The Need for Flow Fingerprints to Link Correlated Network Flows

Amir Houmansadr[1] and Nikita Borisov[2]

[1] The University of Texas at Austin
amir@cs.utexas.edu
[2] University of Illinois at Urbana-Champaign
nikita@illinois.edu

Abstract. Linking network flows is an important problem in the detection of stepping stone attacks as well as in compromising anonymity systems. Traffic analysis is an effective tool for linking flows, which works by correlating their communication patterns, e.g., their packet timings. To improve scalability and performance of this process, recent proposals suggest to perform traffic analysis in an *active* manner by injecting invisible *tag*s into the traffic patterns of network flows; this approach is commonly known as *flow watermarking*. In this paper, we study an *under-explored* type of active traffic analysis that we call it *flow fingerprinting*. Information theoretically, flow watermarking aims at conveying a *single bit* of information whereas flow fingerprinting tries to reliably send *multiple bits* of information, hence it is a more challenging problem. Such additional bits help a fingerprinter deliver extra information in addition to the existence of the tag, such as the network origin of the flow and the identity of the fingerprinting entity. In this paper, we introduce and formulate the flow fingerprinting problem and contrast its application scenarios from that of the well-studied flow watermarking. We suggest the use of coding theory to build fingerprinting schemes based on the existing watermarks. In particular, we design a non-blind fingerprint, Fancy, and evaluate its performance. We show that Fancy can reliably fingerprint millions of network flows by tagging only as few as tens of packets from each flow.

Keywords: Flow fingerprinting, traffic analysis, linear codes, network security

1 Introduction

Linking network flows is an important problem in different applications, including stepping stones detection [13, 28] and compromising anonymity [22, 23]. For instance, it is widely known that attackers can de-anonymize a low-latency anonymity system by linking its egress and ingress flows. Since network flows are commonly encrypted in such applications, linking flows is feasible only through correlation of communication patterns such as packet timings, packet counts, and packet sizes [13, 22, 23]; this is known as *traffic analysis*.

Traditional traffic analysis links network flows *passive*ly, i.e., through observing and correlating patterns inherent in network flows such as packet timings [3,28]. Unfortunately, this suffers from high rates of false positive errors due to the intrinsic correlation that exists among network flows, even if they are not related [13]. For instance, HTTP connections to the same webpage exhibit highly correlated packet timings [14], even if they are initiated independently by individuals residing in different network locations. In response to this, researchers have suggested an *active* approach for traffic analysis. In this approach, communication patterns of network flows are slightly perturbed, e.g., by delaying packets, such that this perturbation is detectable from the flows even after passing through a noisy network like the Internet. The existing designs for active traffic analysis are referred to as *flow watermarks*. A flow watermarking system is composed of *watermarker*s, who tag network flows by perturbing their patterns, and *detector*s, who analyze intercepted flows to identify those carrying the watermark perturbation.

In this paper, we study *flow fingerprinting*, an under-explored[3] variant of active traffic analysis. Flow fingerprinting is similar to flow watermarking in that it *tag*s network flows by slightly perturbing their communication patterns—we call these tags *flow fingerprint*s. Flow fingerprinting, however, differs from flow watermarking in the amount and the kind of information that it embeds: *A watermark tag contains a "single bit", which solely states that the carrying flow has been tagged by "some" tagger (watermarker); on the other hand, a fingerprint tag contains "multiple bits" of information, which convey additional information about the flows being tagged.* In simpler words, for each observed flow a watermark detector only seeks the answer to the following question: "*Has this flow been tagged by any of our watermarking agents?*" A fingerprint extractor, however, looks for additional information about the intercepted flow, such as the network origin of that flow, its relation to other observed flows, the identity of its tagger, and so on. Example questions asked by a fingerprint extractor are: "*Which specific fingerprinter (out of all fingerprinters) tagged this flow?*" "*Which specific flow is related to the observed flow?*" "*In which region of the network has this flow been tagged?*" etc.

The need for flow fingerprints. While flow watermarking and fingerprinting look very similar in how they operate, they have different capabilities and limitations. To illustrate this better, consider the following application scenario. Previous research [22, 23] have designed flow watermarking systems in order to attack anonymity systems by linking their egress and ingress flows. Such watermarks, however, can only conduct a *targeted* attack in this scenario, i.e., they can identify the egress flows corresponding to a "single" target, ingress flow. On the other hand, one might need to perform a *non-targeted* attack in this case, i.e., to simultaneously identify the egress flows corresponding to "many" ingress flows observed; this can not be accomplished through flow watermarking [10, 13, 19, 23, 27] as they do not have efficient mechanisms to confidently

[3] By fingerprinting we mean fingerprinting of traffic patterns, not packet contents. The latter is orthogonal to the problem studied in this paper, and is widely studied.

distinguish among various watermark tags. Performing such a non-targeted attack requires a flow fingerprinting scheme that embeds various tags on different ingress flows, and that is reliably able to extract them from the egress flows. In Section 2 we elaborate more on the differences between flow watermarking and fingerprinting.

