# Compressive Traffic Analysis: A New Paradigm for Scalable Traffic Analysis

Milad Nasr
University of Massachusetts Amherst
milad@cs.umass.edu

Amir Houmansadr
University of Massachusetts Amherst
amir@cs.umass.edu

Arya Mazumdar
University of Massachusetts Amherst
arya@cs.umass.edu

## ABSTRACT

Traffic analysis is the practice of inferring sensitive information from communication patterns, particularly packet timings and packet sizes. Traffic analysis is increasingly becoming relevant to security and privacy with the growing use of encryption and other evasion techniques that render content-based analysis of network traffic impossible. The literature has investigated traffic analysis for various application scenarios, from tracking stepping stone cybercriminals to compromising anonymity systems.

The major challenge to existing traffic analysis mechanisms is scaling to today's exploding volumes of network traffic, i.e., they impose high storage, communications, and computation overheads. In this paper, we aim at addressing this scalability issue by introducing a new direction for traffic analysis, which we call *compressive traffic analysis*. The core idea of compressive traffic analysis is to compress traffic features, and perform traffic analysis operations on such compressed features instead of on raw traffic features (therefore, improving the storage, communications, and computation overheads of traffic analysis due to using smaller numbers of features). To compress traffic features, compressive traffic analysis leverages linear projection algorithms from compressed sensing, an active area within signal processing. We show that these algorithms offer unique properties that enable compressing network traffic features while preserving the performance of traffic analysis compared to traditional mechanisms.

We introduce the idea of compressive traffic analysis as a new generic framework for scalable traffic analysis. We then apply compressive traffic analysis to two widely studied classes of traffic analysis, namely, flow correlation and website fingerprinting. We show that the compressive versions of state-of-the-art flow correlation and website fingerprinting schemes—significantly—outperform their non-compressive (traditional) alternatives, e.g., the compressive version of Houmansadr et al. [44]'s flow correlation is two orders of magnitude faster, and the compressive version of Wang et al. [77] fingerprinting system runs about 13 times faster. We believe that our study is a major step towards scaling traffic analysis.


*CCS '17, October 30-November 3, 2017, Dallas, TX, USA*

ACM ISBN 978-1-4503-4946-8/17/10…$15.00
https://doi.org/10.1145/3133956.3134074

## CCS CONCEPTS

• **Security and privacy** → *Pseudonymity, anonymity and untraceability*; *Privacy-preserving protocols*;

## KEYWORDS

Traffic analysis; compressed sensing; website fingerprinting; flow correlation

## 1 INTRODUCTION

Traffic analysis is the art of inferring sensitive information from communication *patterns*, particularly packet timings and packet sizes. Traffic analysis is becoming increasingly more relevant to security and privacy with the surging use of encryption and other evasion techniques that render content-based analysis of traffic infeasible. For instance, stepping stone relays [73, 88] re-encrypt packet payloads and modify packet headers to prevent matching of packets based on content. Also, the flows comprising a Tor connection (e.g., between various Tor relays) can not be correlated by content matching as each of these flows are encrypted with a different key. As another example, the use of VPNs conceals the contents of the underlying network packets.

Researchers have investigated the use of traffic analysis in various application scenarios to either *defend* or *attack* the security and privacy of networked systems. On one hand, various traffic analysis techniques have been designed to identify cybercriminals (such as botmasters) who proxy their attack traffic through compromised machines or public relays in order to conceal their identities [40, 65, 73, 88, 90]. On the other hand, researchers have demonstrated various traffic analysis techniques that enable adversaries to demote online privacy [3, 24, 43, 64, 79, 80, 92], e.g., by compromising anonymity systems like Tor [28] and mix networks [25, 67, 68] through flow correlation [64, 80] or website fingerprinting [12, 62, 77] attacks.

The major challenge to traffic analysis is *scaling* to today's gigantic volumes of network traffic. First, traffic analysis parties need to *store* extremely large volumes of collected traffic characteristics. For instance, the border routers of an ISP who intends to identify stepping stone attacks [73, 88] need to collect and store traffic characteristics of all of the flows that they intercept. Second, in many applications, traffic analysis involves the *transmission* of the collected traffic characteristics between multiple traffic analysis parties in real-time. For instance, flow correlation attacks on Tor are conducted by adversaries who communicate among themselves the traffic features that they collect at different points of the Tor network (e.g., on multiple compromised Tor relays). Finally, traffic analysis systems need to run their correlation algorithms (e.g., statistical correlation [43] or machine learning algorithms [52, 77]) on

the—extremely large—database of collected traffic characteristics, imposing high *computation* overheads. In summary, existing traffic analysis mechanisms suffer from enormous **storage, communications, and computation overheads** due to the overwhelming network traffic volumes.

The goal of this work is to improve the scalability of traffic analysis. We introduce a new direction to traffic analysis, which we call *compressive traffic analysis*. Our approach is inspired by an active research area in signal processing, called *compressed sensing* [9, 19, 22, 69, 74, 87], which aims at performing efficient signal acquisition. The core idea of compressive traffic analysis is to use the *linear projection algorithms* used by compressed sensing systems to *compress* the traffic features used in traffic analysis. Traffic analysis operations are then performed on such compressed traffic features—instead of on raw traffic features which is done traditionally—therefore improving storage, communications, and computation overheads, i.e., improving scalability. Note that we call an algorithm `A` to be more scalable than `B` if either (1) when both algorithms offer the same traffic analysis performance (e.g., same fingerprinting accuracy), `A` has less storage, communications, and computation overheads (e.g., it is faster by using fewer features for fingerprinting), or (2) for the same storage, communications, and computation overheads, `A` provides better traffic analysis performance (e.g., higher fingerprinting accuracy). We will use these two equivalent notions interchangeably throughout the paper.

Compressive traffic analysis is possible due to two **unique properties** of compressed sensing's linear projection algorithms: (1) The linear projection algorithms used in compressed sensing are designed to work best on *sparse* data vectors [19, 47]. Fortunately, traffic features used in traffic analysis (particularly, packet timings and packet sizes) are sparse signals, making them a natural target for such sensing algorithms. (2) Because of the *restricted isometry property (RIP)* [31] of compressed sensing algorithms, traffic features preserve their Euclidean distances after compression by linear projection algorithms. This enables us to perform traffic analysis on compressed (sparse) features, instead of raw features, without significantly degrading the performance of traffic analysis. Due to this property, compressive traffic analysis does not need to reconstruct the compressed traffic features, and can perform analysis directly on the compressed traffic features (therefore, avoiding the big computation overhead of reconstruction).

We present the idea of compressive traffic analysis as a *generic* new direction towards scaling various types of traffic analysis algorithms. To demonstrate the applicability of compressive traffic analysis, we investigate it for two widely-studied classes of traffic analysis, namely, *flow correlation* [24, 43, 72] and *website fingerprinting* [49, 62, 78]. We design compressive algorithms for flow correlation and website fingerprinting, and compare their performance to their non-compressive (traditional) alternatives. Our extensive experiments demonstrate the significant scalability improvements of compressive traffic analysis over traditional traffic analysis. For instance, through experiments on Tor [28] we show that our compressive flow correlation offers a true positive correlation rate of $\approx 0.9$, while its non-compressive alternative (Houmansadr et al. [44]) only offers a $\approx 0.3$ true positive rate—when both of the algorithms offer the same false positive rate and have the same overheads; alternatively, our compressive flow correlation is—two orders of magnitude faster—for the same correlation performance. As another example, our compressive website fingerprinting algorithm is—13 times faster—than its non-compressive alternative (Wang et al. [77]) for the same fingerprinting accuracy.

To summarize, we make the following main contributions:

- We introduce and formulate the novel idea of compressive traffic analysis, which aims at making traffic analysis more scalable by leveraging recent advances in signal processing.
- We introduce *compressive flow correlation* by applying compressive traffic analysis to traditional (state-of-the-art) flow correlation schemes. Through extensive experimentation and simulations of network traffic on the Internet and Tor network [28] we show that our compressive flow correlation schemes significantly improve scalability compared to their traditional, non-compressive alternatives.
- We introduce *compressive website fingerprinting* by applying compressive traffic analysis to major state-of-the-art approaches for website fingerprinting (i.e., $k$-NN and SVM based approaches). We demonstrate through comprehensive simulations that our compressive website fingerprinting systems significantly improve scalability compared to their non-compressive alternatives.

The rest of this paper is organized as follows: In Section 2 we overview traffic analysis. We introduce and formulate the idea of compressive traffic analysis in Section 3. We introduce and design compressive flow correlation systems in Section 4, and evaluate them in Section 5 through experiments and simulations. We also introduce and design compressive website fingerprinting systems in Section 6 and evaluate their performance in Section 7. We conclude the paper in Section 8.

## 2 BACKGROUND: TRAFFIC ANALYSIS

Traffic analysis is inferring sensitive information from communication *characteristics*,[1] particularly packet sizes, packet timings, and their derivatives like packet rates and inter-packet delays. Traffic analysis is particularly useful in scenarios where encryption and other content evasion techniques such as content obfuscation do not allow one to inspect the contents of communications. In the following, we review popular types of traffic analysis and discuss their applications.

**Flow correlation** Flow correlation is used to *link network flows* in the presence of encryption and other content obfuscation mechanisms. On one hand, flow correlation is used to link network flows in order to identify and stop cybercriminals who use network proxies to obfuscate their identities [73, 88, 90], i.e., stepping stone attackers. On the other hand, flow correlation is known to be usable by adversaries to compromise privacy in anonymity systems like Tor [28] and mix networks [25, 67, 68] by linking the traffic features of egress and ingress flows [3, 24, 60, 65, 72, 79, 80, 92].

Flow correlation links network flows by evaluating traffic features that do not significantly change by content obfuscation mechanisms. Most flow correlation systems use packet timing characteristics [24, 30, 36, 72, 82, 90] (or derivative features like packet

[1] In this paper, we use the terms flow "characteristics", "patterns", and "features" interchangeably.

counts, packet rates, and inter-packet delays), and some use packet sizes [53]. For instance, Wang et al. [82] cross-correlated the inter-packet delays of network flows. He and Tong use packet counts [36] to correlate network flows, and Paxson and Zhang [90] model packet arrivals as a series of ON and OFF patterns, which they use to correlate network flows.

**Flow watermarking** Network flow watermarking is an *active* variant of the flow correlation mechanism introduced above. Similar to flow correlation schemes, flow watermarking also aims at linking network flows in application scenarios similar to those of flow correlation, e.g., stepping stone detection [73]. However, flow watermarking systems *perturb* traffic features, e.g., packet timings and sizes, before attempting to correlate them across network flows. In particular, many flow watermarking systems [39, 41, 43, 64, 80, 89] perturb timing characteristics by slightly delaying network packets in a way to modulate an artificial pattern, called the watermark. For instance, RAINBOW [43] manipulates the inter-packet delays of packets in order to embed watermark signals. Several proposals [64, 80, 89], known as interval-based watermarks, work by delaying packets into specific, secret time intervals.

**Website fingerprinting** Website fingerprinting aims at detecting the websites (or webpages) visited over encrypted channels like VPNs, Tor, and other proxies [12, 33, 35, 37, 50, 56, 62, 77, 78]. The attack is performed by a passive adversary who monitors the victim's encrypted network traffic, e.g., a malicious ISP or a surveillance agency. The adversary compares the victim's observed traffic patterns against a set of prerecorded webpage traces, called fingerprints, to identify the webpage being browsed. Website fingerprinting is different from flow correlation and flow watermarking in that the adversary only observes one end of the connection, i.e., the target user's flow, but not the traffic going to the destination website. Website fingerprinting has particularly been widely studied in the context of Tor traffic analysis [35, 49, 62, 78].

