful eclipse attack in our system. Unlike the proposal described by Singh et al., our audit consists of a two-step anonymous interaction with the audited node, with the goal of avoiding a network scanning/enumeration information disclosure that would be possible otherwise.

5.2.5 Limited Functionality Obviates Other Known Attacks

Since our network has a very limited, application-specific functionality compared to general purpose peer-to-peer network, some known security issues are not applica-

ble to the proposed design. For example, a survey of peer-to-peer security issues by Dan Wallach lists several secure primitives that are required for implementation of a general-purpose DHT (Distributed Hash Table)-based peer-to-peer network: secure and random node identifier assignment, secure routing table maintenance, and secure message forwarding [65]. These primitives are derived from the DHT's basic architecture, where information about particular data items would be stored at a deterministic, small set of nodes and global routing capability is required so that any member of the network could reach one or more nodes from the set to obtain either the data or specific instructions for a particular data item of interest.

However, our network does not store retrievable data, does not operate a DHT, and does not require a global routing capability to perform its functions. This simplifies the design and obviates some of the known peer-to-peer security issues. In particular, we do not require securely generated random node identifiers since we do not offer any functionality that depends on a specific node's key value (such as storage of metadata about items of interest whose hash is close to the node's identifier, as a traditional DHT would do). While we are concerned with secure routing table maintenance (which, in Wallach's paper, means preventing a neighbor selection attack), we do not require a secure message forwarding primitive (that carries a message to a particular node or a set of nodes with high probability in presence of adversaries) since our network does not need and does not offer routing of messages to particular nodes addressed only by their node identifiers.

This limited functionality leads to simple network algorithms and compact code. Constrained routing is often used to offer secure message forwarding, but if secure message forwarding is not necessary we can use a simpler random routing algorithm.

5.3 Receiver Anonymity for Feedback

In the simple model, the feedback content is identical and broadcast to many clients. In particular, the delivery confirmation and a binary feedback would be downloaded by any relay that has submitted a report in a given epoch; if reputation tracking is used, the delivery confirmation is needed even if the relay has only forwarded reports during the epoch and has not originated any reports. Therefore, neither the core or an adversary gain any information (besides the fact the relay has forwarded at least one report) from feedback access patterns of relays.

In a scenario where there are multiple levels of feedback (for example, binary feedback indicating an agent should fetch and examine the detailed free-form feedback for their report), the fact of a relay downloading the free-form feedback identifies a relay as a sender of a report for which free-form feedback has been generated. This should not be a concern in a honest system given our assumption that it is difficult to track a particular relay over time by associating individual reports, but if an attacker takes over the core they could arbitrarily change the feedback. For example, by directing only the source of a particular report to download a particular feedback layer, an attacker could deanonymize that report's source. To prevent this, we can make a chosen percentage of relays to download an optional feedback layer even if they do not need it otherwise, to create some "masking traffic" for this possibility.

For PIR models, the chosen PIR protocol should provide receiver anonymity.

5.3.1 Partial Queries and Better Bloom Filter Privacy

Bloom filters, by themselves, do not provide privacy: with an optimally filled Bloom filter, there is a significant probability that a given set bit is associated with only one item. In this case, there is a high likelihood that a query for that bit would be coming either from the originator of that item or from relays that have forwarded the

corresponding report. Combined with timing information, the path of a report could be reconstructed completely in this case.

We have performed several experiments with queries that include random bits (not corresponding to any items actually needed), but results were unsatisfactory. Adding small amounts of random bits to the query do not impact the detection likelihood significantly, and large amounts of random bits (on the order of 20-30% of the size of the filter) make the query itself and the response to it scale linearly with the size of the filter. This does not make a significant difference in mitigating the bandwidth cost of a privacy-sensitive query.

An interesting approach, whose application we relegate to future work, has been proposed by Bianchi et al [7]. The authors observe that the optimal calculation of Bloom filter parameters results in a rather small anonymity set, and if larger anonymity set is the goal then the filter parameters might need to be adjusted accordingly. In addition, instead of simply reducing the filter size or setting additional random bits to increase the anonymity set, authors propose to strategically select specific bits in the filter such that the contents of the filter cover as many possible values as possible from within a set universe. A strategic choice of bits to include in a query, as well as splitting a query into multiple independent parts, might allow us to do a partial query instead of downloading a complete filter. Nevertheless, it is certain that the privacy guarantee will be degraded with this approach, and we will be trading more comprehensive privacy for conservation of network resources.

5.4 Disruption

Tampering by the network will lead to the modified report being dropped by the receiving relay, since forging the correct signature without knowing the relay's key should be unfeasible based on the strength of the underlying cryptographic algorithm.

Tampering by a compromised relay will lead to the modified report being dropped at the collector.

Since each relay makes independent routing choices, a DoS attack aimed at any individual relay has a limited impact in the proposed design. For Tor, for example, an attack on a single relay could destroy tunnels belonging to hundreds of users. In our design, if a relay that is being attacked is chosen as the next hop, an availability probe sent before forwarding would not be returned, and the sender would choose a different relay without losing data. We note that, in general, a probe cannot be successfully spoofed by an adversary because of encryption of the communication channel between any two relays; however, an active adversary controlling the channel between a relay and the Internet can either deny service to a relay or perform a man-in-the-middle attack.

A denial-of-service attack on the collector is more problematic. If the attacker submits reports to different relays in the network, they will get forwarded and eventually submitted. Since the IP address of the submitter does not reveal the identity of the attacker, traditional ways to mitigate a DoS at the collector by filtering out some traffic sources or packet patterns would be less effective. In addition to causing load on the collector, this attack degrades the performance of the peer-to-peer network, since many relays' queues get filled to capacity and they start refusing requests to forward reports. In our simulations, it was possible to cause a virtual gridlock on the peer-to-peer network in this way. To resolve the gridlock, we specify a limit on how many times a relay can attempt to forward a received report before discarding it.

When a simple feedback mechanism is used, it is possible to mitigate a DoS attack on the feedback source by allowing relays to redistribute the feedback file. Since the feedback is signed, making an undetectable modification to the feedback should not be possible, and as long as several clients have managed to successfully download the

feedback it should spread within the network even under DoS. A single-server PIR algorithm appears to be very susceptible to DoS. Existing algorithms are very computationally intensive, so an adversary could execute a successful distributed attack simply by submitting numerous feedback queries. A multiple-server PIR algorithm is likely to be a better alternative. The number of cooperating servers can be large and if only a few of them are chosen by each client service, they will not be impacted significantly if certain servers become unavailable due to an attack.

