Ryan Wails*, Andrew Stange, Eliana Troper, Aylin Caliskan, Roger Dingledine, Rob Jansen, and Micah Sherr

Learning to Behave: Improving Covert Channel Security with Behavior-Based Designs

Abstract: Censorship-resistant communication systems generally use real-world *cover* protocols to establish a *covert* channel through which uncensored communication can occur. Unfortunately, many previously proposed systems use cover protocols inconsistently with the way humans normally use those protocols, leading to anomalous network traffic patterns that have been shown to be discoverable by real-world censors. In this paper, we argue that censorship-resistant communication systems should follow two behavior-based design properties: (i) *behavioral independence*: systems should isolate the operation of their covert channels from the operation of their cover protocols, and (ii) *behavioral realism*: systems should either opportunistically use existing genuine cover protocol instances or run new protocol instances that are modeled after genuine ones. These properties ensure that the behavior of a system's users will not degrade its security. We demonstrate how to achieve these properties through the design and evaluation of Raven, a censorship-resistant messaging system that uses email cover protocols identically to the way humans use email. Raven uses a generative adversarial network that is trained on genuine email data to control the timing and sizes of the email messages it sends and receives, and these messages are transferred independently of user actions. Our evaluation shows that, compared to the state-of-the-art email-based Mailet system, Raven raises the false-positive rate from 3% to 50% when detecting covert channel usage with 100% recall.

Keywords: covert communication; Internet censorship

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

***Corresponding Author: Ryan Wails:** Georgetown, U.S. Naval Research Laboratory; ryan.wails@nrl.navy.mil
Andrew Stange: Carnegie Mellon; astange@andrew.cmu.edu
Eliana Troper: Georgetown Univ.; st1038@georgetown.edu
Aylin Caliskan: University of Washington; aylin@uw.edu
Roger Dingledine: The Tor Project; arma@torproject.org
Rob Jansen: U.S. Naval Research Laboratory; rob.g.jansen@nrl.navy.mil
Micah Sherr: Georgetown Univ.; msherr@cs.georgetown.edu

1 Introduction

Internet freedom worldwide is becoming increasingly more restricted [24] as new censorship practices are developed and deployed by nation-states seeking to control the flow of information for political, social, or economic gain [51, 53]. To combat internet censorship, researchers and activists have proposed numerous censorship-resistant tools and techniques that can help restore access to information online [41, 61]. The proposed systems include those that attempt to mimic known protocols [16, 17, 48, 66, 68], those that tunnel covert traffic through multimedia and other cloud-based channels [6, 11, 37, 43, 46, 48, 66], those that attempt to obscure the destination at the routing layer [10, 18, 28, 35, 39, 49, 73, 74], and those that try to produce polymorphic behavior [4, 69, 70].

Most systems are designed to create a *covert* communication channel to a proxy server using a *cover* protocol that should appear innocuous to any censor that can observe its execution. The proxy server then assists in accessing the originally censored information.

Unfortunately, many of the previous systems suffer from limitations that allow more sophisticated censors to identify the presence of the covert channel and block or disrupt it [41, 61]. A primary limitation of previously proposed systems is that they operate cover protocols inconsistently with *de facto* cover protocol operation in the wild [36]. Recent work has demonstrated how inconsistent cover protocol operation can lead to detection and blocking [15, 19, 25, 29, 36] when the censorship-resistant system: (i) exhibits non-standard or unusual responses to errors or other active probes, or (ii) uses valid but unusual choices in implementation which are visible to passive observers. Running a genuine application to instantiate a cover protocol (rather than mimicking it) is now a generally-recognized requirement [36].

However, even if the cover protocol is *perfectly* emulated within a genuine application, the usage of the covert channel could produce traffic inconsistencies (e.g., timing and volume) in the cover protocol that appear anomalous to the censor when compared to the way other humans normally use that protocol. For ex-

ample, naïvely running a user's interactive web browsing session over an email cover protocol [38] could produce recognizable bursts of emails. Using traffic analysis, a censor can learn distinguishing features and use them to discover and block the channel [7, 31, 36, 58, 65]. Ensuring that the covert channel usage behavior that is observable in the cover protocol is indistinguishable from normal use of that protocol is an outstanding problem that recent work does not convincingly solve [7].

Behavior Properties: The traffic analysis issues described above lead us to argue for the following properties when designing censorship-resistant systems. Most importantly, the goal is to ensure that a censor that observes the network traffic produced by the operation of the cover protocol is unable to distinguish it from other benign executions of that protocol. First, we argue for *behavioral independence*: the operation of the covert channel by the user should be isolated from the operation of the cover protocol by the censorship-resistant system software. Isolating user actions from the operation of the cover protocol ensures that users do not unintentionally degrade security. Second, we argue for *behavioral realism*: the cover protocol should be operated according to its *de facto* operation in the real world. This can be achieved either by opportunistically using available existing executions of the cover protocol, or by running new cover protocol instances that are closely modeled after real world protocol behavior.

Raven: We believe that our behavior-based design properties would benefit many censorship-resistant communication systems. However, in this paper, we focus on a narrow set of text-based cover protocols in order to demonstrate the effectiveness of incorporating behavioral independence and realism into a concrete system designed for an established use-case. To this end, we design, implement, and evaluate Raven,[1] a proof-of-concept censorship resistance system that is designed to operate email cover protocols *identically* to the way genuine users would operate email protocols while isolating covert actions from cover protocol execution.

In Raven, clients and proxies use standard encrypted email cover protocols (e.g., SMTPS or IMAPS) and communicate through legitimate email service providers. Because of the link encryption to the service provider, the censor is never in a position to observe email addresses. Further, the client is free to choose (or switch) its email provider, so completely blocking Raven requires blocking all email providers outside of the censor's control (an action bearing severe economic impact [11, 23, 28, 34]). The cover email protocols are executed using popular email clients (e.g., Thunderbird) which respond as they normally would to malformed inputs or errors and enable Raven to resist protocol emulation attacks [15, 44, 45]. The client and a proxy regulate the frequency and size of emails following the output of a model that we trained on genuine email protocol behavior using generative adversarial networks (GANs) [54], ensuring that they do not produce anomalous transmission patterns that are detectable through traffic analysis [7, 31, 36, 58, 65].

Raven supports covert interaction with TLS-based services through the use of Intel processor Software Guard Extensions (SGX) [3]. More specifically, Raven enables secure, delay-tolerant TLS communication with Twitter, an established use-case for censorship circumvention tools [42, 59]. In particular, Twitter has become a recent target of state-sponsored censorship activity as governments attempt to control the flow of information on their networks [76], and Raven could be used to combat these efforts. Raven clients use SGX to securely establish fully encrypted and authenticated TLS connections through a proxy server. The proxy fetches TLS resources on behalf of the client and sends them over encrypted email but is unable to read or modify them.

Contributions: We make the following primary contributions. In §2, we analyze previously proposed censorship-resistant systems for their ability to achieve behavioral realism and behavioral independence and find that no previous system simultaneously achieves both properties. In §3, we further motivate the importance of considering protocol behavior and demonstrate how previous email-based censorship-resistant systems that do not provide behavior independence [38, 42] can be trivially detected through traffic analysis. We present the results from a novel internet measurement of the accessibility of email protocols in censored regions in §4.1, wherein we find that out-of-country email connections are allowed to some degree for all tested countries. We present the design of Raven, a novel censorship-resistant communication system, in §4.3; we are the first to incorporate GANs trained on genuine user behavior in a way that also prevents the censor from identifying our covert channels due to behavior emulation flaws, and we are the first to use SGX to enable delay-tolerant censorship-resistant communication. Finally, we evaluate Raven in §5; we find that Raven raises the false-positive rate in identifying email cover protocols with 100% recall from

1 In Game of Thrones, ravens enable long-distance communication between communities using complex training methods [30].

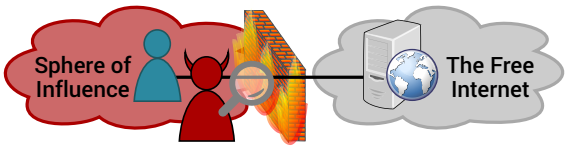

Fig. 1. Censorship Model

3% to 50% compared to the state of the art while offering reasonable performance for periodic use of Twitter.

2 Background and Related Work

Background: The censorship model that we consider is shown in Figure 1 and is based on standard assumptions from previous work [41, 61]. A user in a jurisdiction controlled by the censor (i.e., the *sphere of influence* [41]) wishes to communicate with other users or publishers that exist on the free and open internet (outside the sphere of influence). The censor's goal is to block access to information on the free internet that it determines is prohibited. The user's goal is to circumvent censorship and restore access to information using censorship-resistant tools and protocols.

The censor does wish to maintain users' general connectivity to the internet; although its goal is to block access to prohibited information, it does not want to block access to benign information for economic reasons. Hence, the censor attempts to minimize access to prohibited information (false negatives) while limiting the *collateral damage* caused by erroneously blocking accesses to benign information (false positives).

The censor can passively monitor published information from sources outside of its sphere of influence. Within its sphere of influence, the censor controls the network over which the user is attempting to communicate. The censor can passively observe packets, flows, and hosts within its network; if packet payloads are encrypted, the censor can passively observe the ciphertexts but cannot break encryption. The censor can perform traffic manipulation and traffic analysis on all traffic it observes in order to aid the blocking of packets, flows, or hosts or to discover, inspect, and block censorship circumvention systems. It can actively probe any host inside or outside of its sphere of influence, and analyze distinguishing traffic features such as destination addressing, size, timing, payload content, protocol semantics, and user behavior semantics.

Related Work: Numerous systems have been proposed to resist censorship [41, 61]. These systems include those based on protocol mimicry [16, 17, 48, 66, 68], protocol tunneling [6, 11, 26, 37, 43, 46, 48, 66], polymorphism [4, 69, 70], browser-based or scalable proxy systems [20–22, 50], refraction networking [10, 18, 27, 28, 35, 39, 49, 63, 73, 74], and cloud fronting systems [11, 23, 34, 77]. Many of these approaches are vulnerable to traffic analysis [7, 25, 31, 36, 65] and other active attacks [15, 19, 29, 57]. We believe that most of these systems would improve if they adopted techniques for behavioral independence and realism, and we enable new opportunities for future work to explore this space.

Table 1. Survey of behavioral independence and realism in tunnel-based censorship-resistant systems.

	Castle	CovertCast	CloudTransport	DeltaShaper	Freewave	HTTPT	Mailet	Protozoa	Rook	SkypeMorph	SWEET	Raven
Independence	○	○	○	◐	○	○	○	●	○	◐	○	●
Realism	○	○	○	○	○	○	○	○	●	●	○	●
Citation	[33]	[46]	[11]	[6]	[37]	[26]	[42]	[8]	[64]	[48]	[38]	—

● the property is fully satisfied by the system
◐ attempted, but the property is only partially satisfied
○ the property is not satified by the system

Tunnel-based censorship-resistant systems are particularly relevant to the design of Raven, which tunnels covert traffic over email (see §4.3). As shown in Table 1, most previously-proposed tunnel-based systems, including Castle [33], CovertCast [46], CloudTransport [11], Freewave [37], HTTPT [26], Mailet [42], and SWEET [38], achieve neither behavioral independence nor behavioral realism. These systems do not realistically model human behavior, and user actions in the covert channel may leak a significant amount of information when executing their cover protocols.