Similar to the other classes of traffic analysis discussed above, website fingerprinting also uses traffic features that are *not* much impacted by encryption. Website fingerprinting mechanisms commonly use traffic features like packet timings, packet directions, and packet sizes. Most website fingerprinting mechanisms leverage *machine learning* algorithms to implement their fingerprinting attacks [12, 33, 35, 37, 50, 52, 56, 62, 77, 78]. In particular, the state-of-the-art website fingerprinting mechanisms use one of the two machine learning algorithms of *Support Vector Machines (SVM)* [12, 62, 78] and *k-Nearest Neighbor (k-NN)* [33, 35, 77].

**Other types of traffic analysis** Traffic analysis has also been applied to other application scenarios in order to disclose sensitive information. Particularly, various types of side channel attacks are built on traffic analysis. For instance, multiple studies use traffic analysis to uncover not only the language spoken over encrypted VoIP connections [86], but also the spoken phrases [84, 85]. Chen et al. [20] demonstrate a traffic analysis-based side channel attack on critical web services like online financial services. Schuster et al. [70] use traffic analysis to infer the video streams being watched over encrypted channels. Geddes et al. [32], Houmansadr et al. [42], and Wang et al [76] show that repressive governments can identify and block encrypted circumvention traffic through various types of traffic analysis attacks.

# 3 OVERVIEW OF THE CORE IDEAS

We start by introducing the idea of compressed sensing. We then describe how we use linear projection algorithms from compressed sensing to design compressive traffic analysis mechanisms.

## 3.1 Compressed Sensing

Compressed sensing is an actively studied approach for signal acquisition and compression that aims at *efficient* reconstruction of signals from few linear projections [6, 9, 15–17, 19, 22, 29, 69, 74, 87]. Compressed sensing is built on the principle that the 'sparsity' of a signal can be exploited to recover it from *far fewer samples* than what is required by the Shannon-Nyquist sampling theorem [6, 22, 69, 87].

A sparse signal [6, 19] is one that contains many coefficients close to or equal to zero in *some* domain of representation. For instance, digital images have many zero or near-zero components when represented in the Wavelet domain [19]. Compressed sensing works by sampling a sparse signal in a basis that can be even different from the basis in which the signal is known to be sparse. For instance, an image can be sampled in the spatial domain even though it is sparse in the Wavelet domain (but not in the spatial domain). The compressive samples are the weighted linear combinations of a signal's components. Suppose that $X_{N\times1}$ is a signal vector of size $N$. The compressive measurement is derived as

$$Y_{M\times1} = \Phi_{M\times N} \times X_{N\times1} \tag{1}$$

where $\Phi_{M\times N}$ is the *sampling* or *sensing* matrix, and $Y$ is the compressed vector containing $M$ elements. The ratio $R = N/M$ is the compression ratio, and the goal of compressed sensing is to make this as large as possible, i.e., $M << N$.

**Reconstruction.** In compressed sensing, the reconstruction process involves finding the signal vector given the compressive samples. This amounts to finding the solution of an underdetermined linear system. While in general no unique solution is possible, the sparsity of the signal helps find a unique solution when certain conditions are satisfied. One of the most famous reconstruction algorithms is *basis pursuit*, which involves solving the following optimization problem [17, 31]:

$$\hat{X}_{N\times1} = \min_{f_{N\times1}\in\mathcal{R}^N} ||f_{N\times1}||_1 \quad s.t. \quad \Phi_{M\times N} \times f_{N\times1} = Y_{M\times1} \tag{2}$$

where $||\cdot||_1$ is the L1 norm. In particular, this reconstruction is robust to noise and recovers the *sparse approximation* of any signal as long as $M = O(K \log N)$, where $K$ is the sparsity (number of nonzero values) of the signal vector.

Note that *we do not need to reconstruct data in compressive traffic analysis*, as discussed later.

**Applications.** Compressed sensing has recently been used in various contexts, particularly, data compression [9], channel coding [17], and data acquisition [74]. It has specially been used in image processing, for instance for efficient photography [22, 69] and facial recognition [87]. Compressed sensing has recently been applied to some networking problems, particularly, network datasets [91] and network traffic matrix estimation and completion [58]. To the best of our knowledge, we are the first to apply compressed sensing to network traffic analysis.

## 3.2 Introducing Compressive Traffic Analysis

In this paper, we apply the idea of compressed signal acquisition by linear projection to the problem of traffic analysis of encrypted network traffic (as introduced in Section 2). Specifically, we borrow and adapt *linear projection methods* from compressed sensing to make traditional traffic analysis mechanisms more scalable, e.g., faster and less resource-intensive. We refer to this advanced type of traffic analysis as *compressive traffic analysis*. We particularly apply compressive traffic analysis to two popular classes of traffic analysis, *flow correlation* and *website fingerprinting*, and demonstrate their scalability improvements.

*3.2.1 How it works:* In compressive traffic analysis, we leverage random and deterministic *linear projection algorithms* from compressed sensing to compress the traffic features used for traffic analysis, e.g., packet timings, inter-packet delays, and packet sizes. Consider $f_{N\times1}$ to be a *features vector*, i.e., a vector containing $N$ traffic feature values such as $N$ inter-packet delays. We derive the *compressed features vector*, $f^C_{M\times1}$, as:

$$f^C_{M\times1} = \Phi_{M\times N} \times f_{N\times1} \tag{3}$$

The compressed features vector $f^C_{M\times1}$ contains $M$ values where $M < N$. We define the *compression ratio* metric as:

$$R = N/M \ (R > 1). \tag{4}$$

In compressive traffic analysis, the traffic analysis operations are performed on compressed traffic features, instead of raw features, therefore reducing storage, communications, and computation overheads. The goal of compressive traffic analysis is to achieve reasonable traffic analysis performance (e.g., high flow correlation rates) with a large compression ratio $R$. As will be shown later, increasing this compression ratio improves the scalability advantages of compressive traffic analysis.

Note that compressive traffic analysis only uses the linear projection algorithms of compressed sensing to compress traffic features, however, it *does not* need to reconstruct the compressed traffic features, which is discussed below.

*3.2.2 Why it works:* There are two reasons why compressive traffic analysis improves scalability while preserving the performance of traffic analysis:

**1) Sparsity.** Recall from Section 3.1 that compressed sensing performs best on sparse signals. Fortunately, the traffic features commonly used in traffic analysis, namely *packet timings* and *packet sizes* are sparse signals. Figure 1 shows the inter-packet delays of 10,000 network flows randomly selected from CAIDA 2016 traces [13], and Figure 2 shows the histogram of packets sizes of 500 Tor connections. Both figures confirm the sparsity of the packet timing and packet size features, which is in agreement with previous studies [47, 54].

**2) Restricted isometry property (RIP).** Decompressing compressed traffic features is not only computation-intensive (the basis pursuit algorithm requires solving a linear programming), but also adds noise to the recovered signals if the features vector is not exactly sparse [16] (therefore negatively impacting performance). Compressive traffic analysis, however, *does not need to reconstruct* traffic features, and it performs traffic analysis operations directly

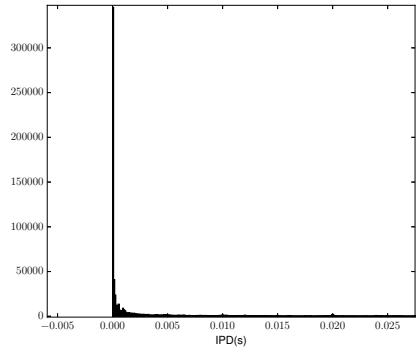


**Figure 1: Sparsity of inter-packet delays (histogram of 10,000 flows)**

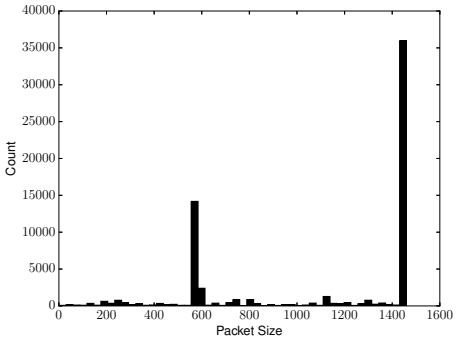


**Figure 2: Sparsity of packet sizes (histogram of 500 Tor connections)**

on the compressed traffic features. This is possible only because the linear projection algorithms used in compressed sensing are designed to preserve the Euclidean distance between traffic features after compression, and therefore traffic analysis can be performed on the compressed features themselves with no need for reconstruction. The Euclidean distance is preserved if the sampling matrix, $\Phi$, satisfies the *restricted isometry property (RIP)* [15], which is a sufficient condition for robust recovery for *any* compressed sensing scheme. The sampling matrix $\Phi$ is said to satisfy the RIP property if for any two sparse vectors $f_{N\times1}$ and $f'_{N\times1}$ with sparsity $K$, we have (5) for some $\delta \in (0, 1)$, where $||\cdot||_2$ is the L2 norm.

$$1 - \delta \leq \frac{||f^C_{M\times1} - f'^C_{N\times1}||_2^2}{||f_{N\times1} - f_{N\times1}||_2^2} \leq 1 + \delta \tag{5}$$

For smaller $\delta$, the compressed feature will better preserve the Euclidean distance.

The RIP property enables us to perform traffic analysis directly on the compressed traffic features (with no need for decompression) without significant degradation in traffic analysis performance. Note that one can perform traffic analysis on the reconstructed traffic features as well, i.e., after decompressing. However, since the reconstruction process can be lossy, this will not only increase the

computation overhead due to running reconstruction algorithms (which involve solving optimization problems), but also degrade the accuracy of traffic analysis (we have demonstrated this in Appendix A).

### 3.3 Candidate Linear Projection Algorithms

The compressive sensing literature has proposed various linear projection methods [1, 4, 7, 14, 18, 27, 51]. We particularly, investigate the random projection method, and a deterministic projection methods based on error-correcting codes in constructing our compressive traffic analysis algorithms.

**Random Sensing:** We investigate the use of the most pervasive compressed sensing method, i.e., the **random projection** [19]. In random sensing, the sampling matrix, $\Phi_{M\times N}$, is generated randomly with i.i.d. entries. Such random $\Phi_{M\times N}$ is then used as a universal compression strategy.

We will discuss the generation of our sensing matrices in the following sections. Comparing various types of random projection algorithms, we find that *Gaussian random projection* performs the best, in which the elements of the basis matrix are generated based on Gaussian distribution. Gaussian random matrices also satisfy the RIP condition due to the *Johnson-Lindenstrauss (JL)* lemma [26], which states that for any $x \in \mathbb{R}^d$ s.t. $|x| \leq 1, |x'| \leq 1$ and an $m \times d$ i.i.d. Gaussian matrix $\Phi$, we have $Pr(|\|\Phi x\|_2^2 - 1| > \delta) < \epsilon$ when $m = O(\delta^{-2}\log(1/\epsilon)$ [5].

**Deterministic Sensing:** Random projection is a traditional way of dimension reduction in Euclidean space. However, in various communications applications it is preferred to use a deterministic projection algorithm; this is primarily because in random projection the sampling matrix needs to be shared between the encoder and decoder parties (an $M \times N$ matrix), whereas in deterministic projection only the parameters of the deterministic matrix need to be shared (e.g., two integer numbers). This is not much of an advantage in the application of compressive traffic analysis since the sensing matrices do not need to be updated frequently. Another advantage of deterministic sensing is the existence of greedy algorithms for fast decompression; this also is not relevant in our application since compressive traffic analysis does not recover the compressed features. However, a third advantage of structured deterministic matrix is that the matrix-vector multiplication in the sampling operation can be done really fast in many cases (by exploiting, say Fast Fourier transform algorithm [75]).