5.5 Validity of Feedback

An impersonation attack (an attacker masquerading as a different agent) does not seem feasible in the proposed design. A user encrypts submitted reports using public key encryption and locally generated keys. The public key serves as a pseudonymous and temporary "user identity". An attacker attempting to generate a valid report using the same (freely observable) public key would have to forge the corresponding cryptographic signature, which should not be possible since the chosen cryptographic algorithm is designed to meet the notion of third-party unforgeability.

Similarly, tampering with the feedback data in the simple model of broadcast feedback should not be possible. Since feedback data is signed with public key signatures, a successful attacker would have to break the underlying cryptographic algorithm and forge a signature, which is conjectured to be unfeasible. However, if an attacker manages to obtain the encryption key of the feedback server, they could generate invalid data and clients would accept it.

The effect of submitting fake but plausible and correctly formatted data into the system is use case dependent. In particular, any kind of voting or counting can be easily manipulated under the proposed system, since a system with short-lived anonymous identity of reporters and without a centralized trust authority is quite

susceptible to sybil attacks. Other types of processing are better suited to our design. For example, classification according to predetermined criteria is not affected by fake data, beyond the resources needed to process additional inputs. Learning classifiers can potentially be affected, especially if the disruptor knows the relevant features and creates fake data accordingly. Disruption using random fake data could be less successful since it might be dismissed as noise by the learning process.

Our approach to mitigating this threat is to combine multiple analysis systems whenever possible. This is facilitated by making the anonymized, aggregated data feed public and having a formalized feedback API for each application, so that it would be possible to integrate multiple analysis systems easily. This would make the task significantly harder for an attacker, who would now have to craft the inputs to mislead multiple different analysis systems at once.

5.6 Unrelated Goals

One of the distinguishing features of the proposed design is that a peer-to-peer network relay will not forward reports that do not have a valid collector as the destination, by checking the delivery header of each incoming report. For a proof-of-concept implementation, this can be implemented as a fixed list of approved destinations, inhibiting the generation of traffic to an arbitrary address via the proposed service.

It is, however, possible to misuse the proposed network to communicate arbitrary data—with significant effort. The process would require the receiver of the data to run a non-trivial percentage of modified relays on the network. The sender would send a large number of reports that contain a message encrypted with public key of the adversary instead of the collector's. The collector would discard such reports. However, as reports are forwarded randomly, some of them will be forwarded via

relays owned by the receiving adversary, who is able to decode and recover the hidden message.

Given the effort and the collusion required, we do not see this form of communication as attractive to most kinds of attackers since, on an average host, there are much easier ways to send information out and highly secure networks tend to disable any services that are not critical to their functionality, including diagnostics.

Chapter 6

Use Cases

This chapter details three sample use cases where the proposed design can be applied. We list the properties of data collected for each use case, explain the contents of the feedback and the utility of the system to a user, and highlight particular challenges associated with each scenario.

6.1 Malware Detection from DNS Traffic

This use case applies the proposed framework to collect DNS traffic from individual hosts and pass the aggregated, anonymized data to be analysed by third-party services. The feedback, and the benefit to user, contains the likelihood the host is being infected by malware or used for malicious activity.

6.1.1 Motivation

The DNS protocol is used by almost all Internet-connected devices to turn human-readable addresses (e.g. "www.google.com") to IP addresses of corresponding servers. Since this is done for any website visited and for many of the services running on a computer, a history of DNS queries tied to a person's identity forms a very rich and personal collection of data.

Most software, including malware, utilizes DNS in order to gain the flexibility of associating a fixed domain name to a number of servers, which may change over time. The importance of the protocol, combined with its relative insecurity (most of the

global DNS traffic is in cleartext, not encrypted and not signed, and uses an easily spoofable UDP protocol), leads to it being an attractive target for abuse.

It is easy to think of several useful ways to analyze DNS data for patterns of potentially malicious activity. Merely by comparing a list of queries to a list of domains known to host malware or to be used for botnet orchestration, it is possible to estimate a likelihood of a host being infected. For a more interesting example, by comparing a query response for a particular domain to responses received by others, it is possible to detect DNS poisoning attempts or tampering with DNS responses en route. Notably, since these malicious activities seldom originate from legitimate DNS servers, looking at the host's designated DNS server's traffic (as some analysis systems do) will not expose this behavior.

Several systems exist that scan DNS queries on the global level to identify known and emerging threats, but the data collected by those systems is proprietary and the analysis results are usually available only to ISPs and enterprises [1]. EXPOSURE [9] is one analysis system that uses machine learning techniques to classify DNS traffic and detect domain names that are likely to be involved in malicious activity.

Typically, EXPOSURE and similar systems that require near-realtime, large scale streams of data use either a proprietary service or establish direct partnerships with commercial entities that stipulate that the dataset is to remain private: for example, the EXPOSURE team has analyzed DNS data collected from an ISP with around 30,000 clients. [9, p. 11], and used the Farsight SIE (Security Information Exchange) feed, a close-to-realtime DNS dataset. The proprietary feed generates 300 Mbps of raw data[56] contributed by "vetted organizations"[57] and is used in some current research. However, a 2014 update mentions neither of these sources is currently being used by EXPOSURE due to "data access problems" [8].

Public anonymized datasets are very useful in comparative evaluation of different

detection systems. However, there are very limited public dataset options for DNS query data[1]. Indeed, the paucity of large public datasets for network intrusion detection systems has been called "arguably the most significant challenge an evaluation faces" [59].

Our use case of using the proposed system to collect DNS queries for analysis has the potential to extend this detection capability down to individual host while retaining the anonymity and privacy properties of the design.

6.1.2 Design Considerations

DNS requests and responses are fairly small, with a typical mean size below 200 bytes. Looking at a single personal computer, we can usually observe periods of low activity (few queries) when the machine is not being used, and high activity when a user is present. When a complex website is loaded, several dozens of queries can be sent within a few seconds. From the global network perspective, however, the volume of DNS traffic is more consistent over time.

For this use case, we would collect DNS response packets, stripping the IP and UDP layers. Individual packets are collected into reports of approximately 2000 bytes in size, containing around ten packets each. The aggregation process produces a feed that maps each domain name to the count of its responses over the epoch: for example, "There were six queries for the domain `example.com` that received two answers with IP address 1.2.3.4 and five answers with IP address 5.6.7.8". The counts of addresses do not sum up to the total number of queries because a single query can return multiple results. It is easy to extend the feed to provide other features, such as ranges of time-to-live (TTL) values seen in the data, and to improve anonymity at the source by analyzing the DNS response and stripping all parts that are not

[1]There are several public historical datasets, notably yearly DNS-OARC (Domain Name System Operations Analysis and Research Center) "Day In The Life of the Internet" recordings of contiguous 48- to 72-hour periods [23], but no realtime feeds as far as we know.

relevant to the features collected by the system, instead of sending the DNS packet in its entirety, including fields that are not relevant for the analysis (for example, the DNS transaction identifier).