A gap in the literature. The literature on active traffic analysis has mainly studied the flow watermarking problem [10,13,19,23,27] while **entirely ignoring flow fingerprinting**. This comes as a surprise given that, in several important applications of traffic analysis, flow fingerprinting *is* indeed the solution to the problem, not watermarking (see Section 2). As fingerprinting aims at the reliable extraction of multiple information bits its design is *more challenging* than flow watermarking, which only conveys a single bit of information. One might design a **naive fingerprinting scheme by having a flow watermarker insert different tags into different flows**. The problem, however, is that the watermark detectors will not able to distinguish among a *large number* of distinct watermarks as they are not designed to do so (they might be able to distinguish within a small set of watermarks though). Note that in several flow watermarking systems [10, 13, 17, 19] a watermark signal is composed of a sequence of numbers, often referred to by the *misleading* term of watermark "bit"s. These watermark bits, however, do not correspond to different "information bits," but they all help to the reliable transmission of a single information bit. To make this more clear, consider the cell-counter attack of [17]. This attack uses a Tor-specific [7] flow watermark that embeds a secret sequence of watermark bits on each Tor flow by modifying its cell counters [6]. In this setting, a single Tor cell delivers a 0 watermark bit and a triplet Tor cells sent together denotes a 1 watermark bit. As network congestion and network delay can separate and merge cells in a circuit, the authors in [17] design a *recovery* mechanism to detect the distorted watermark sequences. While this mechanism is able to *detect* the presence of the watermark sequence on a distorted flow, it is not able to reliably *extract* the watermark bits. For instance, suppose that each watermarker inserts only one of these two watermarks: "010" and "00000". Once a detector receives a flow with a "010" cell pattern, it will return positive correlations against *both* "010" and "00000" watermark sequences due to its recovery mechanism (for "0000" the detector assumes that the triplet cells carrying the 1 bit are split by the Tor routers). In simpler words, the detector can tell, with high assurance, that a flow contains *a* watermark, but it can not tell which one.

As another example, consider RAINBOW [13], which is the basis of our Fancy fingerprint. For a specific set of RAINBOW parameters, i.e., a watermark amplitude of $10ms$ and a watermark length of 500, RAINBOW achieves a false negative rate of around 10^{-6} (Figure 4 of [13]), i.e., it misses only one out of a million watermark tags. For the same parameters, the average correlation between the embedded watermark and the extracted one is about 0.55 (Figure 6 of [13]), meaning that it roughly misses one third of the watermark bits with unknown positions, despite its very high watermark detection ratio. Similarly,

all of the proposed flow watermarks [10, 13, 17, 19] are only able to detect the presence of the watermark, but cannot extract the watermark bits reliably.

Our contributions. In this paper, we introduce and formulate the flow fingerprinting problem and contrast its application scenarios from that of the well-studied flow watermarking. We also design a class of flow fingerprints, called Fancy, that uses a non-blind architecture similar to the RAINBOW [13] watermark. Fancy utilizes communication codes for the reliable extraction of fingerprint bits. We investigate the use of three classes of coding schemes in the design of Fancy, namely block codes, convolutional codes, and turbo codes [21]. We simulate Fancy using real-world network traces and evaluate its fingerprinting performance for different coding algorithms and under different conditions. The simulation results show that it is possible to reliably send dozens of fingerprint bits over very short lengths of network flows. Our methodology in using coding theory can motivate the design of other fingerprints based on existing watermarking systems. In summary, in this paper we make the following main contributions:

- We introduce and formulate the problem of flow fingerprinting, and discuss its necessity in several applications;
- We design the very first flow fingerprinting scheme, called Fancy;
- Through massive simulations on real-world network traces we show the promising reliability of Fancy in fingerprinting network flows and discuss different performance trade-offs.

In this paper, we **do not** study tag invisibility, robustness to active attacks, and similar issues common to flow watermarks. While these issues are important, they have been extensively studied in the watermarking literature. In particular, the invisibility and robustness evaluations of [13, 16] apply to the Fancy fingerprint designed in this paper.

Paper's organization. The rest of this paper is organized as follows: in Section 2, we introduce flow fingerprinting and elaborate on its differences with flow watermarking by mentioning their application scenarios. In Section 3 we describe the design of our proposed flow fingerprinting scheme, Fancy. We design efficient codes for Fancy and evaluate their performance in Section 4. We discuss the related work in Section 5 and the paper is concluded in Section 6.