Various deterministic projections have been proposed in the literature [1, 14, 27, 46, 51, 59]. We particularly use sampling matrices generated from dual codes of Bose-Chaudhuri-Hocquenghem codes (BCH) [8, 61]. For any two integers $m > 2$ and $t > 2$, a binary dual-BCH code is a $\{0, 1\}$-matrix of size $M \times N$ where $M = 2^m - 1$ and $N = 2^{mt}$ (this produces a family of matrices that are indexed by $m$ and $t$). To construct deterministic sampling matrices from a dual-BCH matrix, we use a bipolar mapping: $0 \rightarrow +\frac{1}{\sqrt{M}}; 1 \rightarrow -\frac{1}{\sqrt{M}}$, to obtain $\Phi$. Using the basic distance properties of dual-BCH code [57], we can have the following lemma.

Lemma 3.1. *The inner product of any two different columns of $\Phi$ constructed as above from dual-BCH codes is at most $\frac{2(t-1)}{\sqrt{M}}$, and the L2 norm of any column is 1.*

Based on this lemma, the matrix $\Phi$ will satisfy the RIP property for any two features vectors with sparsity $K$, with $\delta \leq \frac{2K(t-1)}{\sqrt{M}}$ (using a result from [7]).

### 3.4 Related Approaches

Our proposed compressed traffic analysis approach can be considered as a technique for dimension reduction. One may consider other approaches towards this. Particularly, one may consider using sketching algorithms to reduce the dimension of traffic features. The only work we are aware of using sketching algorithms is by Coskun et al. [23], where they use standard sketching schemes like count-min to compress traffic features for the purpose of flow correlation. Coskun et al. show that sketching improves the scalability of flow correlation, but it comes at non-negligible degradation in the performance of traffic analysis. Later in Section 5.6 we show that our compressive flow correlation significantly outperforms the sketching-based algorithm of Coskun et al. through experimentation. We argue that the better performance of compressive traffic analysis is due to the fact that they highly preserve the Euclidean distance between (sparse) traffic features, as discussed above.

Other possible approaches to reduce the dimension of traffic features include learning-based algorithms like Principal Component Analysis (PCA) [48] and Independent Component Analysis (ICA) [45] (note that we are not aware of any prior work applying these mechanisms to scale traffic analysis). In addition to the fact that such approaches impose higher computation overheads compared to compressive traffic analysis (which is due to their learning component), they are known [83] to perform worse on sparse signals compared to compressed sensing algorithms. We confirm this through experimentation (Section 5.6).

## 4 COMPRESSIVE FLOW CORRELATION

Flow correlation is to link network flows by correlating traffic features like packet timings and packet sizes. Flow correlation has particularly been studied for the detection of stepping stone attacks [73, 88, 90] and compromising anonymity systems [3, 24, 60, 65, 72, 79, 80, 92] like Tor and mix networks. For instance, an adversary can use flow correlation to link the ingress and egress flows of a Tor connection, and therefore de-anonymize that connection. In Section 2, we overviewed major previous work on flow correlation.

Figure 3 shows a typical scenario of flow correlation. A number of traffic analysis parties, which we call *correlators*, intercept network traffic at different network locations. The number of correlators depends on the application scenario; the figure shows a scenario with two correlators without loss of generality. For instance, when flow correlation is used to attack Tor, the correlators are malicious/-compromised Tor relays, and in the stepping stone application, the correlators are an enterprise network's border routers. The correlators will need to frequently exchange the traffic characteristics of the network flows that they intercept among themselves to enable real-time flow correlation. In Figure 3, such features are exchanged by being frequently written to a *database*.

**Scalability Challenge:** A major limitation of existing flow correlation mechanisms is scaling to today's Internet. Consider the application scenario shown in Figure 3, and suppose that each of the correlators intercept $S$ flows each containing $N$ packets at any

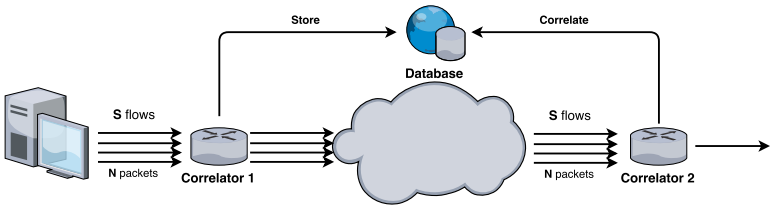


**Figure 3: A flow correlation scenario. For instance, "correlator 1" is an entry Tor relay and "correlator 2" is an exit Tor relay.**

given time.[2] In this case, the order of communications is $O(SN)$, since the correlators need to exchange the traffic features of the flows they intercept (e.g., by storing them in the database). Also, the order of storage is $O(SN)$ as each party will need to store the features of the intercepted flows for the purpose of flow correlation. Finally, the order of computation is $O(S^2N)$; this is because correlator 2 will need to cross-correlate each of the $S$ flows reported by correlator 1 with each of the $S$ flows intercepted by herself, in search for correlated flows. Note that such complexities will increase linearly with the number of traffic correlators, but we will limit our discussions to the two-correlators scenario of Figure 3 to avoid complexity.

In this section, we apply the idea of compressive traffic analysis to flow correlation techniques with the goal of reducing their overheads, as described above. We will demonstrate the advantages of compressive flow correlation by designing compressive flow correlation techniques and comparing them to their non-compressive variants. We will particularly show that our compressive flow correlation schemes are more scalable (as summarized in Table 1).

## 4.1 Overall Architecture

We design a compressive flow correlation algorithm that uses packet timings. Specifically, our compressive flow correlation correlates the inter-packet delays (IPDs) of network flows, similar to a number of traditional flow correlation mechanisms [43, 44, 81, 82]. Many flow correlation schemes use packet timings since obfuscating timing characteristics of network flows is more challenging that other features. For instance, Tor traffic is sent in fixed-size packets called Tor cells to resist size-based flow correlation techniques. This is while Tor does not offer any protection against timing attacks due to its high overhead on Tor traffic [3].

Figure 4 shows the block diagram of our compressive flow correlation algorithm. The correlators (e.g., those in Figure 3) use a *compression algorithm* to compress the features of each of the flows that they intercept. The compressed features vectors are exchanged between the correlators, or are all stored in a centralized database accessible to all of the correlators as shown in Figure 4. In order to check if an intercepted flow is correlated with any of the previously observed flows, a correlator will need to first compress that flow, and then *cross-correlate* its compressed features vector against the compressed features vectors of the previously intercepted flows stored in the database.

Note that the number of entries in the compressed database is linear with the number of intercepted flows, so it can be very large.

[2] In reality, different correlators may intercept various numbers of flows with varying numbers of packets, but we use this setting for simplicity without loss of generality.

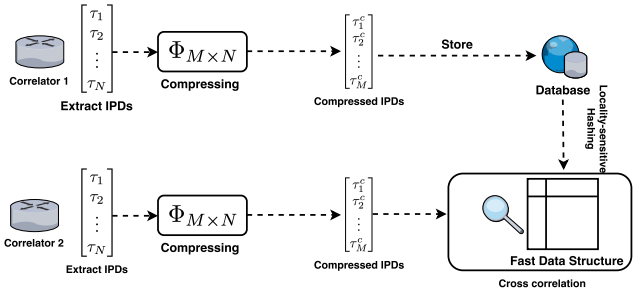


**Figure 4: The block diagram of our compressive flow correlation system**

Therefore, we also design a *fast data structure* which increases the speed of searching for correlated entries in the database.

In the following, we will discuss each of the components of Figure 4 in detail.

## 4.2 IPDs Compression Algorithm

Consider a network flow of length $N+1$ with packet timings $\{t_i | i = 1, 2, ..., N+1\}$, where $i$ is the packet index. The IPDs vector of this flow is given by $\tau_{N\times 1} = \{\tau_i | i = 1, 2, ..., N\}$, where $\tau_i = t_{i+1} - t_i$. As mentioned in Section 3.2, we suggest to build compressive algorithms by compressing raw features vectors. Therefore, our compressive algorithm compresses the IPD features vector $\tau$ as:

$$\tau^C_{M\times 1} = \Phi_{M\times N} \times \tau_{N\times 1} \tag{6}$$

where $\tau^C_{M\times 1}$ is the compressed IPDs vector, $M < N$, and $\Phi_{M\times N}$ (simply $\Phi$) is the sensing matrix.

**Building the sensing matrix ($\Phi$)** The sensing matrix, $\Phi$, needs to be generated and shared between flow correlators before starting the flow correlation process. The selection of the sensing matrix is important to the performance of a compressed sensing algorithm [66]. In Section 3.3 we introduced several candidate mechanisms for generating the sensing matrix. Later in Section 5.2 we will compare these different mechanisms, showing that Gaussian random projection algorithm works the best.

## 4.3 Optimal Cross-Correlation Algorithm

We use *hypothesis testing* [63] to derive the *optimal* cross-correlation algorithm for our compressive flow correlation system. The optimal cross-correlation is one that provides the highest true positive rate given a bounded false positive rate [63].

Suppose that a correlator has just intercepted a network flow $f_1$ with IPDs vector of $\tau_1$ and compressed IPDs vector of $\tau_1^C = \Phi \times \tau_1$ (of size $M \times 1$). The correlator aims at checking if $f_1$ is correlated with (i.e., linked to) a previously observed flow $f_2$ whose compressed IPDs vector, $\tau_2^C = \Phi \times \tau_2$, is recorded in the database. Therefore, the correlator will need to decide which of the following two hypotheses holds:

- *Correlated* ($H_1$): $f_1$ and $f_2$ are correlated, i.e., $f_1$ is a noisy version of $f_2$. That is $\tau_{1,i} = \tau_{2,i} + \Delta_i$ for $i = 1, 2, ...$, where $\tau_{1,i}$ and $\tau_{2,i}$ are the $i$th IPDs of $f_1$ and $f_2$, respectively, and $\Delta_i$ is network jitter on the $i$th IPD.
- *Not Correlated* ($H_0$): $f_1$ is not a noisy version of $f_2$.

We have that:

$$\begin{cases} H_0 : \tau_2^C = \Phi \times (\tau^* + \Delta) \\ H_1 : \tau_2^C = \Phi \times (\tau_1 + \Delta) \end{cases} \tag{7}$$

where $\tau^*$ is the IPDs of an arbitrary flow not related to flow $f_1$.

We model IPDs ($\tau_{1,i}$'s and $\tau_{2,i}$'s) as i.i.d. exponential distributions with rate $\lambda$, as in previous work [44]. We also model network jitter, $\Delta_i$'s, as i.i.d. Laplace distributions with mean zero and standard deviation $\delta$. This model has been used by previous work [43], but we also confirm it through measuring network jitter between 20 Planetlab nodes [10] as shown in Figure 5.