No system surveyed fully achieves behavioral independence and behavioral realism. However, four previous systems do make some attempt at achieving these properties: Protozoa [8] and Rook [64] achieve behavioral independence and behavior realism, respectively, while not supporting the other property. DeltaShaper [6] attempts to adjust cover traffic to avoid introducing new signals, but the adjustment process is heuristic and user actions could still leak through to the cover protocol. SkypeMorph [48] sends traffic according to a shaping oracle that is trained on the distribution of Skype messages' sizes and times. Strictly following the shaping oracle's decisions when shaping traffic would produce behavior-realistic traffic. However, SkypeMorph violates behavioral independence because it (i) imme-

diately sends packets as covert messages arrive and (ii) does not send packets when there are no covert messages to send. The traffic morphing mode of SkypeMorph is a promising method to achieve traffic analysis resistance, but security relies on the choice of a realistic target (cover) distribution [72].

Systems that use email as a cover protocol (as Raven does) include SWEET [38] and Mailet [42]. SWEET proposes to use email to transport both upstream and downstream traffic while covertly web browsing [38], while Mailet is designed to use email to communicate with proxies that access social sites such as Twitter [42]. Neither SWEET nor Mailet achieve behavioral independence or provide a way to produce behavioral-realistic email sending and receiving patterns, making them vulnerable to detection by traffic analysis. We show in §3.1.3 how Mailet's traffic patterns are distinguishable from genuine email patterns.

3 Behavior-Based Protocols

In this section, we discuss the basic properties of behavior modeling that generally apply to censorship circumvention systems.

3.1 Properties of Behavior Modeling

Traffic analysis attacks pose a serious threat to censorship circumvention techniques. In practice today, censors tend to favor cheaper approaches (for example, active probing) to identify and block hosts participating in censorship circumvention protocols. But, new circumvention systems are being designed to frustrate these cheaper attacks [17, 26], which may force a censor to instead rely on traffic analysis to detect these hosts. Moreover, traffic analysis attacks using machine learning classifiers have been shown to be highly effective at detecting network protocol obfuscation [7, 65].

Traffic analysis attacks search through protocol traffic for a signal that enables an attacker to distinguish and separate ordinary, benign protocol traffic flows from those used as cover traffic for censorship circumvention. This signal often takes the form of network packets' transmission times and sizes. For example, Wang et al. found that meek, a circumvention system, produced traffic that exhibited unusual TCP ACK transmission times. A machine learning classifier given ACK intervals as a feature is able to distinguish meek traffic from among ordinary HTTPS traffic (meek's cover protocol) [65]. Many prior works haphazardly model cover protocol behavior, or do not model behavior at all, rendering them susceptible to detection through traffic analysis.

We argue that realistic behavior modeling is a crucial component of censorship circumvention systems. We focus on two key properties of behavior modeling that are often overlooked: (i) *behavioral independence*, and (ii) *behavioral realism*.

3.1.1 Behavioral Independence

Behavioral independence is satisfied if inputs to and actions made by the covert channel do not affect the observable behavior of the cover protocol; e.g., the usage behavior of Twitter (the covert channel) is independent of the operation of the email cover protocols. In other words, behavioral independence is achieved if the cover protocol behaves identically, regardless of which covert channel inputs are given. This property alone is not sufficient for covertness, but does provide a type of traffic analysis resistance—if cover protocol traffic is independent of covert channel traffic, then the censor cannot learn anything about the behavior of the covert channel by observing the cover protocol.

System security can be degraded in two ways when behavioral independence is not achieved. First, the covert channel behavior may introduce artifacts into the cover protocol messages that do not occur frequently with ordinary cover protocol operation, which may serve as a signal to detect the presence of the covert traffic. This degradation is particularly problematic when *user actions* (for example, clicking HTTP links in a covert web-browsing session) affect cover protocol behavior, because it is impractical to rely on users to provide action sequences that result in realistic cover behavior. A case of this reliance is seen in the SWEET email-based censorship circumvention system, where it is assumed that users make only up to 70 webpage fetches per day as to not transmit emails too frequently [38]. Second, if the censor is able to identify and monitor a covert channel within a cover medium, then the censor may be able to infer the contents of covert messages through the information leaked by the cover protocol.

3.1.2 Behavioral Realism

As stated above, behavioral independence is not sufficient to provide channel covertness—an important design question remains: which actions should the cover protocol make? Using the wrong choice of inputs may

lead to out-of-the-ordinary cover protocol behavior (for example, sending messages too quickly) that can be used by the censor to identify channel usage. Behavioral realism requires that the distribution of observable behavior exhibited by the cover protocol is similar to the distribution of behaviors exhibited by typical, genuine usage of the protocol. This property may be achieved by measuring the distribution of *inputs* to the cover protocol during genuine executions, and sampling new sequences of inputs from the learned distribution. Some previously-proposed censorship circumvention systems use heuristic approaches to achieve realism, like rate limiting protocol inputs [33, 38]. However, the choice of parameters in the heuristics are made haphazardly. Instead, we recommend that user actions be explicitly modeled from genuine protocol inputs. In §3.2, we discuss techniques that can be used to model these distributions.

3.1.3 A Study of Mailet

To study the effects that behavior modeling can have on a censorship circumvention system, we examine the Mailet censorship circumvention system which is designed to provide Twitter microblogging services over email [42]. All covert protocol messages are carried within emails, which are encrypted using TLS when transferred between email clients and servers.

In Mailet's design, *user actions determine the sizes and times of emails sent through the system.* For example, when a user wishes to Tweet, the Mailet client immediately sends an email containing the Tweet contents to a Mailet server—there is no queuing or shaping applied to the email. Upon posting a Tweet successfully, the Mailet server immediately responds with a 7 byte `success` string. Li and Hopper argue that Mailet behaviors do not significantly differ from typical email behaviors; i.e., that Mailet behavior is, by nature, difficult to distinguish from genuine email behavior. However, note that Mailet violates behavioral independence, and can violate behavioral realism (if Tweet sizes and times vary significantly from genuine emails). Through a simple traffic analysis attack, we will demonstrate that covert traffic sent by Mailet is easily detected and blocked.

Simulated Dataset: Mailet is not a publicly deployed system, so to analyze its behavior we simulate its usage using the open-source Mailet implementation [2]. Following Li and Hopper, we consider a scenario where Mailet users can (i) post Tweets and (ii) search keywords. We used archived data from August 2019 of Twitter's "Spritzer" stream, a 1% sample of all public Tweets [67]. The corpus contains 97,261,581 Tweets made by 28,564,408 users. From this corpus of Tweets and users, we sampled 100 verified users at random.[2]

To simulate Mailet Tweet behavior, we obtained each user's complete Tweet history from August 1 2019 to October 1 2019 and recorded Tweet sizes and timestamps. To simulate Mailet searches, we identified 250 popular hashtags in the corpus. We obtained Twitter's search results for each keyword with contents from August 1 2019 to October 1 2019. Following Li and Hopper, we assume an average of 4–5 searches are made by each user for each Tweet. We assume the Twitter server responds with between 1 and 100 (the maximum response size) search results, chosen at random. We run the Mailet code to obtain the Mailet emails sent for each Tweet and search.

Evaluation: We treat the task of identifying Mailet emails from among genuine emails as a supervised binary classification machine learning task. Each day of email behavior constitutes a single input example. We use a 30-dimensional feature space where each feature is a summary statistic (count, mean, standard deviation, skewness, kurtosis, minimum, maximum, and percentiles 0.1–0.9) describing the distribution of sizes for emails that were sent and received on a day. These features have been shown to be informative when detecting covert channels [7].

From a corpus of genuine email usage collected from this paper's authors, (discussed further in §5.2) we randomly select six genuine email users and six Mailet users to constitute the training dataset. From each of these twelve users, we sample 60 days worth of input examples. On this training data, we train a random forest classifier (with 100 trees) using the `scikit-learn` Python package. Then, we evaluate classifier performance on a test set of 60 examples taken from one genuine mail user and one Mailet user (distinct from the training users). We used k-fold cross validation and repeated this process for $k = 7$ different folds of training and testing data.

A summary of classifier performance is given as follows. **On average, the classifier was able to achieve 100% recall**, i.e., $\frac{TP}{P}$ where TP is the number of true positives (correctly labeled Mailet examples) and P is the total number of positive/Mailet examples

2 During this sampling, users were partitioned into 20 evenly-spaced percentiles based upon their Tweet frequencies. 5 users were sampled uniformly from each part to ensure we obtained a range of Twitter behaviors.

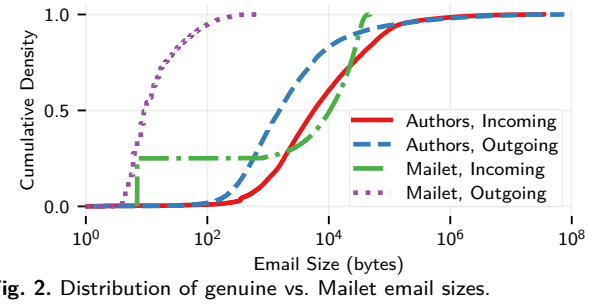

Fig. 2. Distribution of genuine vs. Mailet email sizes.

in the test set. The classifier achieved a relatively-low 3% false positive rate, i.e., $\frac{FP}{N}$ where FP is the number of false positives (genuine examples predicted to be Mailet) and N is the total number of negative/genuine examples in the test set. In other words, on average, the trained classifier was able to identify all instances of Mailet usage, while incorrectly labeling only 3% of genuine email examples as originating from Mailet.

This relatively strong classifier performance is due to Mailet's distinctive email sizes. Figure 2 shows the distribution of all email sizes in the genuine email dataset we collected and the simulated Mailet dataset. Mailet email traffic has unique characteristics: (i) there is a high prevalence of small, incoming emails in Mailet due to the broker sending a 7 byte `success` string upon each successful Tweet; and (ii) most outgoing email is small relative to ordinary email, as it contains only a short Tweet or keyword to search.

Both of these unique characteristics highlight the importance of behavior modeling. By definition, a system with behavioral independence cannot leak covert-channel behavior, such as the transmission of small control messages, through the cover protocol. A system achieving behavior realism must transmit emails with sizes that are in accord with the genuine email patterns, precluding the case seen in Mailet where unusually-many small emails are sent.

3.2 Methods for Behavior Modeling

Traffic analysis vulnerabilities, such as the one observed in Mailet, emphasize the need to adopt behavioral independence and realism in covert channel design. Two techniques can be used to achieve these properties.

To satisfy behavioral independence, cover protocol inputs can be scheduled and executed regardless of which covert channel behaviors need to occur. (These inputs should be scheduled according to a distribution of realistic user behavior, described further below.) Mismatches between cover protocol performance limitations and covert channel performance requirements can lead to poor covert channel goodput. In §3.3, we discuss design choices that can be used to improve performance.