The analysis system would receive the feed and return a set of domains it considers problematic, including a "score" for each detection. Several analysis systems of this nature exist: EXPOSURE has been mentioned earlier; another such system is Notos [4]. Typically, these systems analyze raw DNS feeds but, since the first step is feature extraction, it should be easy to adapt them to receive a feed our system would produce.

Multiple analysis systems can be aggregated when calculating the feedback. The feedback provided to the users combines delivery confirmation, a binary feedback indicating whether a malicious domain has been identified in a report, and a free form feedback containing the flagged domain name and the detection score. On receiving the feedback, the agent can look at the past history of detections and initiate user action if the frequency or confidence of detections exceeds a predefined threshold.

This use case presents unique privacy and anonymity challenges since the majority of DNS traffic is neither encrypted nor signed (DNSSEC adoption was at less than 20% at the time this paper was written) and is therefore visible and modifiable by an attacker residing on the network. Potential mitigations can include using encryption between resolvers (e.g. DNSCurve), validating DNSSEC at the recursive resolver to prevent tampering, and further preprocessing at the agent to make individual samples more generic.

To make a traffic correlation attack (observing DNS queries coming from the host and comparing them to traffic associated to our system on the host) more difficult, we could maintain a buffer of collected DNS queries at the agent and, instead of sending a report as soon as it fills up, generate reports at random intervals using data from

the buffer. The distribution of the intervals can be adjusted over time in response to behavior of the buffer, and some "dummy reports" can be generated when there is little DNS activity of the host.

6.2 Linux Package Popularity

This use case applies the proposed system to collecting installed software information from Linux-based hosts and derives statistics on individual package popularity, speed of adoption of new versions, architecture use trends and so on. The value to user, contained in the system's feedback, would be notification about missing critical software updates.

6.2.1 Motivation

Gauging the popularity of open-source software is not easy. Download counters are easy to implement but do not track use over time, and package-specific diagnostic tools that "call home" are not a generic solution. Debian and derived distributions contain the `popularity-contest` package [50], but it follows the common "trusted core" design and until recently submitted data in cleartext; currently, encryption is optional. Because the data was submitted in cleartext, useful information (such as installed packages' versions) was not collected, presumably since it was considered too sensitive to send.

Many other distributions, in particular embedded ones like OpenWRT, do not have any public tools that show which packages are being used the most or attempt to analyze the popularity of different architectures and hardware platforms. A new user wishing to choose a popular, well supported package from a bewildering array of options offered by the distribution often has to rely on forums, wiki pages (that are potentially well out of date), mailing lists and other opinionated but low on hard data sources of information.

In addition to offering a way to track and aggregate this data anonymously, we imagine a possibility of allowing comparison and ranking not only within a single distribution or a distribution family, but within the wider open-source ecosystem (by enabling association between a distribution package and its upstream origin). Our value proposition to users for this use case is to collect announcements of security issues and notify users if they are running older versions with known security problems.

Most modern distributions already incorporate the functionality of notifying the user of available package updates (indeed, some of the distributions enable this functionality by default). Nevertheless, by looking at upstream data as well as distribution data, we could offer a unique benefit of publishing such notifications once they are posted by the upstream, without waiting for the fix to be incorporated and released by the distribution; it is possible that such a release is delayed, or that the version of the distribution that the user has is no longer supported even for security issues, and in those cases the user could remain in the dark considerably longer unless they proactively track vulnerability disclosures.

In either case, there is significant additional value is in organized collection and analysis of the data itself, and the improved decision making by users and maintainers alike that derives from easy availability of the analysis.

6.2.2 Design Considerations

The collection method for this use case was discussed in Section 3.5. An agent would capture a snapshot of installed packages and versions regularly (for example, daily), split it into smaller reports of several packages at a time, and submit those reports over time at random times until the next reporting period. This strategy achieves the two desirable features of small report sizes and predictable, regular generation of reports.

The feedback in this case could be binary, marking a report as containing a known vulnerable package. Since Linux distributions generally have a simple way to query the repository for package updates, the tradeoff of pointing out a particular package within a small report with the free-form feedback does not appear to be worth the cost of downloading a larger feedback file.

Since delivery confirmation is being provided as well, it is possible to significantly reduce the impact of lost reports. If a delivery confirmation shows a report as lost, the packages contained within the report can be returned to the list of package data to send, and the lost data will be eventually retransmitted. In this way, the chance of losing reports irrecoverably becomes a chance the delivery confirmation's Bloom filter incorrectly shows an undelivered report as delivered. In our simulation, the difference between that probability and the probability a report has been dropped was found to be about two orders of magnitude.

In a full implementation of this use case, we would also like to address two known weaknesses of the `popularity-contest` package [50]: reporting unofficial/local packages and reporting packages that are very lightly used. For the former, we will compare the system's package list with the distribution's package list, and only report packages that are found in both, since a package with a name that is not found in the distribution's package list must be unofficial. For lightly used packages (where, perhaps, there is only one user—the maintainer—and it might be possible to infer the maintainer is not home if the package stops showing up in the results), we would include the ability to filter packages from the submission and only provide a range at the low end of the usage axis (e.g. instead of "2 users of the package" our analysis would show "fewer than 10 users").

This use case has a particular weakness for a scenario of a core compromise combined with an attacker running additional relays and being able to decrypt reports.

For each particular Linux distribution, versions and names of packages at a given point of time are different and characteristic. For example, Debian Linux package versioning strategy combines the upstream package version with Debian package version; therefore, when looking at a particular version string, it is possible to guess which distribution the package belongs to.

In our prototype implementation, we decided to include the distribution name and version with each report to disambiguate any potential package name conflicts– occassionally, packages in different distributions can end up having the same name but corresponding to unrelated upstream software, and counting them together would be erroneous[2]. Nevertheless, because of the particular versioning conventions, it would likely be possible to deduce the Linux distribution identity with high confidence solely relying on version strings, and (with lower confidence) potentially even if the versions are normalized to specific corresponding upstream versions. The latter would be a possibility since stable releases often follow the model of selecting a particular stable version and remaining on it for the life of the release, with updates containing only backported bug and security fixes.