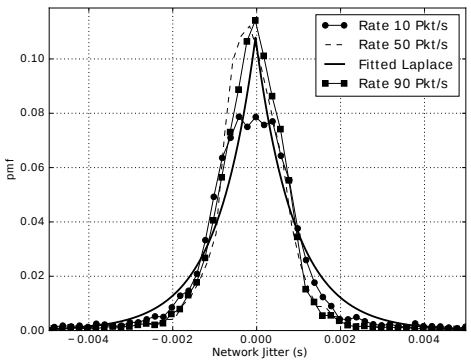


**Figure 5: Measured jitter between Planetlab nodes, which is fitted to a Laplace distribution.**

Finally, we use the Neyman-Pearson Lemma [63] to derive the optimal cross-correlation function as a maximum-likelihood ratio test:

$$\begin{cases} H_1 & LT > \eta \\ H_0 & LT \leq \eta \end{cases} \tag{8}$$

where $\eta$ is the decision threshold and

$$LT = \frac{\mathbb{P}(\Phi \times (\tau_1 + \delta))}{\mathbb{P}(\Phi \times (\tau^* + \delta))}. \tag{9}$$

Deriving the optimal $LT$ function is very complex for arbitrary $\Phi$ matrices as it involves the computation of multiple convolutions (i.e., integrations). In our simulations discussed later, we will implement $LT$ for specific $\Phi$ matrices.

## 4.4 Practical Cross-Correlation Algorithm

Despite being optimal in performance, the optimal cross-correlation function of Section 4.3 has two main drawbacks. First, it involves performing many PDF convolution operations (i.e., integrations), depending on the size of $\Phi$; this makes the correlation process very slow. Second, the optimal algorithm is optimal only if the network parameters such as a flow's data rate and network jitter are estimated precisely. Such parameters change over time, and are hard to predict for a live connection.

We therefore design a non-optimal cross-correlation algorithm that is several orders of magnitude faster than the optimal algorithm, but only slightly underperforms the optimal algorithm. Our non-optimal cross-correlation function uses *cosine similarity* [63] to correlate compressed IPDs vectors. That is, for two vectors of compressed IPDs, $\tau_1^C$ and $\tau_2^C$ (each of length $M$), the cross-correlation is evaluated as:

$$C(\tau_1^C, \tau_2^C) = \frac{\sum_{i=1}^{M} \tau_1^C(i)\tau_2^C(i)}{\sqrt{\sum_{i=1}^{M} \tau_1^C(i)^2}\sqrt{\sum_{i=1}^{M} \tau_2^C(i)^2}} \tag{10}$$

As demonstrated in the experiments (Figures 13 and 11), our cosine-based cross-correlation algorithm is only slightly less accurate than the optimal algorithm of Section 4.3, but is several orders of magnitude faster. It is also not sensitive to flow rate and network jitter parameters. Using Cosine similarity as our correlation function also enables us to speed up the database search process, which is discussed in the following section.

*4.4.1 Resisting Packet-Level Modifications.* In addition to resisting network delays (e.g., jitter), a flow correlation algorithm needs to be resistant to packet-level modifications such as packet drops, repacketization, and reordering. While such modifications can happen very occasionally, depending on the network condition, even a single packet modification can de-synchronize a flow correlation algorithm.

Previous IPD-based flow correlation schemes [43, 44] use a sliding window-based mechanism to resist potential packet-level modifications. However, that algorithm significantly increases the computation complexity as it needs to match every IPD in an egress flow with multiple IPDs in an ingress flow. We propose an alternative algorithm to resist packet-level modifications with negligible overhead. Our algorithm can be used with any IPD-based cross-correlation function, including our cross-correlation functions introduced before.

Our algorithm is summarized in Algorithm 1. It divides the time interval into non-overlapping intervals of length $L$ seconds. Suppose that we receive a network flow with $N$ IPDs ($N + 1$ packets). The algorithm generates the vector of "raw" IPDs by picking only the first $N_I$ IPDs from each of the $L$-long intervals. Therefore, if the flow has more than $N_I$ IPDs in an interval the extra IPDs will be ignored, and if an interval has less than $N_I$ IPDs the algorithm will put zero for the non-existing IPDs. This makes the correlation resistant to packet modifications: if a single packet is dropped or repacketized it will only impact the IPDs in its own interval, but will not impact any of the IPDs in other intervals.

It is still possible that a network packet moves from one interval to its next interval due to network jitter. This will desynchronize

both of the intervals. To prevent this, we define guard subintervals of length $g < L$ at the end of each interval, and we exclude any packets in those guard intervals. $g$ should be chosen based on network jitter. We model network jitter, $\Delta$, as a Laplace distribution with mean zero and standard deviation $\sigma$, as discussed earlier. Therefore, using Chebyshev's inequality:

$$\mathbb{P}(|\Delta| \geq g) \leq \frac{\sigma^2}{g^2}. \tag{11}$$

Therefore, we can ensure that the probability of interval desynchronization is less than $\epsilon$ by choosing $g = \sigma/\sqrt{\epsilon}$. For instance, for $\sigma = 5msec$ and $\epsilon = 0.01$, we have $g = 50msec$.

**Algorithm 1** Algorithm to resist packet-level modifications

```
L ←Interval Length
N_I ←Interval max packets
g ←Guard Value
I ←list of intervals
for each captured flow F do
    T_F ← Extract the Timing information of flow F
    for each t_i^F in T_F do
        i = ⌊t_i^F / L⌋
        if |i * L − t_i^F| < g and |(i + 1) * L − t_i^F| < g then
            Insert t_i^F to I_i
    for each I_i do
        Compute τ_{F_i} from I_i
        Resize τ_{F_i} to N_I, add zeros if needed or remove the end
    Merge all τ_{F_i} to get τ_F
    Save τ_F
```

## 4.5 Fast Data Structure

In a flow correlation scenario, a correlator needs to correlate an intercepted network flow against—all—of the previously observed network flows stored in the IPDs database. For instance, an egress Tor flow observed by a malicious Tor exit node needs to be cross-correlated against all of the ingress flows observed by a malicious Tor guard node. Note that this is not specific to our compressive correlation algorithms, and is a (bigger) constraint for traditional correlation algorithms as well.

We design a fast data structure for storing compressed IPD features based on the *locality-sensitive hashing (LSH)* [34] data structures. LSH is a particular hash function that, unlike traditional cryptographic hash functions, produces similar hashes for similar inputs. LSH offers a provable *sub-linear* query time and *sub-quadratic* space complexity, despite its very good performance [71]. LSH divides the features spaces into a number of "hyperplanes," where similar instances will appear on the same hyperplane with high probability. As illustrated in Figure 6, LSH reduces search time and complexity by only looking at the most probable hyperplane, as opposed to the whole database.

We specifically use a recent LSH algorithm by Andoni et al. [2], which uses Euclidean distance on the sphere to measure the distance between entries. This makes Andoni et al. [2]'s LSH a natural option for our system since cosine similarity (our sub-optimal cross-correlation function) on the plain coordinates is equivalent to the

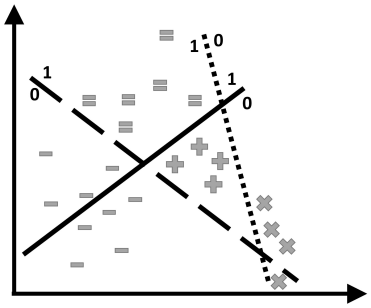


**Figure 6: An illustration of how LSH stores and looks up entries**

Euclidean distance on the sphere. We therefore use this LSH algorithm to fragment the space on the sphere to hyperplanes. As a result, compressed IPD vectors with close cosine-similarity will end up on the same hyperplanes with high probability.

In order to add a new flow to our database, we first compress that flow, and then use LSH to find the hyperplane in the database with the "closest" compressed IPDs vector to that flow (using LSH). We then insert the flow's compressed IPDs vector in that position. In order to find a match in the database for a given flow we similarly identify the position (hyperplane) in the database with the closest compressed IPDs vectors, and only apply our cosine similarity cross-correlation function on the items of that hyperplane.

The use of LSH significantly improves the speed of database search. While the normal search in the database has a query time complexity of $O(S)$ ($S$ is the number of flow entries) Andoni et al.'s LSH reduces this to $O(S^\rho)$, where $\rho < 1$ represents the search accuracy.

## 4.6 Scalability Improvements

Table 1 compares the storage, computation, and communications complexity of our compressive flow correlation algorithms with their traditional, non-compressive alternatives.

**Communications and storage complexities:** Any compressive algorithm will reduce the order of communications and storage from $O(SN)$ to $O(SN/R)$ (where $R > 1$), as they will exchange and store the "compressed" flow features, as opposed to the raw features.

**Computation complexity:** Compressive algorithms also significantly reduce the computation complexity. Cross-correlation algorithms scale linearly with the number of features. Therefore, a compressive correlation algorithm reduces computation by a factor of $R = N/M$.

Note that a compressive system's compression process may add some computation overhead; in particular, our IPD compression algorithm performs matrix multiplications, therefore imposing a computation complexity of $O(NM)$ for a random $\Phi$ matrix (this can further reduce for sparse matrices). Therefore, compressive flow correlation changes computation overhead from $O(S^2N)$ to $O(S^2M + SMN)$. However, since $M$ is negligible compared to $S$ (the number of flows), we can approximate the computation complexity

**Table 1: Complexity comparison of different flow correlation algorithms ($R > 1$, $\rho < 1$)**

| Algorithm | Commun. | Storage | Computation |
|---|---|---|---|
| Non-compressive optimal | $O(SN)$ | $O(SN)$ | $O(S^2N)$ |
| Compressive optimal | $O(SN/R)$ | $O(SN/R)$ | $O(S^2N/R)$ |
| Non-compressive Cosine | $O(SN)$ | $O(SN)$ | $O(S^2N)$ |
| Compressive Cosine | $O(SN/R)$ | $O(SN/R)$ | $O(S^2N/R)$ |
| Compressive Cosine+LSH | $O(SN/R)$ | $O(SN/R)$ | $O(S^{(1+\rho)}N/R)$ |

of compressive algorithm as $O(S^2M)$, which has a factor of $N/M$ reduction compared to traditional flow correlation.

The use of LSH in our algorithm additionally reduces the computation overhead to $(S^{(1+\rho)}M)$, where $\rho < 1$ represents search accuracy as introduced earlier.

Note that even though the computation overhead of both of the optimal and cosine cross-correlation algorithms scale linearly with the number of samples, the optimal correlation algorithm is *by far* slower than the cosine similarity algorithm. This is due to it performing multiple computation-intensive operations such as exponentiation and integration. This is also shown later in the experiments.

# 5 EXPERIMENTS: COMPRESSIVE FLOW CORRELATION

## 5.1 Experimental Setup and Metrics

We use network flows from the CAIDA 2016 anonymized network traces [13] (out of which we pick 50,000 random flows each at least 1000 packets long). For any network flow, we simulated the impact of network jitter by modeling jitter as a zero-mean Laplace distribution. This model is also used in previous studies of flow correlation, but we additionally confirm it by measuring network jitter between 20 Planetlab nodes [10], which is shown in Figure 5. We also simulated packet drops by dropping packets based on a Binomial distribution.

We implemented our compressive flow correlation algorithms in Python, and used Mathematica 11.0 to derive some of the math formulas. We used the Communication System Toolbox of Matlab to generate BCH codes.

**Metrics:** We use two metrics for evaluating the performance of the flow correlation algorithms. The *True Positive (TP)* rate shows the ratio of correctly linking related network flows across all experiments, i.e., when a network flow is decided to be correlated to a noisy version of itself. On the other hand, the *False Positive (FP)* rate shows the ratio of incorrect matches across all experiments, i.e., when the flow correlation function declares two non-related flows to be correlated. Note that the value of the detection threshold, $\eta$, trades off FP and TP.