A model of the distribution of typical cover protocol inputs can be used to achieve behavioral realism. Learning such a distribution consists of two tasks: (i) gathering genuine user inputs, and (ii) building a model to encode the distribution of inputs. Once inputs are gathered, *generative machine learning models* can be a useful tool for learning distributions—given a number of unlabled input examples $x_1, x_2, \ldots, x_n$ from a distribution D, a generative model can learn a synthetic distribution $\hat{D} \approx D$ and produce a new *synthetic* example $\hat{x} \sim \hat{D}$ such that $\hat{x} \neq x_i$ for all $1 \leq i \leq n$. Generative models based on deep neural networks, such as generative adversarial networks (GANs), have been trained to accurately model highly complex distributions, such as images of human faces [32, 40]. Using generative models, as opposed to replaying behaviors from a corpus of user behavior, prevents the censor from obtaining the corpus and testing if behaviors are contained within the corpus. In §5, we show how we trained GANs to learn and produce email behaviors.

3.3 Considerations of Modeling

Although behavior modeling can improve circumvention-system covertness, it can also degrade performance. The rigid schedule of inputs given to the cover protocol (e.g., "send the next email 3 hours from now") may disagree with the performance desiderata of the covert channel (e.g., "post a Tweet now"). In particular, high latency cover protocols cannot be used naïvely for low-latency applications. For example, a TCP connection cannot be tunneled over email with human-scale delays inserted without experiencing timeouts. Similarly, low bandwidth cover protocols (e.g., VoIP) cannot be used to realize high bandwidth covert channels (e.g., streaming HD video) without violating behavior modeling principles.

To realize useful functionalities over high-latency channels, application proxies can be used. For example, instead of trying to tunnel raw TCP payloads from a client to a web server over email, an HTTP proxy can receive a short command (e.g., "fetch `https://example.com`") and dispatch the command on behalf of the client. Unfortunately, most designs for application proxying require trust in the proxy to execute the functionality correctly, introducing an additional point of failure. Instead, we recommend proxy

designs that implement these functionalities *securely*. In the case study that follows, we show that secure hardware enclaves can be used to securely implement the transmission of TLS-protected HTTP commands.

4 Case Study: Communicating over Email Cover Protocols

In this section, we present a case-study in designing a censorship-resistant communication system called Raven that uses email cover protocols and our behavior-based protocol design principles from §3 after first demonstrating the feasibility of using encrypted email as cover protocols in the real world.

4.1 Accessibility of Email Protocols

We conduct real world internet measurements to inform our use of email cover protocols for censorship resistance. We consider the use of email service providers that are located outside of the censor's sphere of influence, which is advantageous because it prevents the censor from directly observing email headers and content. We thus consider the question: *is it possible to access out-of-country email services using encrypted IMAP and SMTP from within censored regions?*

Towards answering this question, we first formed a baseline of countries that perform censorship. Using RIPE Atlas [55] probes distributed among 169 countries, we attempt HTTPS connections to Facebook, Google, Reddit, the National Democratic Institute (NDI), and the Tor Project—websites that the ICLab censorship measurement platform [51] report as being often subject to censorship. Prior work has shown that RIPE Atlas probes are largely located at service provider networks and are suitable for performing measurements from within provider networks [5]. We consider the 32 countries where we encountered the greatest number of unsuccessful HTTPS requests, where a request is considered unsuccessful if it either fails to connect or cannot validate the certificate, for example, due to TLS man-in-the-middle (MitM). Certificate validation was performed using the default trust roots packaged with Ubuntu Linux. The 32 countries that exhibited the most censorship are listed in Table 2 (see Appendix D). This set represents the regions that could potentially benefit the most from the use of email cover protocols given their prevalence of censorship.

We next determine the accessibility of email protocols in these regions. We attempted approximately 14,000 connections from RIPE Atlas probes located in these 32 countries to the SMTPS (465) and IMAPS (993) servers hosted by Gmail, Yahoo!, Riseup, and an email server operated by an author in North America. We include Gmail and Yahoo! since they represent major email providers with large numbers of users. Riseup is an organization that provides tools for "people and groups working on liberatory social change" [56], and thus we expect it to be particularly prone to blocking. Finally, the small single-user email server is included to measure whether countries block access to unknown email services. We consider a connection to be successful if and only if (i) the probe is able to connect to both the SMTPS and IMAPS ports and (ii) we are able to authenticate the certificate. We performed certificate verification at our institution (by post-processing the observed certificates) to mitigate the effects of corrupted roots of trust at the probes. Conceptually, our success criteria indicates instances in which a user could both send (SMTPS) and receive (IMAPS) email with the email server, without being subject to MitM. Since RIPE Atlas is incompatible with STARTTLS, our results are conservative, since some regions may support STARTTLS but not SMTPS.

From this measurement, we found that the country-averaged probe connection success rate to Gmail, Yahoo!, Riseup, and the personal email server were 86%, 87%, 84%, and 90%, respectively. The full set of measurement results is given in Table 2. Note that it is possible to use email protocols as cover protocols whenever a connection to *any* email service succeeds. For example, in China, only 45% of probes were able to connect to Gmail, but 85% could connect to Yahoo. Iran blocked 11% of probes' connections to Yahoo!, but did not block any connections to either Gmail or Riseup. Uganda and Kyrgyzstan exhibited the most blocking, but in both cases 50% or more of probes succeeded in establishing secure connections to out-of-country email servers.

Overall, we find that all tested countries allow some degree of out-of-country IMAPS and SMTPS, and that in most countries, the vast majority ($> 85\%$) of our probes could securely connect to the email servers. Our results also show that blocking unknown email servers (i.e., our local server) is rare.

4.2 Design Preliminaries

Assumptions: We derive three key assumptions based on our measurement study in §4.1 and the censorship model from §2 that will inform the design of our email-based communication channel in §4.3. First, we consider a blocklisting adversary that does not block *all* email protocols, but may block *specific* email service providers or individual servers. Second, we assume that any service located *inside* the censor's sphere of influence is untrusted and can be manipulated or otherwise compromised by the censor. And third, any service *outside* of the censor's sphere of influence is also untrusted and uncooperative: it cannot be directly manipulated, but the censor can pressure it. In particular, outside services may seek to discover and block service usage that does not comply with their terms of service agreements (as has happened in the case of domain fronting [60]). However, we assume that not all service providers will participate in *both* discovery *and* blocking.

Goals: Our primary design goal is to ensure that a censor that observes the network traffic produced by the operation of email cover protocols is unable to distinguish them from other benign executions of those protocols. If the censor suspects that some users within its sphere of influence are sending covert messages over email, it should be unable to detect those users with certainty. (Recall from §2 that the censor is unwilling to block broad email usage due to a high collateral damage that it would incur.) Guided by our behavioral realism and behavioral independence principles discussed in §3, we aim to achieve our indistinguishability goal by ensuring that the operation of the cover protocol is realistic and independent of user actions. Note that we aim to support high-latency, low-volume applications like Twitter; we explicitly consider low-latency, high-volume, and anonymous communication as out of scope.

4.3 Raven Design

We now present the design of Raven, an email-based censorship-resistant communication channel that provides independence between covert and cover protocols, uses generative adversarial networks (GANs) [54] to enforce behavior-realistic communication patterns, and uses SGX [3] to enable delay-tolerant TLS communication with Twitter (or other TLS-based services).

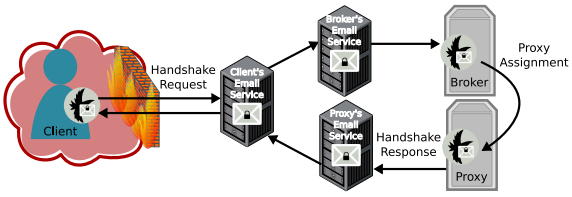

Fig. 3. To bootstrap, a single message is sent from the *client* to the *broker* and includes channel configuration parameters. The broker assigns the channel to a *proxy*, which responds to the client to form the Raven channel.

4.3.1 Operation of the Cover Protocols

Parties: Figure 3 shows the parties involved in Raven. The primary parties that run Raven software include a *client* that wants to circumvent censorship, and *broker* and *proxy* servers that facilitate circumvention to enable the user to access the free internet. Raven also utilizes third-party *email service providers* to facilitate communication between nodes. Our use of the email service providers aligns with the way normal humans use them; i.e., we use them to transport encrypted Raven messages while mimicking the stochastic traffic patterns of plausible email users. The service providers allow us to obfuscate the Raven servers as the destinations of Raven messages within the censor's sphere of influence, as traffic to and from the Raven servers would not be directly observable by the censor when using email service providers as intermediaries.

Bootstrapping: The bootstrapping process is shown in Figure 3. A user obtains the Raven client software and configures it to connect to a new client-side email account that is independent of the user's identity via the standard encrypted email protocols SMTPS and IMAPS. The client uses this account to send and receive email messages to and from a third-party email provider located outside of the sphere of influence.

We use a modular design for the Raven servers, which is possible due to the asymmetric and location-independent nature of email. The broker registers an email account with a public email service provider (e.g., Gmail). This account is only used to receive a single handshake message for each Raven client that wants to bootstrap a Raven channel, but not to send messages. This design choice allows new clients to reach the broker via a well-known public email address while the broker remains a low-volume user of the email service. On the other hand, the proxy uses a Raven mailserver that is capable of interacting with other email service providers. We run this mailserver so we can support higher volumes than some email service providers would find acceptable,

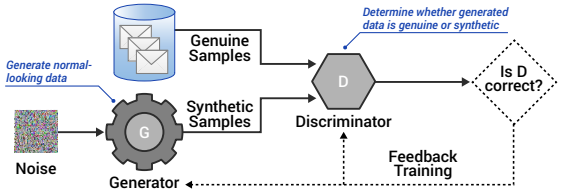

Fig. 4. The generator is trained using genuine email samples and a feedback loop with a discriminator.

and we can dynamically provision higher-volume services to carry the load from the interaction with clients over the covert channel. The Raven servers can temporarily go offline or migrate to different hosts if necessary; as long as their mail user agents can access their email providers, they are location-independent. This design is based on our initial deployment experiences, which we detail in Appendix B.

To bootstrap the Raven channel, the client sends an initial handshake request message (including channel configuration parameters) to the broker at its public email address. Upon receiving a handshake request from a client, the broker assigns the new request to a proxy. The proxy uses the client's choice of parameters to initialize a sending process that is *synchronized* with the client's receiving process, and then sends a handshake response to the client using the proxy's email service. After receiving the response, the client and proxy can continue communicating over the channel.

Behavioral Realism: Raven uses a behavior generator that is trained using generative adversarial networks (GANs) [54] to determine realistic email timings and sizes. GANs are trained to model a distribution over the training data and can efficiently generate new, synthetic samples from the learned distribution. The GAN training process consists of a game played between a *generator* that produces synthetic samples, and a *discriminator* that evaluates the samples. The goal of the generator is to produce samples that the discriminator is unable to distinguish from true data elements during evaluation. As shown in Figure 4, the generator is trained using genuine, human-generated email meta-data (email sending times and sizes) and a feedback loop with the discriminator to produce new sending patterns that are difficult to distinguish from genuine user behavior.

Raven requires genuine email meta-data (email times and sizes) for training the GAN, but our design offers flexibility on the data source. Some example sources include public email datasets such as the Enron [1] and Avocado [52] sets, email meta-data collected by a trusted organization such as a university or non-profit, or even users of Raven themselves. The choice of data source should consider the risk of a censor poisoning the data to inject a signal that could later be used to detect Raven traffic; some data sources are much less susceptible to such an attack. This issue is discussed further in § 5.1.