The result of this information disclosure, in the scenario being discussed, that an attacker would be able to figure out the Linux distribution used by a particular relay by obtaining or predicting the Linux distribution identity using one of the methods described above and comparing the relative frequencies of distributions seen in reports coming from the relay to network-wide statistical data.

[2]For example, the `docker` package refers to Docker the lightweight containerization engine in Fedora, but to a system tray application for KDE 3 or Gnome graphical desktop environments in Ubuntu.

6.3 Anonymous Traffic Information

This use case applies the proposed system to collecting GPS location from mobile devices that are being used for GPS navigation, providing an aggregated an anonymized output feed to a system that predicts traffic levels and speeds. There is no immediate feedback provided to users, however by using such a system instead of an existing commercial solution users can be more certain their private location information is not being associated with their identity and used for purposes unrelated to navigation.

6.3.1 Motivation

It is difficult to imagine a modern world without GPS navigation and traffic prediction. The navigation software market is dominated by a small number of big players, some of which explicitly state that data collected while using their navigation apps may be retained and used for other purposes [66]. Open mapping projects such as OpenStreetMap present an alternative by providing free and open source basemap data; yet, to compete effectively with the major players, their route planning engines need a source of high quality, close-to-realtime traffic information.

This use case would regularly collect location, heading and speed samples. Several samples could be aggregated to reduce the number of reports. The receiver's aggregation process for this use case would combine the received information to assign speed/traffic density scores to particular road sections of the base map.

6.3.2 Design Considerations

One challenge of this use case is the deanonymization potential in the core compromise scenario based on the reported data, particularly in locales where there are few users of the system and overall traffic is light. Reducing the granularity of reporting to a particular road segment of the map and epoch instead of the precise location and time

could help increase the anonymity set of individual samples that could be correlated with out-of-band information (for example, a time-tagged traffic camera feed with license plate recognition software), and absence of persistent identifiers should make associating collected data points with individual users more difficult [44].

Another challenge of this use case is a potential for malicious use: by sending invalid data (for example, by adding multiple reports of slow speeds on a particular road), an attacker can modify the generated predictions and affect routes of the system's other users. An attack of this kind was reportedly carried out by residents angry at Waze mapping software for routing traffic over their quiet side streets; however, Waze detects fake reports by asking other users to confirm them (and removing unconfirmed events) and by applying reputation tracking: "...reports are given more weight if they are sent in by experienced users, as opposed to an irate homeowner who just installed the app" [47].

Since our system does not have a persistent identifier, it does not easily permit long term reputation tracking that is possible when each user is identifiable. We can attempt to detect fake data (for example, if half the reports for the same segment of road and direction of travel show a speed of 60 mph and the other half shows 5 mph, something is clearly amiss), but it might be difficult to determine the correct information.

It might also be possible to leverage a third party, such as the mobile device's operator (who always knows, within a certain level of precision, where a given device is located according to its base station association), to independently vouch for a device's location: by including both a precise location from device and a rough location provided and signed by the operator, our service could verify that the two match without learning the user's identity, and discard or deprioritize reports where this information does not correlate.

Chapter 7

Prototype Implementation

The prototype implementation of the system builds on the components developed for the message-level simulator and adds other parts that are necessary to complete the implementation for a particular use case. We add serialization and deserialization of main protocol messages into defined binary message formats and implement them as a class hierarchy, and develop a module to match request/response pairs over a connectionless protocol and to interact with the network stack to send and receive protocol messages over TCP/IP. Furthermore, we integrate a cryptographic library for end-to-end encryption of reports and implement use case specific functionality to generate data at the agent and aggregate it at the receiver. Altogether, the prototype implements the basic functionality of the proposed system for the anonymous collection of Linux package information. This specific use case was chosen because it is conceptually simple and does not require integration with any third party analysis software.

7.1 Architecture

The implementation is guided by two goals: easy portability between simulation and implementation, so that protocol and particularly behavior changes could be tested in simulation and integrated quickly into the implementation; and a clear dividing line between general functionality and implementation that would be common across use cases, and particular specializations (report contents, use-case specific processing

by the receiver, application specific parameters) that would be different for each use case. Having this division clearly marked would make it much easier for the prototype implementation to serve as a blueprint for implementations of the proposed system by other use cases.

The prototype implementation is based on an event loop architecture, similar to the simulation. An event loop architecture uses asynchronous input/output processing to handle multiple independent tasks concurrently without resorting to multithreaded or multiprocess implementation. It is a natural fit for the proposed system since most components can be decomposed into a collection of tasks triggered either periodically (for example, relay set maintenance); randomly (such as report generation); or in response to an external event (for example, arrival of a UDP packet that requires decoding and processing).

As outlined in Section 4.2.1, the APIs of the chosen simulation framework (SimPy) and the implementation framework (Python 3.4's `asyncio` library) are similar but different enough that a separate adapter for each would be required to preserve the desirable source level compatibility between both. Such an adapter is not currently included, but we strove to keep the two codebases as close as possible.

The major difference between the two implementations occurs around communication with other entities (relay to relay, relay to collector, etc.) To recall, the simulation performs such communication synchronously and returns to the caller with the response. Such implementation is not possible in the prototype, as such blocking would stall the event loop and all other tasks until a response has been received or the transaction has timed out. A "blocking"-looking style of coding can be preserved with asynchronous input/output, since a task that initiated an asynchronous I/O operation can be suspended until a response is received, while in the meanwhile control returns to other tasks sharing the same event loop. To adopt this

implementation style, some of the logic had to be changed to convert procedural calls into independent, asynchronous tasks that can be easily suspended in this manner.

The overall object graph for the prototype implementation can be seen in Figure 7·1. Comparing it to the graph of the simulation in Figure 4·6, we can observe that the main entities and classes remained the same. To make simulation compatibility easier, `Relay`'s interaction model with other entities (other Relays, Receiver, and Feedback) via method calls has been preserved, but adapters (proxy classes) have been implemented for each one of those entities. The proxies convert method calls to asynchronous requests over the network, and process the responses into a format expected by the `Relay` (same as the simulation).

New, use case specific components have been added: for example, two ways to scan the system for packages (using `dpkg` for Debian-based distributions, and by reading a text file containing package descriptions for development and simulation) and two ways to output the collected data from the receiver–a simple screen dump and aggregation to JSON).

Several base classes have been subclassed to provide use case specific functionality. For example, while the `Report` class deals with opaque binary data as report contents, the `ApcReport` subclass contains a structured list of software package information; `ApcReportGenerator` maintains a list of package information to send and uses it to produce reports from time to time; and so on.