We show a flow correlation algorithm `A` is more *scalable* than `B` by showing that one of the two equivalent conditions hold: (1) when both algorithms offer the same flow correlation performance (e.g., same TP/FP numbers), `A` has less storage, communications, and computation overheads (e.g., it is faster by using fewer IPDs for correlation), or (2) for the same storage, communications, and computation overheads (i.e., same number of IPDs), `A` provides better flow correlation performance (e.g., better TP/FP metrics).

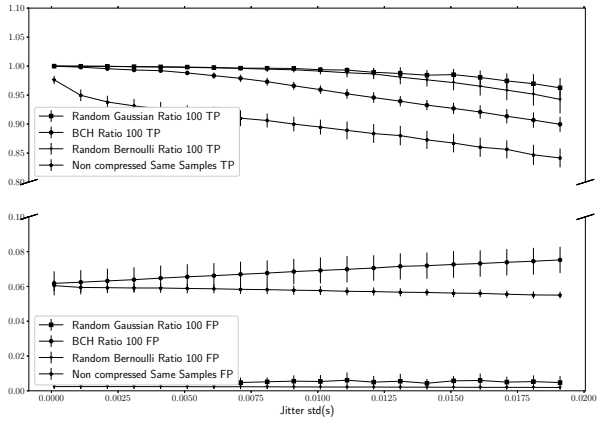


**Figure 7: Comparing compressive flow correlation with various projection algorithms with non-compressive correlation. All compressive correlation algorithms outperform the non-compressive correlation, and the random Gaussian projection performs the best.**

## 5.2 Generating the Sensing Matrix $\Phi$

We investigate several major linear projection algorithms, introduced in Section 3.3, to compress traffic features. We generate the sensing matrix $\Phi$ for each of these algorithms, and compare their performance. Particularly, Figure 7 compares the TP and FP performance of compressive flow correlation for Random Gaussian projection, Random Bernoulli projection, and BCH projection, along with non-compressive flow correlation (for $M = 10$, $N = 1000$, and $R = 100$). As can be seen, all three types of compressive correlation algorithms outperform non-compressive (traditional) traffic analysis.

**Random Gaussian projection performs the best:** We see from Figure 7 that the random Gaussian projection outperforms all the other projection mechanisms. This is because, as discussed in Section 3.3, based on the Johnson-Lindenstrauss (JL) lemma [26] Gaussian random sensing tightly preserves the Euclidean distance on the compressed features. Therefore, for the rest of the experiments, we use Gaussian random projection for compressing traffic features. We particularly use Candes and Tao's mechanism [17], where the elements of the $\Phi$ matrix are i.i.d. Gaussian random variables with mean zero and a constant standard deviation $\sigma$ [17]. Figure 8 shows

the impact of $\sigma$ on the performance of flow correlation. We choose $\sigma = 0.01$ as it optimizes the performance of flow correlation.

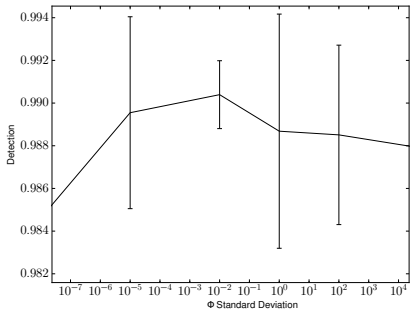


**Figure 8: Impact of Φ's standard deviation ($\sigma$) on correlation.**

**Impact of Compression Ratio:** As intuitively expected, increasing the compression ratio $R$ improves scalability, but at the expense of degrading the accuracy of flow correlations. Figure 9 shows the TP and FP metrics for different compression ratios.

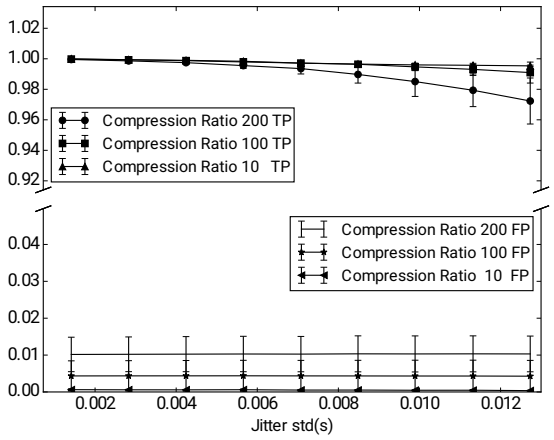


**Figure 9: Impact of compression ratio $R$ on TP and FP metrics**

## 5.3 Cosine Similarity vs. Optimal Detector

We compare our optimal (Section 4.3) and cosine similarity (Section 4.4) compressive algorithms. Since the optimal correlation needs to be tailored for each specific flow rate, we pick network flows from our CAIDA dataset that are in a short range of flow rates, specifically 10 to 20 packets per second. Figure 13 compares optimal and cosine similarity algorithms for various jitters (note that in our Planetlab measurements, we find the average jitter SD ($\delta$) to be around 5$ms$, but we simulate our correlation algorithms for a wider range of $\delta$). As can be seen, the optimal correlation only *slightly* outperforms the non-optimal, cosine-similarity correlation function. This is while the optimal correlation is *significantly* more computation-intensive, as discussed earlier in Section 4.6. For instance, for $S = 1000$ the cosine correlator is about two orders of magnitude faster (see Figures 10 and 11), which is in full agreement with Table 1. Additionally, unlike the optimal detector, the cosine similarity algorithm is not sensitive to network parameters like flow rates.

## 5.4 Comparing Runtime

We compare the runtime of different algorithms on a computer with a 3.5 GHz Intel Xeon processor and 48GB of memory. Figure 10 compares the runtime of compressive and non-compressive optimal correlation functions ($R$=10). As can be seen, the compressed alternative is faster *by an order of magnitude*, while both algorithms have the same accuracy (which is in agreement with Table 1). Also, Figure 11 compares the runtime of the compressed and non-compressed versions of the cosine-based correlator. As can be seen, the Compressed version is *two orders of magnitude faster* than its non-compressed version (for similar correlation performance).

## 5.5 Impact of Packet Drops

Figure 12 shows the performance of our algorithm designed to resist packet level modifications (Algorithm 1) against dropping packets at different rates. Note that in our Planetlab measurements, the average packet drop rate is only 0.001, but we simulate higher rates of drops to simulate active attackers.

## 5.6 Comparison to Other Approaches

**Non-compressive flow correlation:** We compare our compressive correlation algorithms to Houmansadr et al. [44], which is the state-of-the-art non-compressive alternative to our system. Figure 14 compares our optimal compressive algorithm (with $R = 10$) to the optimal non-compressive algorithm of Houmansadr et al. [44]. As can be seen, our system performs the same while being significantly faster as demonstrated in Figure 10 (discussed above) and shown in Table 1. On the other hand, when both systems use the same number of IPDs (and therefore have the same runtime), our compressive algorithms performs significantly better flow correlation (See Figure 15).

**Sketching-based correlation:** We compare our algorithm to the sketching-based algorithm of Coskun et al. [23]. Figure 16 shows that our algorithm significantly outperforms Coskun et al. [23]'s for the same number of IPD samples.

**Dimensionality reduction-based correlation algorithms:** A potential alternative to compressive traffic analysis is using learning based algorithms to reduce the dimension of traffic features. While we are not aware of any prior work applying these mechanisms to scale traffic analysis, we implement a flow correlation algorithm that uses Principal Component Analysis (PCA) [48] to compress traffic features. Our algorithm works by first training PCA on the dataset of traffic features, and then we reduce the dimension of dataset by multiplying the its instances into the PCA matrix. As shown in Figure 17, our compressive flow correlation algorithm outperforms the PCA-based algorithm, except for small jitters where both perform the same. Also, the PCA-based algorithm is much slower due to its training phase. Nonetheless, even the PCA-based

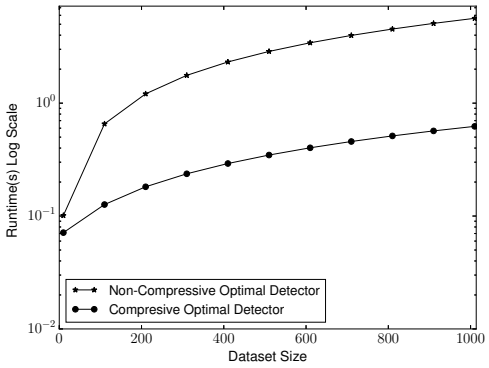


**Figure 10: Runtime comparison: optimal compressive vs. optimal non-compressive ($R = 10$)**

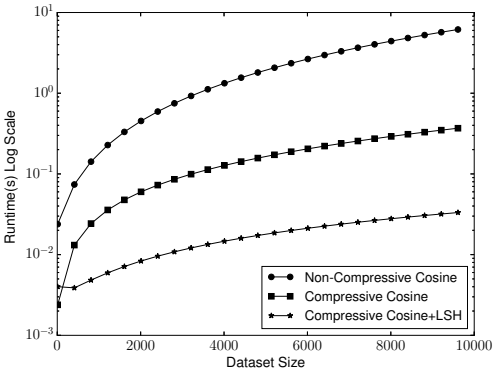


**Figure 11: Runtime comparison: Cosine compressive vs. Cosine non-compressive ($\rho = 0.2$ and $R = 10$)**

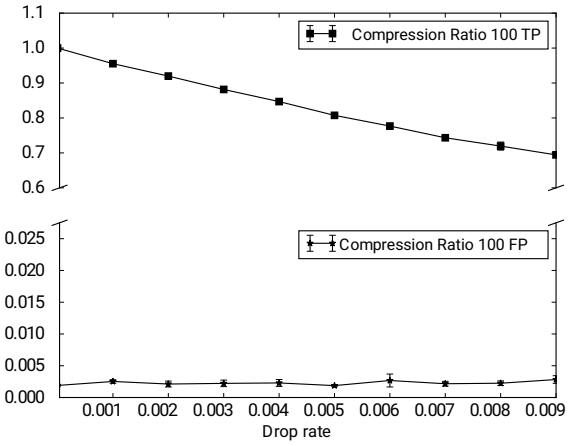


**Figure 12: Impact of packet drops on performance (1000 packets, no network jitter)**

algorithm significantly outperforms the traditional flow correlation, which may be further investigated by future research.

## 5.7 Experiments on Real Tor Network

We implemented our compressive flow correlation tool, and experimented with it on live Tor traffic by intercepting Tor connections. In our experiments, we have two correlator entities that are intercepting Tor traffic with the goal of de-anonymizing Tor connections.

In our experiments, we browse the top 500 websites from Alexa[3] over Tor. We do not run any Tor relays, instead we use Tor's existing relays. In order to be able to intercept traffic going to a webpage through Tor, we use a sniffing HTTP proxy and set up our Tor connections to go through that proxy. We also capture Tor traffic on our local client before going to a Tor guard relay.

[3]http://www.alexa.com/topsites

Figure 18 compares the performance of our compressive flow correlation algorithm (with rate 4pps) with its non-compressive alternative when they both use *the same* number of IPDs features (therefore, they have the same overhead). As can be seen, our compressive algorithm *significantly outperforms* its non-compressive alternative. Note that a system performs better if its lines of TP and FP metrics have a larger gap. For instance, when both systems offer a FP of 0.1, our compressive algorithm offers a TP of $\approx 0.9$, whereas the non-compressive algorithm offers a TP of only $\approx 0.3$.