The email meta-data is distilled through a training process into a behavior generator model that contains a set of *user profiles* $\mathcal{P}$ that encompass the email sending patterns of one or more users. Depending on the data source, the model could be trained either (i) by the Raven deployment team or trusted affiliate and distributed with the Raven software (and model updates sent as compressed email attachments of a few MB), or (ii) by the Raven client on first start. Once trained (see §5.2), the generator model is used by a message handler component to shape email cover traffic, i.e., to determine precisely when to send email messages and how large they should be.

Behavioral Independence: Raven buffers user-initiated requests to send covert messages in order to enforce independence between the covert channel and the email cover protocols. When the behavior generator indicates that it is time to send an email, the message handler checks the outgoing user buffer; if application data is present in the buffer, the message handler will pack, fragment, and pad it to the correct size as instructed by the behavior generator. Otherwise, the handler generates a dummy message of the correct size. To obfuscate the message content, the message handler then GPG-encrypts the message (a standard method of encrypting email content) before passing it to the mail user agent (e.g., Thunderbird). The mail user agent writes the encrypted message into the body of a new email that it sends to the proxy's email address using an encrypted SMTPS connection to the email service provider.

When messages from the client are delivered to its inbox, the proxy receives them through an encrypted IMAPS connection with the proxy's mailserver. From here, the message moves through the reverse process that was applied at the client: the message is decrypted, stripped, extracted, and reassembled. Dummy messages are discarded, and legitimate application data is passed up to an incoming buffer that is read by the proxy's side of the Raven plugin. Once an initial message is received from the client, an analogous process to the one we described above (and in Figure 5) operates in the reverse direction (from proxy to client) in parallel on the proxy in order to support bidirectional communication.

Tuning Raven: Raven is tunable with respect to the user profiles: it allows clients to operate i user profiles in

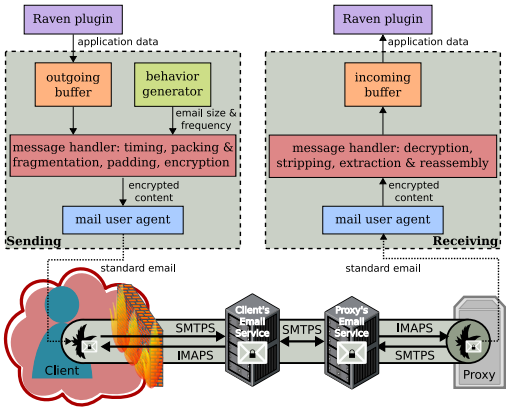

Fig. 5. The Raven channel design. Sending and receiving through the channel is analogous in both directions.

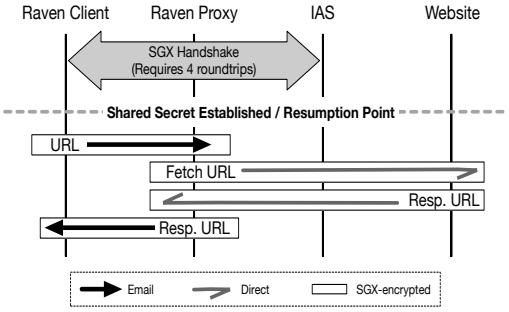

Fig. 6. A Raven client performs SGX remote attestation with a Raven proxy, and then remotely fetches TLS resources using the SGX-encrypted secure enclave.

parallel such that $1 \leq i \leq |\mathcal{P}|$. The i profiles are chosen at random from all possible combinations, and fixed for the duration of Raven usage. Larger values of i result in Raven sending and receiving more email traffic, which will increase the capacity of the covert channel but may also increase the suspicion of a network adversary observing the channel. Raven uses $i = 1$ as the default (the most secure setting), however, we consider that choosing $i > 1$ is reasonable since the network traffic that would result could have plausibly been created by multiple genuine email users who share an IP address due to network address translation (NAT) devices like home or commercial routers.[3]

4.3.2 Using the Covert Channel

Overview: Raven enables covert communication between its clients and proxies, which can support general, plug-in functionalities. Malicious proxies constitute a serious threat in this design. Tunneling HTTPS/TLS connections through Raven is not an option due to the timeouts that would occur as a result of the high latency of its email cover protocols. Instead, we design a secure, delay-tolerant proxy in a trusted execution environment (Intel SGX) that can verifiably transmit TLS messages, such as Twitter feeds, specified confidentially by the client and return the contents with confidentiality and integrity guarantees.

Software Guard Extensions: Intel SGX is a recent, popular trusted computing platform proposed by Intel [14]. SGX allows for trusted code to be verifiably and confidentially executed inside of a secure *enclave* process on an otherwise untrusted host; any other process (including kernel/ring-0 processes) cannot read or write enclave memory and cannot modify instructions executed inside the enclave. These properties are supported by the CPU and motherboard—special assembly instructions are used to enter and leave a protected enclave process, and the CPU and memory-management unit ensure that enclave memory is only decrypted and accessible when the enclave is running.

The SGX platform includes a *remote attestation* procedure that allows an external process (in particular, a non-enclave process running on another host) to (i) establish a shared-secret with an enclave process, and (ii) become convinced that the enclave process is correctly running an unmodified version of a known, trusted program. The shared-secret is established using a Diffie-Hellman handshake that is partially executed inside the enclave. To prove program correctness, the enclave produces a "quote", i.e., a cryptographic hash of the enclave's code and data memory pages. This quote is checked and signed by Intel's Attestation Service (IAS) and presented to the client.[4]

Note that Intel SGX is a relatively new cryptographic technology, and there have been some side-channel attacks published affecting SGX security (e.g., SgxPectre [12]) which should be addressed in any real-world systems depending on SGX.

SGX-Protected Proxy: The ability to delegate sensitive tasks to the trusted SGX enclave enables se-

3 A recent report from the U.N. explains that the average household size across 153 countries ranges from 2.1 to over 8 persons per household [62]; those users may have several devices that send and receive email and that are sporadically connected to the internet, and they may frequently host guests that use email.

4 The remote-attestation process requires trust in Intel's public-key infrastructure, in which Intel is the root of trust.

cure functionalities that otherwise would be difficult to achieve in a high-latency setting. The Raven client first needs to establish a shared secret with the proxy. The Raven client performs remote attestations with the enclave to receive proof that the enclave is running correctly and establishes a persistent symmetric key sk. (Note that all messages sent to/from the client are transferred through Raven). This process requires four round-trips between the Raven client and proxy (shown above the dashed line in Figure 6). Once a shared secret between the Raven client and proxy has been established, the client can send any number of messages (shown below the dashed line in Figure 6) without reestablishing the secret.

To perform a URL fetch operation, the client encrypts a URL under sk with an authenticated encryption scheme and sends it to the enclave. Then, the enclave decrypts the URL, *and while remaining inside the enclave*, establishes a TLS connection with the specified web host and fetches the resource. Crucially, all cryptographic operations occur within the enclave; the URL, GET request, and fetched resource are inaccessible and unmodifiable from outside the trusted enclave. (The untrusted host running the SGX proxy does learn the identity of the web host, as do Tor exit relays and all proxies fetching web resources in other proxy systems.) Once the resource has been fetched, the enclave encrypts it with sk and sends it to the client (over the Raven channel). The overall process is shown as a message sequence diagram in Figure 6.

4.4 Sketch of Raven's Security Properties

We briefly summarize Raven's security properties with respect to their effect on the protocol parties.

The Censor: The censor observes TLS connections between email clients and providers. The censor can prevent availability by uniformly blocking SMTPS and IMAPS connections, but will have difficultly selectively blocking only Raven users. Since Raven uses real email applications, it resists probing attacks.

The Provider: All communications between parties are GPG-encrypted using our simple PKI (the broker is a known entity with a global public key); this provides point-to-point secure channels between the client, broker, and proxies. The email provider can detect the presence of Raven messages, but cannot read the contents of messages or modify them. By assumption (see §4.2), there exists at least one provider that does not attempt to deny service to each protocol participant.

The Proxy: As stated above, we assume GPG provides a secure channel between the client and the proxy's untrusted operating system. The remote attestation process negotiates a secure channel between the client and the secure enclave running on the untrusted proxy. The proxy's untrusted components (and by extension, the broker—see below) can learn the network identity (IPv4 or IPv6 address) of the website contacted, but cannot view or modify the contents of TLS fetches made through the secure enclave.

The Broker: A malicious broker can redirect the client to any proxy that the broker desires. However, the SGX client will only use proxies that can perform the Intel SGX remote attestation process, which guarantees the client access to a secure enclave. The broker can deny service by ignoring client connection requests. However, in practice, we assume the broker will be run by a known trusted party (or parties), similar to Tor's directory authorities.

We remark that, because Raven is probe-resistant, behavior-realistic, and behavior-independent, the censor will be required to rely on behavior analysis and traffic analysis in order to attempt to distinguish Raven from genuine use of email on its networks. We evaluate the effectiveness of Raven against such analysis in §5.

5 Evaluation

In this section, we demonstrate the benefits of applying our behavior-based design principles through an in-depth evaluation of Raven. Note that our evaluation additionally includes a real world deployment of a prototype implementation of Raven, the details of which we present in Appendix B due to space constraints.

5.1 Data Methodology

To maintain covertness, Raven connections need to be modeled realistically after genuine email behavior. Producing such a model requires a dataset of email sending and receiving behavior. In a real world deployment, this data could be collected from a variety of sources, but the choice of the source of email meta-data should consider the ability for an adversary to poison the data in order to inject a signal that could later be used to detect the presence of Raven clients. Datasets that were created prior to Raven have a very low risk of being poisoned specifically against Raven. Similarly, there is little risk

of data poisoning for Raven users that provide their own personal email meta-data to privately train a behavior model locally before using Raven. There may be more risk associated with data provided by volunteers; this risk could be mitigated by accepting data only from trusted volunteers, or by forgoing volunteers and working directly with email providers to obtain meta-data.

For the purposes of evaluating Raven in this paper, we use two sources of email behavior: (1) email data collected from this paper's authors, and (2) archived email data, made available through the Linguistic Data Consortium[5], gathered from the "Avocado" IT Company.

Author Dataset: Our first dataset, the *Author* dataset, was collected voluntarily from the seven authors of this paper. We collected the timestamps and sizes of emails sent and received during the six year period starting on Jan. 01, 2014. In total, we collected (i) 66,327 sent emails with an average of 4 emails sent per volunteer-day and an average size of 173 KiB, and (ii) 605,250 received emails with an average of 40 emails received per volunteer-day and an average size of 88 KiB.

We use the email data from our limited set of 7 authors to simulate a more realistic scenario where many volunteers provide fewer data points. To do this, we organize each distinct week of genuine data from each author into a *user profile*; i.e., each user profile corresponds to the email behavior of a particular author during a single week. This process results in the creation of a total of 2,620 user profiles. Raven is evaluated on the basis of these profiles, explained later in this section.

Avocado Dataset: Although the Author dataset does contain a large volume of modern email behavior, it is limited due to the small number of authors whose data comprises the dataset. In order to supplement our data, we also ran our system evaluation on a second, older email dataset. The Avocado Research Email Collection (the *Avocado* dataset) is a collection of real users' emails from a now-defunct, pseudonymous "Avocado" IT company. "Avocado was an Information Technology software and services firm developing products for the mobile Internet market, operating from the late 1990s to the middle of the first decade of the 21st century" [52].