The implementation offers a clear separation betwen the forward (report submission) and the reverse (feedback) communication directions. The forward direction uses the peer-to-peer protocol over UDP using a pair of abstraction classes, `ApcUdpProtocol` and `ApcServerUdpProtocol`, while the reverse direction downloads feedback using standard HTTP over TCP using a simple HTTP server and client implementations included in the `aiohttp` asynchronous network communication library.

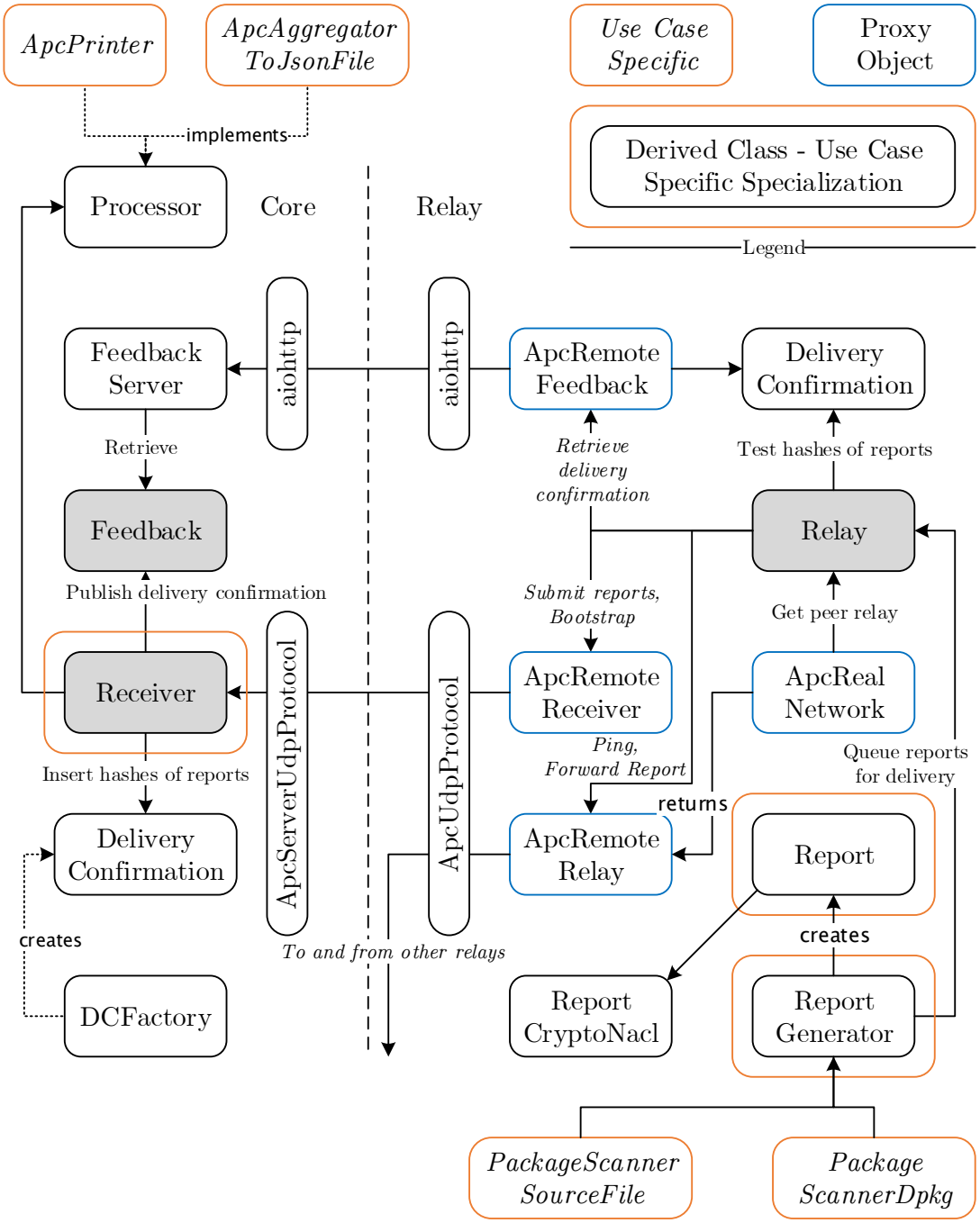

Figure 7·1: Object graph and interactions in the prototype implementation

7.2 Protocol Messages

We define ten types of messages and implement eight of them. Additional messages will be needed for advanced functionality (for example, audits of peer connectivity that aim to detect and prevent eclipse attacks). The messages are described below. The exact binary format of the messages can be seen in the code and is omitted here for brevity.

"Get Key": this message and its reply are used when communication between a pair of relays is encrypted. The "Get Key" message would be the only unencrypted message exchanged between relays, and provides the peer with a public key of the originator. The response contains the public key of the peer and allows further messages to be encrypted.

Ping: this message is used to verify connectivity between two relays. The response also conveys whether the responding peer is currently accepting reports into its mix queue. A negative response might mean that the responder's queue is close to full, and a relay should refrain from forwarding any reports in this case.

Bootstrap: this message currently has a dual purpose: it can be sent to the receiver to bootstrap a connection to the peer-to-peer network, and it can be sent to a relay to request a number of additional peers to strengthen and diversify connectivity. A response contains a list of peer information: IP address and port. The message could be extended to contain public keys as well.

Report: this message contains a report to be forwarded. The report may be encrypted. A reply message indicates whether the report has been accepted by the peer; if the peer does not reply, or replies negatively, the report would be forwarded to another peer instead.

Network Settings: this message retrieves the network settings (whether encryption is enabled, what is the public key to use for encrypting reports, and additional use-case specific parameters in the future) from the receiver. The response will contain the necessary settings.

We implemented report encryption as described in the Design chapter. The encryption uses the default public key authenticated encryption implementation of `libsodium`, and adds 73 bytes of overhead (algorithm marker, a 32-byte public key, and a 40-byte internal opaque block) to a 414-byte report contents and a 17-byte report header, causing a total of 17% increase in size compared to a plaintext report.

7.3 Message Serialization

We have considered the usage of existing serialization libraries, such as Google Protocol Buffers [52]. Since our usage scenarios are estimated to be relatively low bandwidth, small savings gained by eliminating any metadata and using a hard-coded, predefined binary format are not significant. Nevertheless, we have decided to use a manual, hardcoded serialization in this implementation, with the main reason being the excessive flexibility of Protocol Buffers and other similar formats.