# 6 COMPRESSIVE WEBSITE FINGERPRINTING

Website fingerprinting is a class of traffic analysis that aims to identify the websites (or webpages) visited over encrypted channels like VPN, Tor, and network proxies [12, 21, 38, 62, 77]. While there has been various proposals for website fingerprinting [12, 33, 35, 37, 50, 56, 62, 77, 78], state-of-the-art website fingerprinting mechanisms mainly use one of the two machine learning algorithms of *Support Vector Machines (SVM)* [12, 62, 78] or *k-Nearest Neighbor (k-NN)* [33, 35, 77]. In this section, we present the idea of compressive website fingerprinting by applying it on recent $k$-NN and SVM-based fingerprinting schemes and demonstrating their significant scalability improvements.

## 6.1 Overall Architecture

Compressive website fingerprinting differs from regular website fingerprinting in that it trains and classifies using compressed traffic features, as opposed to using raw traffic features as in the traditional approach. Figure 19 shows the overall architecture of compressive website fingerprinting. As can be seen, an $N$-long vector of webpage features, $f_{N\times 1}$, is compressed to

$$f^C_{M\times 1} = \Phi_{M\times N} \times f_{N\times 1} \tag{12}$$

This results in the *compressed features vector* of length $M < N$, $f^C_{M\times 1}$, which is then used by the website fingerprinting system's machine learning algorithms for training and classification. What enables us to apply compressive traffic analysis to website fingerprinting is

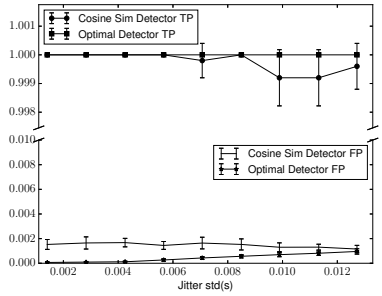


**Figure 13: Comparing our compressive algorithms: optimal vs. cosine-similarity ($R = 10$).**

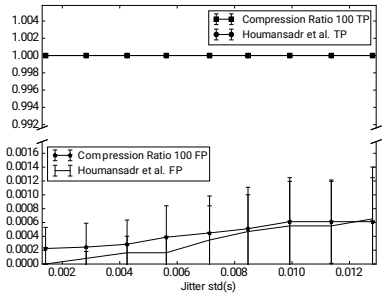


**Figure 14: Comparing our optimal compressive algorithm ($R = 10$) to its optimal non-compressive alternative [44]. Our system performs the same despite being significantly faster (Table 1).**

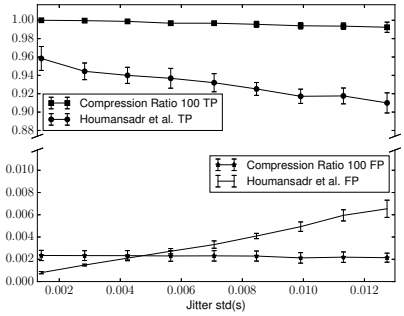


**Figure 15: Comparing our cosine compressive algorithm ($R = 100$) to optimal non-compressive algorithm [44] with the same runtime. The compressive algorithm significantly outperforms for the same runtime.**

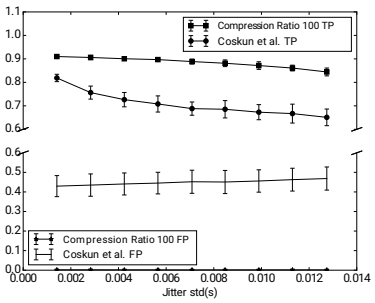


**Figure 16: Comparing our compressive algorithm to Coskun et al. [23] for a drop rate of 0.002; with the same number of IPD samples, our algorithm significantly outperforms.**

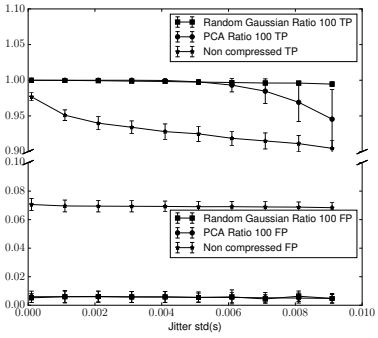


**Figure 17: Comparing Compressive flow correlation with PCA-based correlation.**

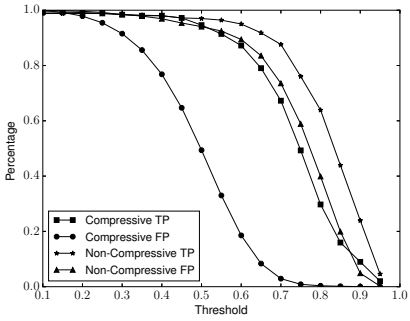


**Figure 18: Comparing our compressive algorithm against its non-compressive algorithm on Tor; when both use the same number of IPDs features, our compressive algorithm significantly outperforms regarding the TP and FP metrics.**

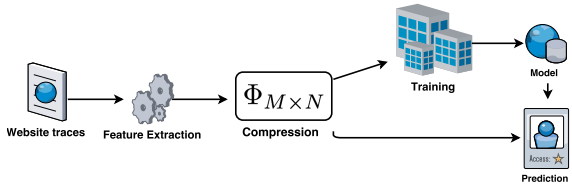


**Figure 19: General framework of compressive website fingerprinting**

the sparsity of traffic features like packet timings and sizes, as well as the RIP property of linear projection algorithms that preserves the Euclidean distance of features vectors after compression (as discussed in Section 3.2).

Similar to compressive flow correlation, we investigate the use of various linear projection algorithms in generating the basis matrix, $\Phi$. We will discuss our choices later in the experiments section.

## 6.2 Improving Scalability

The state-of-the-art website fingerprinting attacks mainly use one of the two machine learning algorithms of Support Vector Machines (SVM) [12, 62, 78] and $k$-Nearest Neighbor ($k$-NN) [33, 35, 77]. Each of these mechanisms are composed of a learning phase and a prediction phase. For any webpage, they generate a features vector of size $N$; this $N$-long features vector is then used for training and prediction. The size of the features vector, $N$, plays an important role in the performance of a website fingerprinting mechanism: increasing $N$ improves the classification performance but makes the fingerprinting process more expensive by increasing the computation and storage overheads (it also increases the communications overhead proportional to the storage overhead if the learning and classification nodes are running at different network locations).

The goal of compressive website fingerprinting is to improve the scalability of website fingerprinting by achieving similar classification accuracy with a smaller $N$. Table 2 demonstrates the

scalability improvement for $k$-NN and SVM-based fingerprinting systems, which is discussed in the following.

*6.2.1 Scaling $k$-NN based schemes.* Several recent website fingerprinting schemes [33, 35, 77] use the $k$-NN approach for classification using custom weight learning. For instance, Wang et al. [77] uses packet sizes, packet timings, and packet directions as features for classification, and it uses the $L_1$ distance metric for finding the nearest neighbors.

The prediction time of a $k$-NN based fingerprinting system is $O(SN)$, where $S$ is the size of the dataset (number of webpages) and $N$ is the number of the features per webpage [11]. Our compressive alternative of a $k$-NN fingerprinting system reduces the number of features to $M < N$, therefore lowering the time of $k$-NN computations to $O(SN/R)$, where $R = N/M$ is the compression ratio ($R > 1$). This is summarized in Table 2.

*6.2.2 Scaling SVM-based schemes.* Several recent proposals [12, 62, 78] use the support vector machines (SVM) algorithm to learn and classify websites. Support vector machines have a time complexity between $O(S^2)$ and $O(S^3)$ [55] for the training phase ($S$ is the size of dataset). The SVM mechanisms also have a prediction phase time complexity of $O(N_{sv}(N) \cdot N)$, where $N_{sv}$ is the number of support vectors and $N$ is the number of features [55]. Note that $N_{sv}$ itself is a function of $N$, therefore the prediction time of SVM is not linear in $N$. However, as also shown in our experiments, the prediction time is monotonic with $N$ (therefore it is improved due to compression). This is summarized in Table 2.

# 7 EXPERIMENTS: COMPRESSIVE WEBSITE FINGERPRINTING

## 7.1 Experimental Setup and Metrics

We demonstrate the scalability improvements of compressive website fingerprinting for the two leading approaches for website fingerprinting, i.e., $k$-NN-based [12, 62, 78] and SVM-based [33, 35, 77] mechanisms. We particularly pick the state-of-the-art systems from each group in our experiments, namely, we pick Panchenko et al. [62] to represent $k$-NN and Wang et al. [77] to represent the SVM-based systems. We use *the original codes and datasets* from Wang et al. [77] and Panchenko et al. [62] in our experiments. We implement the compressive versions of these systems by modifying their codes as described in Section 6.1 (e.g., by adding the compression stage). Our experiments are run on a Linux machine with 48GB of memory and a 3.5GHz Xeon(R) CPU.

**Metrics:** We use website fingerprinting *accuracy*, as used in the literature, to evaluate and compare the performance of website fingerprinting systems. We show a fingerprinting algorithm `A` is more *scalable* than `B` by showing that one of the two equivalent conditions hold: (1) when both algorithms offer the same (or very close) website fingerprinting accuracy, `A` has less storage, communications, and computation overheads (e.g., it is faster by using fewer features), or (2) for the same storage, communications, and computation overheads, `A` provides higher fingerprinting accuracy.

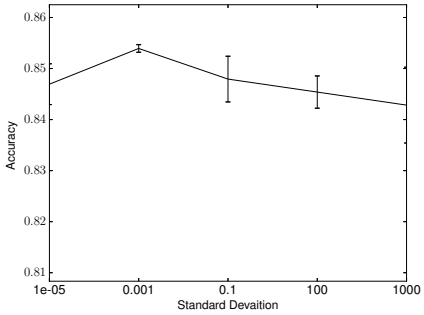


**Figure 20: Impact of $\Phi$'s standard deviation ($\sigma$) on the accuracy of $k$-NN website fingerprinting algorithm.**

## 7.2 Generating the Sensing Matrix $\Phi$

We investigate several major linear projection algorithms, introduced earlier, for compressive website fingerprinting. We generate the sensing matrix $\Phi$ for each of these algorithms, and compare their performance. We find that the Gaussian and Bernoulli random projection algorithms perform the best in improving the scale of website fingerprinting, as will be presented in the following. Also, note that the parameters of $\Phi$ impact the performance. Particularly, Figure 20 shows the impact of $\Phi$'s standard deviation, $\sigma$, on the accuracy of our compressive version of Wang et al. [77] when Gaussian random projection is used (we pick $\sigma = 10^{-3}$).

## 7.3 Compressive $k$-NN

We compare the accuracy and run time of the state-of-the-art $k$-NN system of Wang et al. [77] to its compressive alternative. We run our experiments in an open-world scenario with 100 monitored websites with 90 instances each, and 5000 non-monitored websites. To demonstrate the scalability improvements, we compare the compressive version of the algorithm with the original algorithm for the same number of features (we always keep the most significant features when we sample down the original non-compressive algorithm). Table 3 summarizes the results (note that the accuracy loss and speedup metrics are evaluated compared to non-compressive for $R = 1$). As can be seen, both Bernoulli and Gaussian compressive algorithms outperform the original system, i.e., for the same number of features they result in higher fingerprinting accuracy. For instance, when all algorithms use 467 features, and therefore have the same storage/computation complexity, our Gaussian compressive algorithm provides a 0.8703 accuracy compared to 0.6880 of the original, non-compressive algorithm. Also, similar to our results on compressive flow correlation, we see that Gaussian linear projection provides a better performance compared to other compression mechanisms.

Table 3 also shows the speedup of our Gaussian compressive over Wang et al. [77] when both algorithms offer the *same accuracy*. For instance, with $R = 16$, for the same accuracy, our compressive algorithm makes the fingerprinting process *about 13 times faster*. Note that the speedup is a little bit less than $1/R$ (shown in Table 2) which is due to the slight drop in accuracy due to compression (i.e., compression preserves the Euclidean distance, but not perfectly).