The dataset contains 279 cleaned and privatized[6] Microsoft Outlook mailbox files. We found that 273 of these mailbox files contained email files (several contained only contact files, for example). We assume that each Outlook mailbox file corresponds to a unique user (in the dataset documentation, it is said that "[m]ost of the accounts are those of Avocado employees"). However, unlike our author dataset, we did not know during which time periods each user was active and using email. In particular, accurately identifying periods of inactivity is challenging: for example, if no emails are present during a given time, we do not know if this is due to the user being inactive, or if the user was active but no emails were transmitted. To infer periods of activity, we look for sent emails as a sign that the user was active. On days when a user sent an email, we collect the user's incoming and outgoing email activity on that day, as well as the activity on the prior and next days.

In total, we collected (i) 145,320 sent emails with an average of 5.1 emails sent per employee-day and an average size of 85 KiB, and (ii) 481,401 received emails with an average of 17 emails received per employee-day and an average size of 53 KiB. The dataset contains 28,630 total days of activity for 218 users (profiles).

Features: We process the email data into a representation that is appropriate for the modeling task. We aggregate the email data into day-long periods of behavior and summarize the daily sending and receiving behavior with 12 hand-picked features: the volunteer ID, day of week, number of emails transmitted, and 5 percentiles (0th, 20th, 50th, 80th, and 100th) of the email size and transmission time distributions. These features have been shown to be highly informative when characterizing encrypted network traffic [7].

5.2 GAN Evaluation

CTGAN: We train a conditional tabular generative adversarial network (CTGAN) [75] to learn and imitate email transmission patterns. (In particular, we use the CopulaGAN model provided by the Synthetic Data Vault project[7].) The CTGAN is a state-of-the-art generative network that uses an efficient fully connected neural network to handle the tabular feature space associated with our user profiles.

Our selection of CTGANs over more traditional GANs or conditional GANs is due to two factors. First, although GANs have been found to be effective at generating fake images for the computer vision domain [13, 32, 47], they perform poorly when operating on lower dimensional, tabular data that is char-

5 https://www.ldc.upenn.edu/

6 See https://catalog.ldc.upenn.edu/docs/LDC2015T03/README.txt Section 5 for the procedures used to remove personally-identifiable information from the dataset.

7 https://sdv.dev/

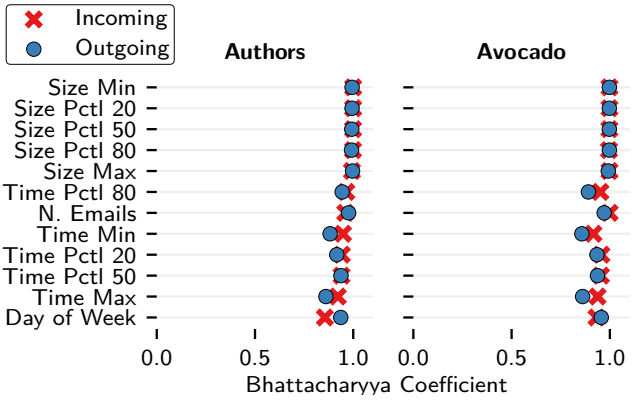

Fig. 7. The Bhattacharyya coefficient of samples drawn from the genuine and synthetic univariate distributions over features. The coefficient achieves a maximal value of 1.0 when the synthetic samples' density matches the genuine samples'. The left plot shows the results for the Author dataset and the right plot shows the Avocado dataset.

acteristic of our email dataset. Second, CTGANs are more amenable than traditional GANs for inputs that have a non-Gaussian multimodal distribution of continuous features and imbalanced discrete features. Our dataset contains data from different email users, each of whom exhibits email behavior that does not necessarily follow a Gaussian distribution. (More precisely, the distribution of their continuous feature values is often non-Gaussian.) CTGAN effectively detects continuous features and represents features from different distributions by applying mode-specific normalization. For discrete data, CTGAN learns via the training-by-sampling method.

Training and Evaluation: Using the CTGANs, we preserve the original dataset size while generating synthetic incoming email samples and outgoing email samples from the incoming and outgoing data points (i.e., days) in our datasets. Overall, the learned behavior model overall consists of four GANs: two GANs trained on the "number of emails" field in the incoming and outgoing direction, and two GANs trained on the email size/time distribution fields in the incoming and outgoing directions. We evaluate the output of the model after each GAN for 500 epochs, which we empirically determined was the point of diminishing returns.

To evaluate the quality of the training process, we take synthetic examples produced by the trained model and compare them to original points in the Author and Avocado datasets. More specifically, we look at the univariate distribution the generator learned for each numeric feature and compute the Bhattacharyya coefficient between this learned distribution and the original distribution [9]. The Bhattacharyya coefficient estimates the similarity of two samples by partitioning points into non-overlapping regions and comparing sample density in these regions. The coefficient equals 1.0 when the density is equal in all regions (i.e., when the samples are distributed equivalently). The coefficient equals 0.0 when the samples disagree entirely (i.e., when each region contains density from only one sample).

Results: Figure 7 shows the results of this comparison. The accuracy of the learned univariate distributions is high (greater than 0.8) in all features for both datasets, indicating that the distribution of outputs produced by the model is similar to the distribution of inputs when compared on a per-feature basis. Note that we do not analyze the accuracy of the multivariate distributions learned over related features, but we will quantify the overall quality of the synthetic distribution when evaluating Raven's security in §5.3. Additional results are provided in C, e.g., exploring the amount of time required to train the GAN and the effect of number of input features on the GAN's output. In general, we found that training the CTGAN model was relatively inexpensive (required fewer than 10 minutes) on a server configured with a GPU.

5.3 Security Evaluation

In this section, we evaluate Raven's susceptibility to traffic analysis. In particular, we estimate the degree to which Raven is able to achieve behavioral realism, because Raven's design already incorporates independence between the covert and cover channels (message patterns are determined by only the output of the GAN).

Similar to our study of Mailet (§3.1.3), we view the problem of identifying Raven traffic as a binary classification machine learning task. Our evaluation considers the scenario in which a network-layer adversary with white-box access to Raven attempts to identify Raven users from among genuine email users.

Features: Similar to the GAN training process, we treat each day of email sending and receiving as a single input example to the classifier. When performing classification in these experiments, we use an expanded 93-dimensional feature space—the day of week, the number of emails sent and received that day, 15×2 summary statistics[8] describing the size of emails sent and received, 15×2 summary statistics describing the times

8 The distribution's mean, standard deviation, skewness, kurtosis, min., max., and percentiles 10–90 (in steps of 10).

that emails are sent and received, and 15×2 summary statistics describing email interarrival times.

Classification: This classification experiment is designed to evaluate the degree to which the output from the GAN (i.e., the behavior of Raven users) is realistic among genuine email users *outside of the GAN training set*. (In contrast, in §5.2, we presented an evaluation of the extent to which the GAN is able to learn the distribution of behavior *within* the training set.)

In each trial, profiles are randomly shuffled evenly into two sets: (1) GAN-Train, and (2) Classify. For the profiles in the GAN-Train set, we train sending and receiving GANs, consistent with our methodology described in §5.2. Then, we sample 1,000 synthetic examples from the trained GAN and 1,000 examples from the profiles in the Classify set. We use this set of 2,000 examples for classification. We split this classification dataset into a training set with 1,000 examples and a testing set with 1,000 examples. The training set and testing set are balanced with an equal number of genuine and synthetic examples. We fit a random forest classifier (100 trees, `python's sklearn` implementation) on the training set. The classifier's goal is to correctly label examples as `genuine` or `synthetic` (i.e., Raven behavior). We then evaluate classifier performance on the examples in the test set. We repeated for 5 samples of the GAN-Train and Classify sets and for 10 random folds of the Train/Test splits, resulting in 50 total trials.

Random forest classifiers perform classification by training an ensemble of individual decision trees and having the individual decision trees vote on the correct classification. Classifier performance can be tuned by applying a threshold τ such that examples are considered `synthetic` only if the `synthetic` label receives at least τ votes. Applying a high threshold will limit the amount of false positive classifications made (`genuine` examples labeled as `synthetic`), but also decreases the recall of the classifier (the proportion of `synthetic` samples correctly detected).

Results: Figure 8 shows the receiver operating characteristic (ROC) curve of classifier performance. The ROC curve shows the trade-off between the false-positive rate (FPR, i.e., the fraction of genuine examples mislabeled *synthetic*) on the x-axis and true-positive rate (TPR, i.e., the fraction of synthetic examples labeled correctly) on the y-axis as the threshold τ is adjusted. The red and blue shaded regions show the minimum and maximum TPRs achieved at a given FPR across the trials for the Author and Avocado datasets, respectively. The green dotted line shows the baseline performance of a classifier that makes completely random guesses, which represents the worst possible performance of a classifier. Classification performance is optimal when the true-positive rate is high and the false-positive rate is low—this occurs when the ROC curve is in the top-left corner of the plot and the area under the curve (AUC) is close to 1.0.

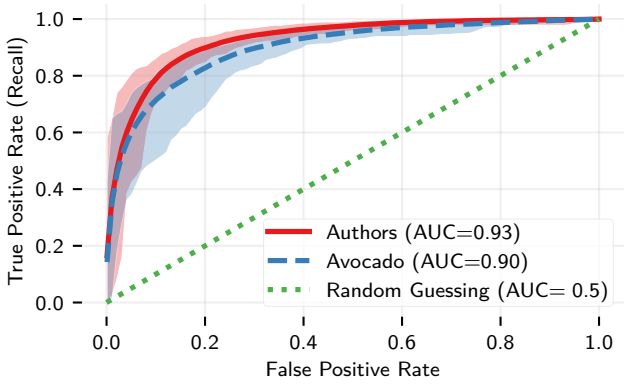

Fig. 8. ROC curves showing classification performance across different classification thresholds. The shaded regions show the minimum and maximum TPR achieved at a given FPR, and the bold lines show the average TPR vs. FPR curve over all trials.

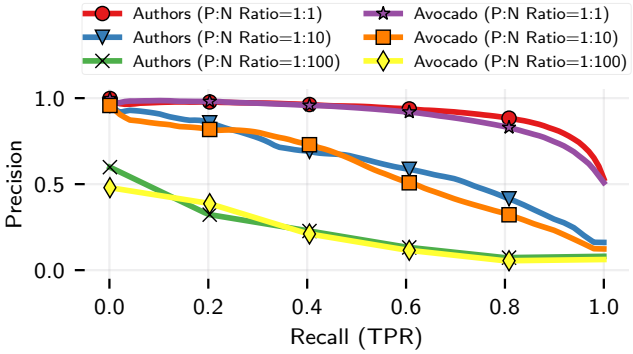

Fig. 9. Average classifier precision plotted versus classifier recall for different ratios of positive/`synthetic` (P) to negative/`genuine` (N) examples in the test set.

The random forest classifier is able to accurately identify some Raven usage: on average, the AUC is relatively high (0.93 and 0.90). However, for any high rate of recall (e.g., ≥ 0.8), the false positive rate is prohibitively expensive (e.g., ≥ 0.1). In our Mailet study, we found that Mailet usage could be detected with 100% recall at a FPR of only 3%. In contrast, to achieve 100% recall on Raven traffic, the classifier would incur an FPR of at least 50%. We believe these results are promising, as high false-positive rates correspond to high rates of collateral damage, especially when the incidence of Raven traffic is low relative to the rate of genuine email traffic.