Specifically, fields can be freely reordered within the message; fields that have default values can be omitted from the serialization by protocol; additional fields might be present in the message and silently discarded, and so on. This introduces a layer of abstraction and automatic efficiency that is very beneficial for ordinary applications; however, for security-oriented purposes, such implicit transformation can have unintended consequences. For example, it allows an attacker to convey data or mark messages by reordering fields, adding fields of the same type, adding unrecognized fields (that would be simply serialized back as-is by Google Protocol Buffers version 2) and so on. To avoid this risk, and due to a small number and low

complexity of our messages, we specify the format exactly and manually perform the conversion to and from binary format. Any protocol versioning would be negotiated during the initial communication between peers.

For each message (and particular data structures that form parts of that message), the `to_wire` method implements the serialization from Python into a binary string, and the `from_wire` class method implements deserialization from a binary string and returns an object of the type on success, or raises an exception on error. Any parsing exception would result in a message being dropped by the receiver.

7.4 Request/Response Tracking

The simulation offered a trivial way to connect a request and its response together: they were parts of the same synchronous invocation. This is not an option for the prototype implementation, which receives all incoming traffic on the same network port and needs to distinguish between a response to a request sent earlier and a new incoming request. We offer two different ways to perform tracking and association of messages to conversations.

The first way is used for development and debugging in case the encryption is turned off. As part of any message's header, we have a "request identifier" field. The field is filled randomly by the originator of the request, and must be identical in the response. If the value of the request identifier is different, then an incoming message is part of a new conversation.

The second way will be used for encrypted communication. When encryption is used, each packet requires a "nonce"– a unique identifier that should not be repeated. Borrowing from the design of DNSCurve [24], we divide the nonce into two equal parts. The nonce for the request contains of the first part generated by the sender, and the second part filled with zeroes. The nonce for the response contains of the

same first part (copied from the request), and the second part generated by the receiver of the request. In the encrypted communication mode, the specifics of nonce generation algorithm can also be used to prevent replay attacks of previously recorded communication.

7.5 Data Aggregation and Anonymization

Since this use case does not require processing by third parties before being able to generate feedback, we are able to avoid storing received reports altogether. When a report is received, it is decrypted and parsed into a set of packages. At this point, we can consult a list of vulnerable packages and versions and determine the feedback; currently, the feedback part of the functionality is not implemented. Once the feedback calculation is complete, we can aggregate the data with other reports.

The current code produces an aggregated output file (in JSON format) every epoch. However, given that the default reporting period for each agent is one day, there is no particular benefit to allowing access to individual epoch files; the per-epoch rationale is to lose the least amount of data if the receiver encounters an issue, and to receive some output quickly when debugging. Eventually, we anticipate combining a day's worth of data into a final source data file that would be publicly available and would form the basis of further analysis and visualization.

In addition to aggregation, further processing would be done by removing rarely occurring entries (any versions that were reported fewer times than a predetermined threshold would be counted to the package, but will not be individually reported), and by rounding up lightly-used packages (instead of counting a single-digit number of users, we would display "fewer than 10" users, and source files for those packages would contain 10 users). Both of those transformations aim to remove the ability to track behavior of individual users who happen to be using a rare package or version,

and can reveal their presence on or absence from the Internet by presence of their output within the public data file. This concern has been raised in the original Debian `popularity-contest` as well.

Chapter 8

Conclusion

Security and privacy protections for the processes of collection and storage of background, diagnostic or application telemetry data are often not a high priority for vendors and application designers, with a resulting popularity of simple "trusted core" architectures for such systems and practices that make the data easy to store and analyze but retain a lot of potentially privacy sensitive information that could be exposed if the system is compromised.

In this thesis, we have proposed an alternative architecture that moves many of the responsibilities, such as source anonymization and long-term tracking of feedback, away from the system core and bakes privacy and encryption into all elements of the design from the outset, thereby providing measurable security and privacy properties.

Our design includes a new transport layer that leverages time-tested ideas of mix networks and random path forwarding, as well as newer developments such as practical private information retrieval. By specializing our design to a particular class of applications, we were able to utilize distinctive features of background diagnostic traffic—asymmetric bandwidth requirements, high tolerance to delays, some tolerance to data loss—to choose different algorithms that offer more resistance to some types of attacks compared to general-purpose anonymizing networks such as Tor and Crowds.

Our simulations show that our architecture is viable for networks of tens of thousands of hosts, and several use cases provided in this work outline the process of adapting the framework to particular applications. In addition, we provide a proto-

type implementation that can be used for further evaluation and study of the proposed design, as well as serve as a reference for potential adoption.

8.1 Future Work

We see three main directions for future work. A long-term utilization of the proposed design for a real-world use case would allow us to iron out any remaining algorithmic issues, collect valuable metadata for future research (for example, actual availability data of typical network's users), and provide a proof of viability to potential adopters. Extensions of our design, for example adding optional proofs of possession of a valid unique key similar to the scheme in Anonize [34], would improve its suitability for specific use cases related to verifiable counting. Finally, a non-technical study could be undertaken to understand the impact that a secure underlying technology, such as our architecture, can have on users' decisions of volunteering potentially sensitive background data about their surroundings, interactions with the environment, and patterns of application usage—both for their own benefit and for scientific research.

Bibliography

[1] *Advanced Malware Protection*. OpenDNS. URL: https://www.opendns.com/enterprise-security/solutions/advanced-malware-protection/.

[2] Istemi Ekin Akkus et al. "Non-tracking web analytics". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM. 2012, pp. 687–698.

[3] Paulo Sérgio Almeida et al. "Scalable bloom filters". In: *Information Processing Letters* 101.6 (2007), pp. 255–261.

[4] Manos Antonakakis et al. "Building a Dynamic Reputation System for DNS". In: *USENIX security symposium*. 2010, pp. 273–290.

[5] Kevin Bauer et al. "Low-resource routing attacks against Tor". In: *Proceedings of the 2007 ACM workshop on Privacy in electronic society*. ACM. 2007, pp. 11–20.

[6] Alastair R Beresford and Frank Stajano. "Location privacy in pervasive computing". In: *IEEE Pervasive computing* 2.1 (2003), pp. 46–55.

[7] Giuseppe Bianchi, Lorenzo Bracciale, and Pierpaolo Loreti. ""Better Than Nothing" Privacy with Bloom Filters: To What Extent?" In: *Privacy in Statistical Databases*. Springer. 2012, pp. 348–363.

[8] Leyla Bilge, Sevil Sen, and Christopher Kruegel Balzarotti Engin Kirda. "EXPOSURE: a passive DNS analysis service to detect and report malicious domains". In: *ACM Transactions on Information and System Security (TISSEC), 2014, ISSN: 1094-9224* (Jan. 2014). URL: http://www.eurecom.fr/publication/4209.