**Table 2: Comparing compressive and traditional website fingerprinting mechanisms regarding storage and computation complexities ($R > 1$ is the compression ratio).**

| Algorithm | Storage | Learning Phase | Prediction Phase |
|---|---|---|---|
| SVM | $O(SN)$ | $O(S^2) \sim O(S^3)$ | $O(N_{sv}(N) \cdot N)$ |
| Compressive SVM | $O(SN/R)$ | $O(S^2) \sim O(S^3)$ | $O(N_{sv}(N/R) \cdot N/R)$ |
| $k$-NN | $O(SN)$ | N/A | $O(SN)$ |
| Compressive $k$-NN | $O(SN/R)$ | N/A | $O(SN/R)$ |

**Table 3: Comparing compressive and non-compressive (regular) $k$-NN-based fingerprinting performance (the $R = 1$ row is the baseline).**

| Features ($N$) | $R$ | Gaussian Compressive | Bernoulli Compressive | Non-compressive [77] | Total runtime | Accuracy loss | Effective Speedup |
|---|---|---|---|---|---|---|---|
| 3736 | 1 | 0.9076 | 0.9076 | 0.9076 | $\sim 33m$ | 0.0 | ×1 |
| 934 | 4 | 0.8913 | 0.86 | 0.6920 | $\sim 9m$ | 0.0163 | ×3 |
| 467 | 8 | 0.8703 | 0.85 | 0.6880 | $\sim 4m$ | 0.0373 | ×6.75 |
| 233 | 16 | 0.8520 | 0.82 | 0.6800 | $\sim 2m$ | 0.0556 | ×13.5 |

**Table 4: Comparing compressive and non-compressive (regular) SVM-based fingerprinting performance (the $R = 1$ row is the baseline).**

| Features ($N$) | $R$ | Gaussian Compressive | Bernoulli Compressive | Non-compressive [62] | Prediction Time | Accuracy loss | Speedup |
|---|---|---|---|---|---|---|---|
| 104 | 1 | 0.824 | 0.8240 | 0.8240 | $\sim 3s$ | 0.0 | ×1 |
| 26 | 4 | 0.8052 | 0.7923 | 0.7860 | $\sim 1.5s$ | 0.0188 | ×2 |
| 13 | 8 | 0.7867 | 0.7775 | 0.7552 | $\sim 1s$ | 0.0373 | ×3 |

## 7.4 Compressive SVM

We also compare the accuracy and run time of the state-of-the-art SVM-based system of of Panchenko et al. [62] to its compressive alternative. We run our experiments in the closed-world setting with 100 websites, each with 40 instances. As before, to demonstrate the scalability improvements, we compare the compressive version of the algorithm with the original algorithm for the same number of features (as before, we keep the most significant features when we sample down the original non-compressive algorithm). Table 4 summarizes the results, showing that both Bernoulli and Gaussian compressive algorithms outperform the original system, i.e., for the same number of features they result in higher fingerprinting accuracies (as before, the accuracy loss and speedup metrics are evaluated compared to non-compressive for $R = 1$). For instance, when they both use 26 features (and therefore have a similar storage/runtime overhead), the Gaussian compressive algorithm provides a 0.8052 accuracy compared to the 0.7860 accuracy of the original algorithm. Also, as before the Gaussian linear projection provides a better performance compared to other compression mechanisms.

We also see that increasing the compression ratio $R$ results in a larger speedup, while slightly reducing the fingerprinting accuracy. For instance, a compression ratio of $R = 8$ speeds up the prediction time three times for a negligible 0.03 accuracy reduction. Note that as discussed in Section 6.2.2, the prediction time of SVM is not linear with the number of features. However, as also shown here, the prediction time is monotonic with the number of features, therefore compression always improves the scalability.

The runtime gain of compression is larger for $k$-NN-based fingerprinting systems, as shown above, due to the linear relation of its complexity with the number of features.

## 7.5 Significance of the Scalability Improvements

As discussed earlier, the scalability improvement of compressive fingerprinting is two-fold: lower storage and computation (prediction time). One may argue that with the reduced costs of storage, the storage benefits of compressive website fingerprinting may not matter that much, especially when the number of fingerprinted websites is moderate (e.g., if the close world is limited to only 100 specific websites). Nonetheless, the fingerprinting adversary will still benefit from the improved computation overhead, resulting in faster detection of the fingerprinted webpages in live traffic, as shown in Tables 3 and 4. This will be more significant if the fingerprinting adversary needs to inspect a large number of network flows in real-time, e.g., an adversary in control of an Internet IXP.

# 8 CONCLUSIONS

We introduced a new direction to traffic analysis, which we call compressive traffic analysis. Our approach is inspired by the trending research area of compressed sensing, and works by using compressed traffic features, as opposed to raw features, for traffic analysis. We discussed why and how compressive traffic analysis improves the scalability of traffic analysis. We also demonstrated compressive traffic analysis on flow correlation and website fingerprinting, two

widely-studied types of traffic analysis. Our evaluations show significant improvements (e.g., orders of magnitude speedup) over traditional algorithms.

We believe that compressive traffic analysis is a significant step forward in designing traffic analysis algorithms that scale to the exploding volumes of network communications. An interesting topic for future work will be to investigate the application of compressive traffic analysis to other kinds of traffic analysis, such as active flow correlation mechanisms. Improving the performance of feature compression is another topic for future work.

## ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their comments. This work was supported by the NSF grants CNS-1525642 and CCF-1642550.

## REFERENCES

[1] Arash Amini and Farokh Marvasti. 2011. Deterministic construction of binary, bipolar, and ternary compressed sensing matrices. *IEEE Transactions on Information Theory* 57, 4 (2011), 2360–2370.
[2] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. In *Advances in Neural Information Processing Systems*. 1225–1233.
[3] Adam Back, Ulf Möller, and Anton Stiglic. 2001. Traffic Analysis Attacks and Trade-Offs in Anonymity Providing Systems. In *Information Hiding (Lecture Notes in Computer Science)*, Vol. 2137. Springer, 245–247.
[4] Waheed U Bajwa, Jarvis D Haupt, Gil M Raz, Stephen J Wright, and Robert D Nowak. 2007. Toeplitz-structured compressed sensing matrices. In *Statistical Signal Processing*. IEEE, 294–298.
[5] Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. 2008. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation* 28, 3 (2008), 253–263.
[6] Richard G Baraniuk. 2007. Compressive sensing. *IEEE Signal Processing Magazine* 24, 4 (2007).
[7] Alexander Barg and Arya Mazumdar. 2010. Small ensembles of sampling matrices constructed from coding theory. In *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*. IEEE, 1963–1967.
[8] Alexander Barg, Arya Mazumdar, and Rongrong Wang. 2013. Random subdictionaries and coherence conditions for sparse signal recovery. *arXiv preprint arXiv:1303.1847* (2013).
[9] Dror Baron, Marco F Duarte, Michael B Wakin, Shriram Sarvotham, and Richard G Baraniuk. 2009. Distributed compressive sensing. *arXiv preprint arXiv:0901.3403* (2009).
[10] Andy C Bavier, Mic Bowman, Brent N Chun, David E Culler, Scott Karlin, Steve Muir, Larry L Peterson, Timothy Roscoe, Tammo Spalink, and Mike Wawrzoniak. 2004. Operating Systems Support for Planetary-Scale Network Services. In *NSDI*, Vol. 4. 19–19.
[11] Christopher M Bishop. Pattern recognition and machine learning.
[12] Xiang Cai, Xin Cheng Zhang, Brijesh Joshi, and Rob Johnson. 2012. Touching from a distance: Website fingerprinting attacks and defenses. In *ACM CCS*. 605–616.
[13] The CAIDA UCSD Anonymized Internet Traces 2016 - [2016]. http://www.caida.org/data/passive/passive_2016_dataset.xml.
[14] Robert Calderbank, Stephen Howard, and Sina Jafarpour. 2010. Construction of a large class of deterministic sensing matrices that satisfy a statistical isometry property. *IEEE journal of selected topics in signal processing* 4, 2 (2010), 358–374.
[15] Emmanuel J Candès, Justin Romberg, and Terence Tao. 2006. Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on information theory* 52, 2 (2006), 489–509.
[16] Emmanuel J Candes, Justin K Romberg, and Terence Tao. 2006. Stable signal recovery from incomplete and inaccurate measurements. *Communications on pure and applied mathematics* 59, 8 (2006), 1207–1223.
[17] Emmanuel J Candes and Terence Tao. 2005. Decoding by linear programming. *IEEE Transactions on Information Theory* 51, 12 (2005), 4203–4215.
[18] Emmanuel J Candes and Terence Tao. 2006. Near-optimal signal recovery from random projections: Universal encoding strategies? *IEEE transactions on information theory* 52, 12 (2006), 5406–5425.
[19] Emmanuel J Candès and Michael B Wakin. 2008. An introduction to compressive sampling. *IEEE signal processing magazine* 25, 2 (2008), 21–30.
[20] Shuo Chen, Rui Wang, XiaoFeng Wang, and Kehuan Zhang. 2010. Side-channel leaks in web applications: A reality today, a challenge tomorrow. In *IEEE S&P*. IEEE, 191–206.
[21] Heyning Cheng and Ron Avnur. Traffic Analysis of SSL Encrypted Web Browsing. https://pdfs.semanticscholar.org/1a98/7c4fe65fa347a863dece665955ee7e01791b.pdf.
[22] Compressive Imaging: A New Single-Pixel Camera. http://dsp.rice.edu/cscamera.
[23] Baris Coskun and Nasir D Memon. 2009. Online Sketching of Network Flows for Real-Time Stepping-Stone Detection. In *ACSAC*. Citeseer, 473–483.
[24] George Danezis. 2004. The traffic analysis of continuous-time mixes. In *International Workshop on Privacy Enhancing Technologies*. Springer, 35–50.
[25] George Danezis, Roger Dingledine, and Nick Mathewson. 2003. Mixminion: Design of a type III anonymous remailer protocol. In *IEEE S&P*. IEEE, 2–15.
[26] Sanjoy Dasgupta and Anupam Gupta. 2003. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms* 22, 1 (2003), 60–65.
[27] Ronald A DeVore. 2007. Deterministic constructions of compressed sensing matrices. *Journal of complexity* 23, 4-6 (2007), 918–925.
[28] Roger Dingledine, Nick Mathewson, and Paul Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*.
[29] David L Donoho. 2006. Compressed sensing. *IEEE Transactions on information theory* 52, 4 (2006), 1289–1306.
[30] David L Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford. 2002. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *RAID*. Springer, 17–35.
[31] Yonina C Eldar and Gitta Kutyniok. 2012. *Compressed sensing: theory and applications*. Cambridge University Press.
[32] J. Geddes, M. Schuchard, and N. Hopper. 2013. Cover Your ACKs: Pitfalls of Covert Channel Censorship Circumvention. In *CCS*.
[33] Xun Gong, Negar Kiyavash, and Nikita Borisov. 2010. Fingerprinting websites using remote traffic analysis. In *ACM CCS*. ACM, 684–686.
[34] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. 2012. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of computing* 8, 1 (2012), 321–350.
[35] Jamie Hayes and George Danezis. 2016. k-fingerprinting: A robust scalable website fingerprinting technique. *arXiv preprint arXiv:1509.00789* (2016).
[36] Ting He and Lang Tong. 2007. Detecting encrypted stepping-stone connections. *IEEE Transactions on Signal Processing* 55, 5 (2007), 1612–1623.
[37] Dominik Herrmann, Rolf Wendolsky, and Hannes Federrath. 2009. Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier. In *ACM workshop on Cloud computing security*. ACM, 31–42.
[38] Andrew Hintz. 2002. Fingerprinting websites using traffic analysis. In *International Workshop on Privacy Enhancing Technologies*. Springer, 171–178.
[39] Amir Houmansadr and Nikita Borisov. 2011. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *NDSS*.
[40] Amir Houmansadr and Nikita Borisov. 2013. BotMosaic: Collaborative network watermark for the detection of IRC-based botnets. *Journal of Systems and Software* 86, 3 (2013), 707 – 715. https://doi.org/10.1016/j.jss.2012.11.005
[41] Amir Houmansadr and Nikita Borisov. 2013. The need for flow fingerprints to link correlated network flows. In *PETS*. Springer, 205–224.
[42] Amir Houmansadr, Chad Brubaker, and Vitaly Shmatikov. 2013. The Parrot Is Dead: Observing Unobservable Network Communications. In *S&P*.
[43] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2009. RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows. In *NDSS*.
[44] Amir Houmansadr, Negar Kiyavash, and Nikita Borisov. 2014. Non-blind watermarking of network flows. *IEEE/ACM Transactions on Networking (TON)* 22, 4 (2014), 1232–1244.
[45] Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. 2004. *Independent component analysis*. Vol. 46. John Wiley & Sons.
[46] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. ACM, 604–613.
[47] Hao Jiang and Constantinos Dovrolis. 2005. Why is the Internet traffic bursty in short time scales?. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 33. ACM, 241–252.
[48] Ian Jolliffe. 2002. *Principal component analysis*. Wiley Online Library.
[49] Marc Juarez, Sadia Afroz, Gunes Acar, Claudia Diaz, and Rachel Greenstadt. 2014. A critical evaluation of website fingerprinting attacks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 263–274.
[50] Albert Kwon. 2015. *Circuit fingerprinting attacks: Passive deanonymization of tor hidden services*. Ph.D. Dissertation. Massachusetts Institute of Technology.
[51] Shuxing Li, Fei Gao, Gennian Ge, and Shengyuan Zhang. 2012. Deterministic construction of compressed sensing matrices via algebraic curves. *IEEE Transactions on Information Theory* 58, 8 (2012), 5035–5041.
[52] Marc Liberatore and Brian Neil Levine. 2006. Inferring the source of encrypted HTTP connections. In *ACM CCS*. ACM, 255–263.
[53] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. 2009. A new cell counter based attack against tor. In *Proceedings of the 16th ACM*