To further explore the effect of false positives on classifier utility, we run an additional set of experiments. The setup is the same as previously described, except we vary the ratio of positive (`synthetic`) examples to negative (`genuine`) examples in the *test set only* (the classi-

fier was trained on a balanced training set). In addition to the 500-positive to 500-negative ratio we used previously, we ran experiments for 50-positive to 500-negative and 5-positive to 500-negative ratios. In these experiments, classifier precision and recall (TPR) is recorded. Precision is defined as $\frac{TP}{TP+FP}$, where TP is the number of true positives (correctly labeled `synthetic` examples) and FP is the number of false positives (`genuine` examples that were labeled as `synthetic`) and indicates the fraction of predictions that are correct from among all examples that the classifier labeled `synthetic`.

Figure 9 shows the results of this experiment. Each line in the graph shows classifier precision versus recall as the classifier threshold τ is adjusted, averaged across the 50 trials at a particular positive-negative ratios. *Notice that when negative examples largely outnumber positive examples (which is expected in realistic deployments), false-positive classifications erode classifier utility.* For example, even at just a 20% recall rate, the classifier achieves only 32% (Authors) or 39% (Avocado) precision (meaning $\geq$ 60% of `synthetic` label predictions are mis-labeled) at a 1% base rate. These results are encouraging, and suggest that Raven usage may be difficult to classify precisely at realistic base rates.

We refer an interested reader to Appendix A, which explores the effects of having the censor augmenting its classifier with additional data from an auxiliary dataset.

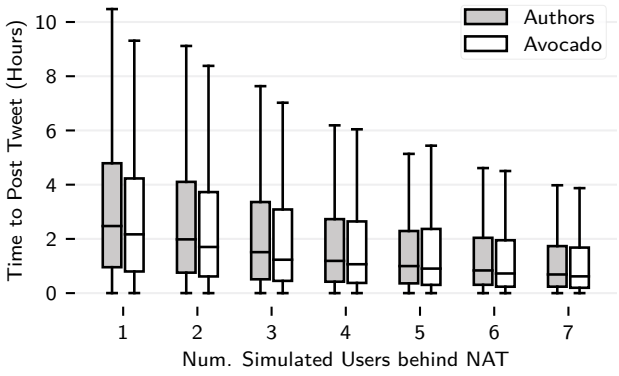

Fig. 10. Time required to post a Tweet, for various values of $|\mathcal{P}|$.

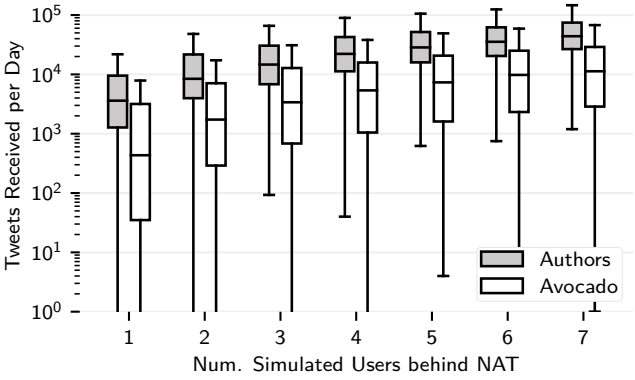

Fig. 11. The number of Tweets that can be retrieved per day (log-scale), for various values of $|\mathcal{P}|$.

5.4 Performance Analysis

To evaluate Raven's performance, we run simulations in which Raven users (i) post Tweets and (ii) access their Twitter feeds. We simulate the use of the SGX-protected proxy to securely delegate TLS connections, which are necessary for Twitter's REST-based API, and use the email profiles produced by the GAN as described in §5.2 to regulate the times and sizes of transmitted emails.

To determine the number of bytes required to post and retrieve Tweets, we use the official Twitter API and empirically measure the size of requests and responses. Our measurements conservatively use Tweets of 280 characters (the maximum length). We multiply required message sizes by 4/3 to adjust for the use of base64-encoding in our implementation of Raven (due to GPG; see Appendix B).

Raven allows the user to select a variable number of profiles to use, where each profile corresponds to a mimicked user (e.g., a user behind a NAT). Intuitively, the more users the Raven client mimics, the better the system's performance since there will be more opportunities to send and receive messages. We denote the profiles in use by the client as $\mathcal{P}$.

Using the GAN described in §5.2, we generate for each user profile in $\mathcal{P}$ ordered sets of incoming and outgoing *email descriptors*. Each email descriptor consists of a timestamp (the transmission time of the email) and the size of the email (excluding headers). We combine the sets of outgoing emails for all of the profiles in $\mathcal{P}$ and label it as OUT; we do the same for the incoming emails and label this combined set as IN. Finally, we order IN and OUT by time.

Tweeting or Sending a Direct Message: We simulate sending a Tweet (or equivalently, a direct message) as follows: we select a random day and choose a time uniformly at random from 9 a.m. to 5 p.m. on that day. We consider this to be the *decision time* (t_{dec}) in which a user opts to use Raven to send a Tweet. The simulator then determines the first email in OUT whose timestamp (t_{send}) is greater or equal to t_{dec}. This reflects the earliest opportunity for the user to send the Tweet (via email). The time required to Tweet is thus $t_{\text{send}} - t_{\text{dec}}$.

We vary the size of $|\mathcal{P}|$ from 1 to 7, and for each number of emulated users, we repeat the Tweeting experiment 2500 times. Figure 10 shows the time required to post a Tweet for various values of $|\mathcal{P}|$. Each box depicts the interquartile range and whiskers show the range of the data. When emulating one user ($|\mathcal{P}| = 1$), the median time to post the Tweet is 2.5 hours (Authors) and 2.2 hours (Avocado); emulating 7 users behind the NAT reduces the time to less than 45 minutes.

Retrieving Tweets: We simulate a Raven instance in which the SGX-protected proxy periodically retrieves Tweets (e.g., corresponding to the Raven user's Tweet feed) on behalf of the user, and then relays these Tweets via emails back to the user. We select a random day and consider all emails in IN—that is, all of the Raven-generated emails that the user receives that day. The number of bytes required to carry a maximum-length Tweet message is 635 B (847 B after base64-encoding), as determined empirically using the Twitter API. The number of Tweets retrieved per day is the sum of the body sizes of the messages in IN divided by 847.

The number of incoming Tweets that can be retrieved per day using Raven is shown in Figure 11. As before, we repeat each configuration 2,500 times and plot the interquartile ranges. The median number of retrieved Tweets per day is 3,606 (Authors) and 475 (Avocado) at the median when $|\mathcal{P}| = 1$. This increases by two orders of magnitude to approximately 44,000 (Authors) / 11,000 (Avocado) when $|\mathcal{P}| = 7$.

Summary: Our simulation-based performance evaluation shows that the performance of Tweeting is dominated by the time between deciding when to Tweet and having the GAN signify that it is safe to do so. This can incur delays measured in hours, but as we argue in §5.3, such delays may provide much stronger security protections than those of existing schemes that fail to model realistic human behavior. On the other hand, incoming email volumes are sufficient to retrieve a large number of Tweets per day, suggesting that Raven is especially well-suited for privately accessing Twitter feeds.

6 Conclusion

In this work, we argue the importance of *behavioral realism* and *behavioral independence*, two important properties that are often overlooked in the design of censorship circumvention systems. These properties provide protections against traffic analysis attacks aimed at revealing the presence of a covert channel within a cover protocol. However, covert channel designs satisfying these properties must cope with increased implementation costs and weak performance guarantees. To study how these properties affect real systems, we prototyped Raven, a covert channel designed to satisfy behavioral realism and independence. Raven was designed to be practical and follow best practices identified by the community. For instance, we use a ubiquitous cover channel that is accessible in many different regions, which we verified in a measurement study. Also, we tunnel traffic through real email client software, avoiding issues related to protocol mimicry. Our evaluations show that Raven significantly raises the false-positive rate during traffic analysis when compared to previous state-of-the-art systems, while securely providing acceptable application performance.

Future Work: Protocol emulation by itself is insufficient for covert communication because anomalous cover channel usage can produce traffic inconsistencies that are detectable by a censor. We recommend that future designs for censorship circumvention systems directly incorporate designs supporting behavioral independence and realism. Although we focused on email, these properties are general and can be applied in other channels. For example, our approach could be translated to emulate genuine user behavior for instant messaging, web browsing, or video conferencing. Porting behavior-based design principles to these and other cover channels is a promising direction for future work, as well as evaluating behavior-based designs against adversaries employing more sophisticated machine-learning models.

This paper is focused on evaluating email behavior and, like previous work, does not consider behavior across applications, devices, or networks. However, composed behavior could leak information about the presence of a covert channel on a user's network and should be considered when designing new anti-censorship techniques in future work. In Raven, for example, running multiple email profiles could lead to inconsistencies between email and non-email usage patterns; consistency could be improved by simultaneously running non-email applications along with Raven to better emulate the presence of additional users on the network.

When gathering data for training, capturing all important email patterns including weekends, holidays, and cultural differences (such as different dates for holidays in different countries), and constructing more diverse email datasets that include email dependencies (e.g., I only receive after I send) represent opportunities for future work.

7 Acknowledgements

This work is partially funded by the Office of Naval Research (ONR), the Defense Advanced Research Projects Agency (DARPA) (under Contract No. FA8750-19-C-0500), and the Georgetown University Callahan Family Professor Chair Fund. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

We would like to thank Aaron Johnson for reviewing this work and helping us reason about Raven's security. We would like to thank Diogo Barradas for help shepherding this work. Finally, we would like to thank Paul Syverson for reviewing this work and providing useful feedback.

References

[1] Enron Email Dataset. https://www.cs.cmu.edu/~./enron, 2015. Accessed: 2020-10-03.

[2] Mailet Source Code. https://github.com/magicle/Mailet, 2016. Accessed: 2022-03-10.

[3] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. nnovative Technology for CPU Based Attestation and Sealing. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, volume 13, 2013.

[4] Yawning Angel and Contributors. obfs4 Specification. https://github.com/Yawning/obfs4/blob/master/doc/obfs4-spec.txt, 2019. Accessed: 2021-05-31.

[5] Vaibhav Bajpai, Steffie Jacob Eravuchira, and Jürgen Schönwälder. Lessons Learned from using the RIPE Atlas Platform for Measurement Research. *ACM SIGCOMM Computer Communication Review*, 45(3):35–42, 2015.

[6] Diogo Barradas, Nuno Santos, and Luís E. T. Rodrigues. DeltaShaper: Enabling Unobservable Censorship-resistant TCP Tunneling over Videoconferencing Streams. *Proceedings on Privacy Enhancing Technologies Symposium*, 2017 (4):5–22, 2017. 10.1515/popets-2017-0037.

[7] Diogo Barradas, Nuno Santos, and Luís E. T. Rodrigues. Effective Detection of Multimedia Protocol Tunneling using Machine Learning. In *Proc. of the 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018*. USENIX Association, 2018.

[8] Diogo Barradas, Nuno Santos, Luís E. T. Rodrigues, and Vítor Nunes. Poking a Hole in the Wall: Efficient Censorship-Resistant Internet Communications by Parasitizing on WebRTC. In *Proc. of the 2020 ACM SIGSAC Conference on Computer and Communications Security, CCS 2020*. ACM, 2020.