[9] Leyla Bilge et al. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." In: *2011 Network and Distributed System Security Symposium (NDSS)*. 2011. URL: http://www.iseclab.org/papers/bilge-ndss11.pdf.

[10] Burton H Bloom. "Space/time trade-offs in hash coding with allowable errors". In: *Communications of the ACM* 13.7 (1970), pp. 422–426.

[11] Edward Bortnikov et al. "Brahms: Byzantine resilient random membership sampling". In: *Computer Networks* 53.13 (2009), pp. 2340–2359.

[12] Andrei Broder and Michael Mitzenmacher. "Network Applications of Bloom Filters: A Survey". In: *Internet Math.* 1.4 (2003), pp. 485–509. URL: http://projecteuclid.org/euclid.im/1109191032.

[13] Kenneth L Calvert et al. "Instrumenting home networks". In: *ACM SIGCOMM Computer Communication Review* 41.1 (2011), pp. 84–89.

[14] *Carrier IQ on Android - FAQ*. URL: http://blog.fortinet.com/post/carrier-iq-on-android-faq.

[15] David Chaum. "The dining cryptographers problem: Unconditional sender and recipient untraceability". In: *Journal of cryptology* 1.1 (1988), pp. 65–75. URL: http://link.springer.com/article/10.1007/BF00206326.

[16] Alice Cheng and Eric Friedman. "Sybilproof reputation mechanisms". In: *Proceedings of the 2005 ACM SIGCOMM workshop on Economics of peer-to-peer systems*. ACM. 2005, pp. 128–132.

[17] Benny Chor et al. "Private information retrieval". In: *Journal of the ACM (JACM)* 45.6 (1998), pp. 965–981.

[18] Henry Corrigan-Gibbs and Bryan Ford. "Dissent: accountable anonymous group messaging". In: *Proceedings of the 17th ACM conference on Computer and Communications Security*. ACM. 2010, pp. 340–350. URL: http://korz.cs.yale.edu/dissent/papers/ccs10/dissent.pdf.

[19] Jakub Czyz et al. "Taming the 800 Pound Gorilla: The Rise and Decline of NTP DDoS Attacks". In: *Proceedings of the 2014 Conference on Internet Measurement Conference*. IMC '14. Vancouver, BC, Canada: ACM, 2014, pp. 435–448. ISBN: 978-1-4503-3213-2. DOI: 10.1145/2663716.2663717. URL: http://doi.acm.org/10.1145/2663716.2663717.

[20] George Danezis, Roger Dingledine, and Nick Mathewson. "Mixminion: Design of a type III anonymous remailer protocol". In: *2003 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2003, pp. 2–15. URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1199323.

[21] Frank Denis. *Introducing Sodium, a new cryptographic library*. 2013. URL: http://labs.opendns.com/2013/03/06/announcing-sodium-a-new-cryptographic-library/ (visited on 08/21/2014).

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. DTIC Document, 2004. URL: http://www.dtic.mil/dtic/tr/fulltext/u2/a465464.pdf.

[23] *DITL Traces and Analysis*. Domain Name Systems Operations Analysis and Research Center. 2014. URL: https://www.dns-oarc.net/oarc/data/ditl (visited on 03/03/2015).

[24] *DNSCurve: Usable Security for DNS*. URL: http://dnscurve.org/index.html.

[25] John R Douceur. "The sybil attack". In: *Peer-to-peer Systems*. Springer, 2002, pp. 251–260.

[26] Cynthia Dwork. "Differential privacy: A survey of results". In: *Theory and Applications of Models of Computation*. Springer, 2008, pp. 1–19.

[27] Matthew Edman and Bülent Yener. "On anonymity in an electronic society: A survey of anonymous communication systems". In: *ACM Computing Surveys (CSUR)* 42.1 (2009), p. 5. URL: http://www.cs.ucf.edu/~dcm/Teaching/COT4810-Spring2011/Literature/AnonimityCommunication.pdf.

[28] Tariq Elahi, George Danezis, and Ian Goldberg. "PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks". In: *CCS '14: proceedings of the 21st ACM Conference on Computer and Communications Security*. 2014, pp. 1068–1079.

[29] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. "RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2014, pp. 1054–1067.

[30] Jon Fingas. *Windows 10 will deliver updates through your fellow PC users*. engadget. Mar. 2015. URL: http://www.engadget.com/2015/03/15/windows-10-peer-to-peer-updates/.

[31] Michael J Freedman and Robert Morris. "Tarzan: A peer-to-peer anonymizing network layer". In: *Proceedings of the 9th ACM conference on Computer and Communications Security*. ACM. 2002, pp. 193–206. URL: http://ecee.colorado.edu/~ekeller/classes/fall2013_advsec/papers/tarzan_ccs02.pdf.

[32] Saikat Guha and Neil Daswani. *An experimental study of the Skype peer-to-peer VoIP system*. Tech. rep. Cornell University, 2005. URL: http://dspace.library.cornell.edu/bitstream/1813/5711/1/TR2005-2011.pdf.

[33] Michael Herrmann. "Privacy-Implications of Performance-Based Peer Selection by Onion-Routers: A Real-World Case Study using I2P". MA thesis. Technische Universität München, 2011. URL: https://131.159.74.67/sites/default/files/herrmann2011mt.pdf.

[34] Susan Hohenberger, Steven Myers, Rafael Pass, et al. "ANONIZE: A Large-Scale Anonymous Survey System". In: *2014 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2014, pp. 375–389.

[35] *I2NP Specification*. URL: https://geti2p.net/en/docs/spec/i2np (visited on 07/21/2015).

[36] *Inside TAO: Documents Reveal Top NSA Hacking Unit*. SPIEGEL Online. Dec. 2013. URL: http://www.spiegel.de/international/world/the-nsa-uses-powerful-toolbox-in-effort-to-spy-on-global-networks-a-940969-2.html.

[37] Rob Jansen et al. "The Sniper Attack: Anonymously Deanonymizing and Disabling the Tor Network". In: *2014 Network and Distributed System Security Symposium (NDSS)*. 2014. URL: http://www.robgjansen.com/publications/sniper-ndss2014.pdf.

[38] Aaron Johnson et al. "Users get routed: Traffic correlation on Tor by realistic adversaries". In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, pp. 337–348.