*conference on Computer and communications security*. ACM, 578–589.
[54] Zhen Ling, Junzhou Luo, Wei Yu, Xinwen Fu, Dong Xuan, and Weijia Jia. 2012. A new cell-counting-based attack against Tor. *IEEE/ACM Transactions on Networking (TON)* 20, 4 (2012), 1245–1261.
[55] Nikolas List and Hans Ulrich Simon. 2009. SVM-optimization and steepest-descent line search. In *Proceedings of the 22nd Annual Conference on Computational Learning Theory*.
[56] Liming Lu, Ee-Chien Chang, and Mun Choon Chan. 2010. Website fingerprinting and identification using ordered feature sequences. In *European Symposium on Research in Computer Security*. Springer, 199–214.
[57] Florence Jessie MacWilliams and Neil James Alexander Sloane. 1977. *The theory of error-correcting codes*. Elsevier.
[58] Mehdi Malboubi, Cuong Vu, Chen-Nee Chuah, and Puneet Sharma. 2013. Compressive sensing network inference with multiple-description fusion estimation. In *Global Communications Conference (GLOBECOM), 2013 IEEE*. IEEE, 1557–1563.
[59] Jiří Matoušek. 2008. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms* 33, 2 (2008), 142–156.
[60] Steven J Murdoch and George Danezis. 2005. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy*. IEEE, 183–195.
[61] A Nir and Edo Liberty. 2009. Fast dimension reduction using Rademacher series on dual BCH codes. *Discrete Computational Geometry* 42, 4 (2009), 615.
[62] Andriy Panchenko, Fabian Lanze, Andreas Zinnen, Martin Henze, Jan Pennekamp, Klaus Wehrle, and Thomas Engel. 2016. Website Fingerprinting at Internet Scale. In *NDSS*.
[63] H Vincent Poor. 2013. *An introduction to signal detection and estimation*. Springer Science & Business Media.
[64] Young June Pyun, Young Hee Park, Xinyuan Wang, Douglas S Reeves, and Peng Ning. 2007. Tracing traffic through intermediate hosts that repacketize flows. In *INFOCOM*. IEEE, 634–642.
[65] Daniel Ramsbrock, Xinyuan Wang, and Xuxian Jiang. 2008. A first step towards live botmaster traceback. In *Recent Advances in Intrusion Detection*. Springer, 59–77.
[66] Holger Rauhut. 2010. Compressive sensing and structured random matrices. *Theoretical foundations and numerical methods for sparse recovery* 9 (2010), 1–92.
[67] Michael K Reiter and Aviel D Rubin. 1998. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security (TISSEC)* 1, 1 (1998), 66–92.
[68] Marc Rennhard and Bernhard Plattner. 2002. Introducing MorphMix: peer-to-peer based anonymous Internet usage with collusion detection. In *Proceedings of the 2002 ACM workshop on Privacy in the Electronic Society*. ACM, 91–102.
[69] David Schneider. 2013. New camera chip captures only what it needs. *IEEE spectrum* 3, 50 (2013), 13–14.
[70] Roei Schuster, Vitaly Shmatikov, and Eran Tromer. 2017. Beauty and the Burst: Remote Identification of Encrypted Video Streams. In *USENIX Security*.
[71] Gregory Shakhnarovich, Piotr Indyk, and Trevor Darrell. 2006. *Nearest-neighbor methods in learning and vision: theory and practice*.
[72] Vitaly Shmatikov and Ming-Hsiu Wang. 2006. Timing analysis in low-latency mix networks: Attacks and defenses. In *European Symposium on Research in Computer Security*. Springer, 18–33.
[73] Stuart Staniford-Chen and L Todd Heberlein. 1995. Holding intruders accountable on the Internet. In *Security and Privacy, 1995. Proceedings., 1995 IEEE Symposium on*. IEEE, 39–49.
[74] D Takhar, V Bansal, M Wakin, M Duarte, D Baron, KF Kelly, and RG Baraniuk. 2006. A compressed sensing camera: New theory and an implementation using digital micromirrors. *Proc. Computational Imaging IV at SPIE Electronic Imaging, San Jose* (2006).
[75] Yaakov Tsaig and David L Donoho. 2006. Extensions of compressed sensing. *Signal processing* 86, 3 (2006), 549–571.
[76] Liang Wang, Kevin P. Dyer, Aditya Akella, Thomas Ristenpart, and Thomas Shrimpton. 2015. Seeing Through Network-Protocol Obfuscation. In *ACM CCS*.
[77] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. 2014. Effective attacks and provable defenses for website fingerprinting. In *USENIX Security*. 143–157.
[78] Tao Wang and Ian Goldberg. 2013. Improved website fingerprinting on tor. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. ACM, 201–212.
[79] Xinyuan Wang, S. Chen, and S. Jajodia. 2005. Tracking Anonymous Peer-to-peer VoIP Calls on the Internet. In *CCS*.
[80] Xinyuan Wang, Shiping Chen, and Sushil Jajodia. 2007. Network flow watermarking attack on low-latency anonymous communication systems. In *IEEE S&P*. IEEE, 116–130.
[81] Xinyuan Wang and Douglas S Reeves. 2003. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *CCS*.
[82] Xinyuan Wang, Douglas S Reeves, and S Felix Wu. 2002. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *ESORICS*. Springer, 244–263.
[83] Yair Weiss, Hyun Sung Chang, and William T Freeman. 2007. Learning compressed sensing. In *Snowbird Learning Workshop, Allerton, CA*.
[84] Andrew M White, Austin R Matthews, Kevin Z Snow, and Fabian Monrose. 2011. Phonotactic reconstruction of encrypted VoIP conversations: Hookt on fon-iks. In *IEEE S&P*. 3–18.
[85] Charles V Wright, Lucas Ballard, Scott E Coull, Fabian Monrose, and Gerald M Masson. 2008. Spot me if you can: Uncovering spoken phrases in encrypted VoIP conversations. In *IEEE S&P*. IEEE, 35–49.
[86] Charles V Wright, Lucas Ballard, Fabian Monrose, and Gerald M Masson. 2007. Language identification of encrypted VoIP traffic: Alejandra y Roberto or Alice and Bob?. In *USENIX Security*.
[87] Allen Y Yang, Zihan Zhou, Yi Ma, and Shankar Sastry. 2010. Towards a robust face recognition system using compressive sensing. In *INTERSPEECH*, Vol. 2010. Citeseer, 11th.
[88] Kunikazu Yoda and Hiroaki Etoh. 2000. Finding a connection chain for tracing intruders. In *ESORICS*. Springer, 191–205.
[89] Wei Yu, Xinwen Fu, Steve Graham, Dong Xuan, and Wei Zhao. 2007. DSSS-based flow marking technique for invisible traceback. In *IEEE S&P*. IEEE, 18–32.
[90] Yin Zhang and Vern Paxson. 2000. Detecting Stepping Stones. In *USENIX Security*, Vol. 171. 184.
[91] Yin Zhang, Matthew Roughan, Walter Willinger, and Lili Qiu. 2009. Spatio-temporal compressive sensing and internet traffic matrices. In *ACM SIGCOMM Computer Communication Review*, Vol. 39. ACM, 267–278.
[92] Ye Zhu and Riccardo Bettati. Unmixing Mix Traffic. In *PETS*. 110–127.

# A RECOVERING COMPRESSED FEATURES

As discussed in Section 3.2, compressive traffic analysis does not need to recover compressed traffic features for correlation, and can directly perform correlation on the compressed features. This is thanks to the RIP property of the utilized linear projection algorithms (as discussed in Section 3.2), which preserves the Euclidean distance of traffic features vectors after compression. One can perform traffic analysis on the reconstructed traffic features as well, i.e., after decompressing. However, since the reconstruction process can be lossy, this will not only increase the computation overhead due to running reconstruction algorithms (which involve solving optimization problems), but also degrade the accuracy of traffic analysis as we confirm through experimentation.

We implement code to reconstruct compressed traffic features, i.e., by solving (2). We particularly, used the CVXOPT[4] optimization tools to solve the involved optimization problem and recover compressed traffic features. Figure 21 compares the performance

[4] http://cvxopt.org/

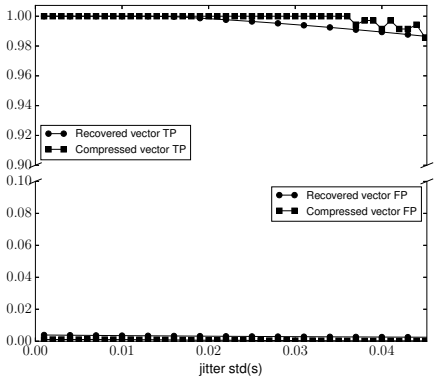


**Figure 21: Performance of compressive traffic analysis with and without recovery.**

of our cosine-based compressive algorithm with and without reconstructing compressed features. As can be seen, constructing features before correlation slightly drops the performance of correlation (due to the noise in recovery) *in addition to making the correlation process much slower* (due to so solving the optimization problem).