[9] Anil Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[10] Cecylia Bocovich and Ian Goldberg. Slitheen: Perfectly Imitated Decoy Routing through Traffic Replacement. In *Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS 2016, Vienna, Austria, Oct. 24–28, 2016*. ACM. 10.1145/2976749.2978312.

[11] Chad Brubaker, Amir Houmansadr, and Vitaly Shmatikov. CloudTransport: Using Cloud Storage for Censorship-Resistant Networking. In *Proc. of the 14th Privacy Enhancing Technologies Symposium*, 2014.

[12] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *Proc. of the 2019 IEEE European Symposium on Security and Privacy*, 2019.

[13] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, 2016.

[14] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016.

[15] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing China's Blocking of Unpublished Tor Bridges. In *Proceedings of the 8th USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2018.

[16] Kevin P. Dyer, Scott E. Coull, Thomas Ristenpart, and Thomas Shrimpton. Protocol Misidentification Made Easy with Format-Transforming Encryption. In *Proc. of the 2013 ACM SIGSAC Conference on Computer and Communications Security, CCS 2013, Berlin, Germany, Nov. 4–8, 2013*. ACM, 2013. 10.1145/2508859.2516657.

[17] Kevin P. Dyer, Scott E. Coull, and Thomas Shrimpton. Marionette: A Programmable Network Traffic Obfuscation System. In *Proc. of the 24th USENIX Security Symposium, Washington, D.C., USA, Aug. 12–14, 2015*. USENIX Association, 2015.

[18] D. Ellard, C. Jones, V. Manfredi, W. T. Strayer, B. Thapa, M. Van Welie, and A. Jackson. Rebound: Decoy routing on asymmetric routes via error messages. In *2015 IEEE 40th Conference on Local Computer Networks (LCN)*, pages 91–99, 2015.

[19] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining How the Great Firewall Discovers Hidden Circumvention Servers. In *Proceedings of the 2015 ACM Internet Measurement Conference*, 2015.

[20] Serene Han et al. Snowflake Technical Overview. https://keroserene.net/snowflake/technical/, 2017. Accessed: 2021-05-31.

[21] David Fifield. *Threat modeling and circumvention of Internet censorship*. PhD thesis, University of California, Berkeley. URL https://www.bamsoftware.com/papers/thesis/.

[22] David Fifield, Nate Hardison, Jonathan Ellithorpe, Emily Stark, Dan Boneh, Roger Dingledine, and Phillip A. Porras. Evading Censorship with Browser-Based Proxies. In *Proceedings of the 2012 Privacy Enhancing Technologies Symposium*, 2012.

[23] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. *Proceedings on Privacy Enhancing Tech-*

nologies, 2015, 2015.

[24] Freedom House. Freedom House. https://freedomhouse.org. Accessed Feb. 07, 2022.

[25] Sergey Frolov and Eric Wustrow. The use of TLS in Censorship Circumvention. In *Proc. of the 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, Feb. 24–27, 2019*. The Internet Society, 2019. 10.14722/ndss.2019.23511.

[26] Sergey Frolov and Eric Wustrow. HTTPT: A Probe-Resistant Proxy. In *10th USENIX Workshop on Free and Open Communications on the Internet, FOCI*. USENIX Association, 2020.

[27] Sergey Frolov, Frederick Douglas, Will Scott, Allison McDonald, Benjamin VanderSloot, Rod Hynes, Adam Kruger, Michalis Kallitsis, David G. Robinson, Steve Schultze, Nikita Borisov, J. Alex Halderman, and Eric Wustrow. An ISP-Scale Deployment of TapDance. In *Proc. of the 7th USENIX Workshop on Free and Open Communications on the Internet, FOCI 2017, Vancouver, BC, Canada, Aug. 14, 2017*. USENIX Association, 2017.

[28] Sergey Frolov, Jack Wampler, Sze Chuen Tan, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Conjure: Summoning Proxies from Unused Address Space. In *Proc. of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, Nov. 11–15, 2019*. ACM, 2019. 10.1145/3319535.3363218.

[29] Sergey Frolov, Jack Wampler, and Eric Wustrow. Detecting Probe-resistant Proxies. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium*, 2020.

[30] Game of Thrones Wiki. "Game of Thrones Wiki: Raven". https://gameofthrones.fandom.com/wiki/Raven. Accessed Feb. 07, 2022.

[31] John Geddes, Max Schuchard, and Nicholas Hopper. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, 2013.

[32] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, 2014.

[33] Bridger Hahn, Rishab Nithyanand, Phillipa Gill, and Rob Johnson. Games without Frontiers: Investigating Video Games as a Covert Channel. In *IEEE European Symposium on Security and Privacy, EuroS&P*. IEEE, 2016.

[34] John Holowczak and Amir Houmansadr. CacheBrowser: Bypassing Chinese Censorship without Proxies Using Cached Content. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[35] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention Infrastructure using Router Redirection with Plausible Deniability. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, 2011.

[36] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. The Parrot Is Dead: Observing Unobservable Network Communications. In *Proc. of the 2013 IEEE Symposium on Security and Privacy, S&P 2013, Berkeley, CA, USA, May 19-22, 2013*. IEEE Computer Society, 2013. 10.1109/SP.2013.14.

[37] Amir Houmansadr, Thomas J. Riedl, Nikita Borisov, and Andrew C. Singer. I want my voice to be heard: IP over Voice-over-IP for unobservable censorship circumvention. In *20th Annual Network and Distributed System Security Symposium, NDSS 2013, San Diego, California, USA, February 24-27, 2013*. The Internet Society, 2013.

[38] Amir Houmansadr, Wenxuan Zhou, Matthew Caesar, and Nikita Borisov. SWEET: Serving the Web by Exploiting Email Tunnels. *IEEE/ACM Transactions on Networking*, 25 (3):1517–1527, 2017. 10.1109/TNET.2016.2640238.

[39] Josh Karlin, Daniel Ellard, Alden W Jackson, Christine E Jones, Greg Lauer, David Mankins, and W Timothy Strayer. Decoy Routing: Toward Unblockable Internet Communication. In *USENIX Workshop on Free and Open Communications on the Internet, FOCI '11, San Francisco, CA, USA, August 8, 2011*. USENIX Association, 2011.

[40] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*. IEEE, 2020.

[41] Sheharbano Khattak, Tariq Elahi, Laurent Simon, Colleen M. Swanson, Steven J. Murdoch, and Ian Goldberg. SoK: Making Sense of Censorship Resistance Systems. *Proceedings on Privacy Enhancing Technologies Symposium*, 2016(4), 2016. 10.1515/popets-2016-0028.

[42] Shuai Li and Nicholas Hopper. Mailet: Instant Social Networking under Censorship. *Proceedings on Privacy Enhancing Technologies Symposium*, 2016(2), 2016.

[43] Shuai Li, Mike Schliep, and Nick Hopper. Facet: Streaming over Videoconferencing for Censorship Circumvention. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, 2014.

[44] Zhen Ling, Junzhou Luo, Wei Yu, Ming Yang, and Xinwen Fu. Extensive Analysis and Large-Scale Empirical Evaluation of Tor Bridge Discovery. In *Proc. of the IEEE INFOCOM 2012, Orlando, FL, USA, Mar. 25–30, 2012*. IEEE, 2012. 10.1109/INFCOM.2012.6195627.

[45] Srdjan Matic, Carmela Troncoso, and Juan Caballero. Dissecting Tor Bridges: a Security Evaluation of Their Private and Public Infrastructures. In *Proc. of the 24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, Feb. 26–Mar. 01, 2017*. The Internet Society, 2017. 10.14722/ndss.2017.23345.

[46] Richard McPherson, Amir Houmansadr, and Vitaly Shmatikov. CovertCast: Using Live Streaming to Evade Internet Censorship. *Proceedings on Privacy Enhancing Technologies*, 2016.

[47] Mehdi Mirza and Simon Osindero. Conditional Generative Adversarial Nets. *CoRR*, abs/1411.1784, 2014. URL http://arxiv.org/abs/1411.1784.

[48] Hooman Mohajeri Moghaddam, Baiyu Li, Mohammad Derakhshani, and Ian Goldberg. SkypeMorph: Protocol Obfuscation for Tor Bridges. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[49] Milad Nasr, Hadi Zolfaghari, and Amir Houmansadr. The Waterfall of Liberty: Decoy Routing Circumvention that Resists Routing Attacks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications*

Security, 2017.

[50] Milad Nasr, Hadi Zolfaghari, Amir Houmansadr, and Amirhossein Ghafari. MassBrowser: Unblocking the Censored Web for the Masses, by the Masses. In *Proceedings of the 27th Annual Network and Distributed System Security Symposium*, 2020.

[51] Arian Akhavan Niaki, Shinyoung Cho, Zachary Weinberg, Nguyen Phong Hoang, Abbas Razaghpanah, Nicolas Christin, and Phillipa Gill. ICLab: A Global, Longitudinal Internet Censorship Measurement Platform. In *IEEE Symposium on Security and Privacy (SP)*, May 2020.

[52] Douglas Oard, William Webber, David A. Kirsch, and Sergey Golitsynskiy. Avocado Research Email Collection. Web Download, 2015. https://catalog.ldc.upenn.edu/LDC2015T03.

[53] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the Deployment of Network Censorship Filters at Global Scale. In *Proc. of the 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, Feb. 23–26, 2020*. The Internet Society, 2020. 10.14722/ndss.2020.23099.

[54] Maria Rigaki and Sebastian Garcia. Bringing a GAN to a Knife-Fight: Adapting Malware Communication to Avoid Detection. In *Proceedings of 2018 IEEE Security and Privacy Workshops*, 2018.

[55] RIPE Network Coordination Centre. RIPE Atlas. https://atlas.ripe.net/, 2020.

[56] Riseup. riseup.net. https://riseup.net/, 2021.

[57] Max Schuchard, John Geddes, Christopher Thompson, and Nicholas Hopper. Routing Around Decoys. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[58] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *26th USENIX Security Symposium, USENIX Security 2017*, 2017.

[59] Rima S. Tanash, Zhouhan Chen, Tanmay Thakur, Dan S. Wallach, and Devika Subramanian. Known Unknowns: An Analysis of Twitter Censorship in Turkey. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society, WPES 2015, Denver, Colorado, USA, October 12, 2015*. ACM, 2015. 10.1145/2808138.2808147.

[60] Telegram from Russia: compliance and complicity in the Russian government's attack on privacy. "Telegram from Russia: compliance and complicity in the Russian government's attack on privacy". https://privacyinternational.org/long-read/2026/telegram-russia-compliance-and-complicity-russian-governments-attack-privacy. Publication date May 14, 2018.

[61] Michael Carl Tschantz, Sadia Afroz, anonymous, and Vern Paxson. SoK: Towards Grounding Censorship Circumvention in Empiricism. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*. IEEE Computer Society, 2016. 10.1109/SP.2016.59.

[62] United Nations, Department of Economic and Social Affairs, Population Division. Patterns and trends in household size and composition: Evidence from a United Nations dataset. Technical Report ST/ESA/SER.A/433, 2019.