[39] Brian Neil Levine, Clay Shields, and N Boris Margolin. *A survey of solutions to the sybil attack*. Tech. rep. University of Massachusetts, Amherst, 2006. URL: http://forensics.umass.edu/pubs/levine.sybil.tr.2006.pdf.

[40] Petar Maymounkov and David Mazières. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric". In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems*. IPTPS '01. London, UK, UK: Springer-Verlag, 2002, pp. 53–65. ISBN: 3-540-44179-4. URL: http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf.

[41] Carlos Aguilar Melchor et al. "High-speed Private Information Retrieval Computation on GPU". In: *Second International Conference on Emerging Security Information, Systems and Technologies, 2008. SECURWARE-08.* IEEE. 2008, pp. 263–272.

[42] Alan Mislove et al. "AP3: Cooperative, decentralized anonymous communication". In: *Proceedings of the 11th workshop on ACM SIGOPS European workshop*. ACM. 2004, p. 30.

[43] Prateek Mittal and Nikita Borisov. "Information leaks in structured peer-to-peer anonymous communication systems". In: *Proceedings of the 15th ACM conference on Computer and communications security*. ACM. 2008, pp. 267–278.

[44] Yves-Alexandre de Montjoye et al. "Unique in the Crowd: The privacy bounds of human mobility". In: *Scientific reports* 3 (2013).

[45] Steven J Murdoch and George Danezis. "Low-cost traffic analysis of Tor". In: *2005 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2005, pp. 183–195.

[46] Steven J Murdoch and Piotr Zieliński. "Sampled traffic analysis by internet-exchange-level adversaries". In: *Privacy Enhancing Technologies*. Springer. 2007, pp. 167–183.

[47] Laura Nelson. "New traffic apps may be pushing cars into residential areas". In: *Los Angeles Times* (2015). URL: http://www.latimes.com/local/california/la-me-california-commute-20150106-story.html (visited on 08/10/2015).

[48] Daniel Nurmi, John Brevik, and Rich Wolski. "Modeling machine availability in enterprise and wide-area distributed computing environments". In: *Euro-Par 2005 Parallel Processing*. Springer, 2005, pp. 432–441.

[49] Larry Peterson et al. "Planetlab architecture: An overview". In: *PlanetLab Consortium May* 1.15 (2006), pp. 4–1.

[50] *Popularity-contest Frequently Asked Questions*. URL: `http://popcon.debian.org/FAQ` (visited on 03/03/2015).

[51] Johan A Pouwelse et al. *A Measurement Study of the BitTorrent Peer-to-Peer File-Sharing System*. Tech. rep. PDS-2004-003. Delft University of Technology, The Netherlands, 2004. URL: `http://www.pds.twi.tudelft.nl/~pouwelse/bittorrent_measurements.pdf`.

[52] *Protocol Buffers*. Google. URL: `https://developers.google.com/protocol-buffers/` (visited on 09/03/2014).

[53] Krishna PN Puttaswamy, Ranjita Bhagwan, and Venkata N Padmanabhan. "Anonygator: Privacy and integrity preserving data aggregation". In: *Middleware 2010*. Springer, 2010, pp. 85–106.

[54] Michael K Reiter and Aviel D Rubin. "Crowds: Anonymity for web transactions". In: *ACM Transactions on Information and System Security (TISSEC)* 1.1 (1998), pp. 66–92.

[55] Marc Rennhard. "MorphMix–A Peer-to-Peer-based System for Anonymous Internet Access". PhD thesis. Swiss Federal Institute of Technology Zurich, 2004. URL: `https://home.zhaw.ch/~rema/publications/PhDMorphMix.pdf`.

[56] Farsight Security. *Passive DNS Database - DNSDB*. URL: `https://www.farsightsecurity.com/Services/` (visited on 03/03/2015).

[57] Farsight Security. *Passive DNS Sensor FAQ*. Sept. 2013. URL: `https://archive.farsightsecurity.com/Passive_DNS_Sensor_FAQ/` (visited on 03/03/2015).

[58] Atul Singh et al. "Eclipse attacks on overlay networks: Threats and defenses". In: *In IEEE INFOCOM*. Citeseer. 2006.

[59] Robin Sommer and Vern Paxson. "Outside the closed world: On using machine learning for network intrusion detection". In: *2010 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2010, pp. 305–316. URL: `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5504793`.

[60] Moritz Steiner, Taoufik En-Najjary, and Ernst W Biersack. "Long term study of peer behavior in the KAD DHT". In: *IEEE/ACM Transactions on Networking (TON)* 17.5 (2009), pp. 1371–1384.

[61] Latanya Sweeney. "k-anonymity: A model for protecting privacy". In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10.05 (2002), pp. 557–570.

[62] *The first stage of the WER protocol is not SSL encrypted in Windows*. Microsoft. Mar. 2014. URL: https://support.microsoft.com/en-us/kb/2929733.

[63] *Tor FAQ*. URL: https://www.torproject.org/docs/faq.html.en (visited on 07/26/2015).

[64] Hakon Gabriel Verespej. "A characterization of node lifetime distributions in the PlanetLab test bed". MA thesis. UC San Diego, 2010.

[65] Dan S Wallach. "A survey of peer-to-peer security issues". In: *Software Security–Theories and Systems*. Springer, 2003, pp. 42–57.

[66] *Waze - Privacy Policy*. Waze. Mar. 2015. URL: https://www.waze.com/legal/privacy/.

[67] Philipp Winter et al. "Spoiled onions: Exposing malicious Tor exit relays". In: *Privacy Enhancing Technologies*. Springer. 2014, pp. 304–331.

[68] David Isaac Wolinsky et al. "Dissent in Numbers: Making Strong Anonymity Scale". In: *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*. 2012, pp. 179–182.

[69] Matthew K Wright et al. "The predecessor attack: An analysis of a threat to anonymous communications systems". In: *ACM Transactions on Information and System Security (TISSEC)* 7.4 (2004), pp. 489–522.

[70] Praveen Yalagandula et al. "Beyond Availability: Towards a Deeper Understanding of Machine Failure Characteristics in Large Distributed Systems". In: *Proceedings of the 1st Workshop on Real, Large Distributed Systems (WORLDS '04)*. USENIX Association. 2004.

[71] Jonathan Zdziarski. "Identifying back doors, attack points, and surveillance mechanisms in iOS devices". In: *Digital Investigation* 11.1 (2014), pp. 3–19.

[72] Ge Zhong and Urs Hengartner. "A distributed k-anonymity protocol for location privacy". In: *IEEE International Conference on Pervasive Computing and Communications, 2009. PerCom 2009*. IEEE. 2009, pp. 1–10.

Vita