[63] Benjamin VanderSloot, Sergey Frolov, Jack Wampler, Sze Chuen Tan, Irv Simpson, Michalis Kallitsis, J. Alex Halderman, Nikita Borisov, and Eric Wustrow. Running Refraction Networking for Real. *Proceedings on Privacy Enhancing Technologies Symposium*, 2020(4):321–335, 2020. 10.2478/popets-2020-0075.

[64] Paul Vines and Tadayoshi Kohno. Rook: Using Video Games as a Low-Bandwidth Censorship Resistant Communication Platform. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society, WPES*. ACM, 2015.

[65] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. Seeing through Network-Protocol Obfuscation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[66] Qiyan Wang, Xun Gong, Giang T. K. Nguyen, Amir Houmansadr, and Nikita Borisov. CensorSpoofer: Asymmetric Communication using IP Spoofing for Censorship-Resistant Web Browsing. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, 2012.

[67] Yazhe Wang, Jamie Callan, and Baihua Zheng. Should We Use the Sample? Analyzing Datasets Sampled from Twitter's Stream API. *ACM Transactions on the Web (TWEB)*, 9, 2015.

[68] Zachary Weinberg, Jeffrey Wang, Vinod Yegneswaran, Linda Briesemeister, Steven Cheung, Frank Wang, and Dan Boneh. StegoTorus: A Camouflage Proxy for the Tor Anonymity System. In *Proc. of the ACM Conference on Computer and Communications Security, CCS 2012, Raleigh, NC, USA, Oct. 16–18, 2012*. ACM, 2012. 10.1145/2382196.2382211.

[69] Brandon Wiley. Dust: A blocking-resistant internet transport protocol. Technical report, University of Texas at Austin, 2011.

[70] Philipp Winter, Tobias Pulls, and Jürgen Fuß. Scramblesuit: A polymorphic network protocol to circumvent censorship. In *Proceedings of the 12th Annual ACM Workshop on Privacy in the Electronic Society*, 2013.

[71] wolfSSL Inc. wolfSSL TLS Library. https://www.wolfssl.com/, 2021.

[72] Charles V. Wright, Scott E. Coull, and Fabian Monrose. Traffic Morphing: An Efficient Defense Against Statistical Traffic Analysis. In *Proceedings of the Network and Distributed System Security Symposium, NDSS*. The Internet Society, 2009.

[73] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the Network Infrastructure. In *Proc. of the 20th USENIX Security Symposium, San Francisco, CA, USA, Aug. 8–12, 2011*. USENIX Association, 2011.

[74] Eric Wustrow, Colleen Swanson, and J. Alex Halderman. TapDance: End-to-Middle Anticensorship without Flow Blocking. In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, Aug. 20–22, 2014*. USENIX Association, 2014.

[75] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling Tabular data using Conditional GAN. In *Advances in Neural Information Processing Systems 32 (NeurIPS 2019)*, 2019.

[76] Diwen Xue, Reethika Ramesh, Valdik S. S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. Throttling Twitter: An Emerging Censorship Technique in Russia. In *Proc. of the ACM Internet Measurement Conference (IMC'21), Virtual Event, USA, November 2–4, 2021*. ACM, 2021. 10.1145/3487552.3487858.

[77] Hadi Zolfaghari and Amir Houmansadr. Practical Censorship Evasion Leveraging Content Delivery Networks. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

A Security Evaluation: Training Dataset Augmentation

In this Appendix, we provide supplementary security evaluation results to those presented in Section 5.3. In particular, we replicate the classification experiment on the Author dataset described in Section 5, but with one modification: instead of the adversary training on outputs from the behavior model trained on profiles in the GAN-Train set, we give the adversary a behavior model trained on *both* GAN-Train profile data *and* Avocado profile data (see Appendix 5.1), which can considered to be an auxiliary dataset. Otherwise, the rest of the experiment is the same—the adversary uses the trained classifier to separate instances of behavior generated by the Raven behavior model (trained on only GAN-Train profile data) from instances of user behavior sampled from the Classify set.

Figure 12 shows the result of this experiment. We find that adding auxiliary data to the adversary's behavior model degrades classification performance. This is because the adversary is given an instance of the exact model assumed to be used by Raven clients in the Section 5.3 experiments, which is the most advantageous setting for the adversary. By adding differently-distributed data to the adversary's model, it becomes more difficult to learn the distinctions of Raven-produced email behavior from genuine behavior.

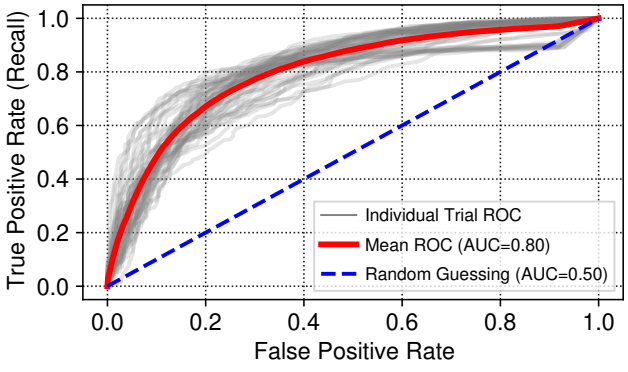

Fig. 12. (ROC) curves showing classification performance across different classification thresholds for the classifier using an augmented training dataset.

B Software Details

To test the real-world efficacy of Raven, we built a prototype implementation of Raven.

Implementation: Our implementation of the Raven client consists of Python controllers, the CTGAN [75], SGX client code, an unmodified Thunderbird version 75, and a custom extension to Thunderbird. To make Raven accessible to a wide variety of platforms, we packaged the Raven client as a Docker container, using the X virtual framebuffer (Xvfb) display server to support non-GUI modes of operation. The broker-side components of Raven include Python controllers, CTGAN, and the SGX enclave code. We will release all Raven components as free and open software concurrent with publication.

We use GPG 2.2 with elliptical curve Diffie-Hellman Curve25519 keys to encrypt Raven messages before they are written to the body of an email. We apply padding to the plaintext (before GPG-encryption) to achieve the desired email size, as determined by either the client's or the proxy's GAN (depending upon the direction of traffic). The encrypted messages from the Raven client include a profile identifier that allows the proxy to select the matching GAN profile.

On the client side, the user configures Raven with the broker's email address and public key. Raven operates a local controller daemon that allows multiple network applications to use Raven while ensuring that communication never deviates from the client's GAN profile. To communicate over Raven, an application opens a connection to the Raven controller and sends and receives network communication using a standard socket interface. The controller multiplexes applications' requests and relays their communications in accordance with the schedule produced by the GAN.

The SGX proxy component is based on the WolfSSL library [71] and supports TLS 1.3. Although certificate validation is performed by the broker, this occurs within the SGX enclave, and SGX attestation provides assurances to the client that both the broker-side software and the root certificates have not been modified.

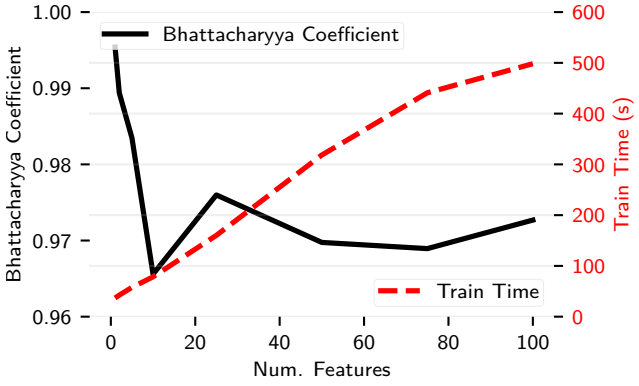

Fig. 13. Average Bhattacharyya coefficient and train time versus the number of features input to the CopulaGAN.

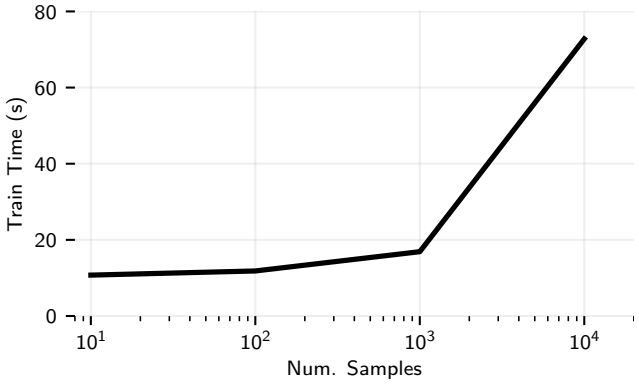

Fig. 14. Train time versus the number of training samples input to the CopulaGAN.

Table 2. Success rates for establishing secure TLS connections with IMAPS and SMTPS servers from various locations. "Local" denotes a single-user email server located in the US.

Country	Gmail	Yahoo!	Riseup	Local
Multiple Countries†	1.00	1.00	1.00	1.00
Lebanon & United Kingdom	1.00	1.00	0.90	1.00
South Africa & Turkey	1.00	0.90	1.00	1.00
Czech Republic	1.00	0.80	0.90	1.00
Sri Lanka	0.83	1.00	1.00	1.00
Rwanda	1.00	1.00	0.67	1.00
Peru	1.00	1.00	0.83	0.83
Hungary	0.90	0.90	1.00	0.90
Ecuador	0.89	1.00	0.89	0.89
Serbia	0.84	0.89	0.89	0.89
Korea, Republic of	0.94	0.94	0.94	0.94
Russian Federation	1.00	1.00	1.00	0.90
Latvia	0.94	0.91	0.91	0.94
Panama	0.80	0.80	0.80	1.00
Armenia	0.89	0.89	0.89	0.89
Morocco	0.67	0.67	1.00	1.00
Saudi Arabia	0.88	1.00	0.88	1.00
Uzbekistan	1.00	1.00	0.83	1.00
Sudan	0.67	0.67	0.67	0.67
Belize	0.75	0.75	0.75	0.75
Paraguay	0.71	0.57	0.71	0.71
Kyrgyzstan & Uganda	0.50	0.50	0.50	0.50
Iran, Islamic Republic of	1.00	0.89	1.00	0.90
China	0.45	0.85	0.18	0.82

† Domin. Repub., Burundi, Ethiopia, Azerbaijan, Egypt, Angola

C GAN Training Performance

In this appendix, we share additional results about the process of training the CopulaGAN.

In Figure 13, the average Bhattacharyya coefficient computed over features is shown as the number of features in the training set is varied. In this experiment, each feature's distribution is chosen to be normal with random mean and variance. Note that the minimum value plotted is ≥ 0.96, indicating that, for any number of features, the GAN is accurately reproducing the input distribution. However, the training process becomes more expensive as the number of features is increased. The red dashed line plots training time, which increases approximately linearly with the number of features input to the GAN. The training time measurements were performed on an AMD EPYC 7551 32-Core Processor and accelerated with a Nvidia Quadro RTX 8000 GPU. (The GAN was configured to train for 100 epochs and was given 10,000 samples of each feature).

Figure 14 shows how training time is affected by the number of input samples. In this plot, the number of features is fixed to ten, and the GAN is trained for 100 epochs. Note that the x-axis is log-scale; overall, our results show that the GAN's training time is not very sensitive to the number of samples input to the GAN.

D Email Measurement

This appendix gives the full results from our email accessibility measurement, described in §4.1. The full table of results is shown in Table 2.