2 Network Flow Fingerprinting

As described above, active traffic analysis has mainly been studied in the concept of flow watermarking, leaving flow fingerprinting unexplored. In this section, we define the flow fingerprinting problem by describing its components and goals. Then, we discriminate flow fingerprinting from its dual, flow watermarking, by explaining their application scenarios.

2.1 Problem Statement

A fingerprinting system is composed of two main components: *fingerprinter*s and *fingerprint extractor*s. A typical implementation of a fingerprinting system may consist of several fingerprinters and extractors mounted at different network locations. A fingerprinter slightly modifies communication patterns of an observed network flow, e.g., its packet timings, so that it modulates an ℓ-bits *fingerprint tag* into that flow. This ℓ-bits fingerprint conveys some information about the carrying flow, e.g., its network origin, hence it might have different values across different flows. A fingerprinted flow passes through a noisy network, e.g., the Internet, before it is intercepted by a fingerprint extractor who, then, tries to extract its ℓ-bits fingerprint tag. A fingerprint tag should be *robust* to the network noise, i.e., an extractor should be able to extract *all* ℓ bits correctly. Also, as with flow watermarks, a fingerprint should be *invisible*, i.e., an entity not part of the fingerprinting system should not be able to distinguish between a fingerprinted flow and a regular flow.

2.2 Application Scenarios

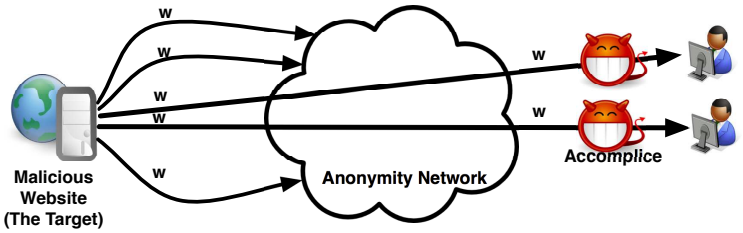
Active traffic analysis is traditionally suggested for two applications: detection of stepping stone attacks [10,13,19], and compromising anonymity systems [17,22]. In the following, we introduce these two applications and discriminate fingerprinting and watermarking in each of these cases.

Compromising Anonymity Systems An anonymity system like Tor [7] maps a number of input flows to a number of output flows while hiding the exact relationships between them. The goal of an attacker, then, is to link an incoming flow to its outgoing flow (or vice versa). Previous research [22,23] has suggested the use of flow watermarks for performing this attack. To do so, an attacker tags the flows entering the anonymity network and watches output flows for the inserted watermark. Such an attack can be performed in two manners, *targeted* and *non-targeted*. As we discuss in the following, flow watermarking is only able to conduct the targeted form of this attack, whereas conducting the non-targeted attack requires flow fingerprinting.

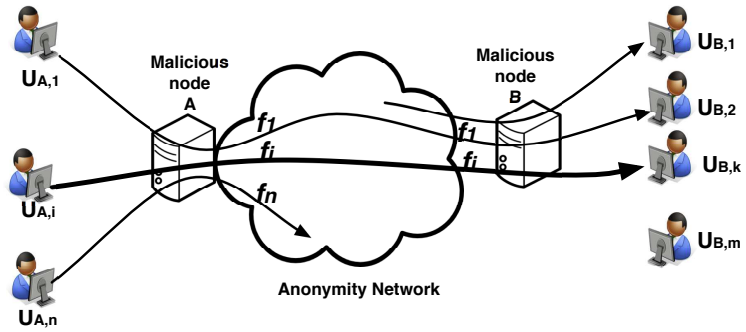
1) Targeted attack Consider a malicious website who intends to identify users who visit that website through an anonymity system (see Figure 1a). To do so, the malicious website inserts a tag on all flows between itself and the anonymizing system. An accomplice who can eavesdrop on a link to the anonymity system (e.g., a malicious Tor entry node, or an ISP) can identify the users browsing the malicious website by looking for the inserted tag. Note that, in this case, the malicious website suffices to insert *the same tag* on any flow that it tags, i.e., it inserts a watermark. This is because the accomplice only needs to check for the existence of the tag, but not its value.

2) Non-targeted attack Now consider a different scenario in which two (or more) compromised/malicious Tor [7] nodes intend to de-anonymize Tor's con-

nections (see Figure 1b). We argue that this application requires flow fingerprinting as a solution. Suppose that the malicious nodes A and B intercept traffic from n and m number of distinct users, respectively. If flow watermarking was used by the attackers, the node A would insert the same tag (i.e., a watermark) on the traffic of all of its n users, and the node B would look for that single watermark on the traffic of all of its m users. In this case, if B detects the watermark on the traffic of one of its m users, namely $U_{B,k}$, the attackers can only infer that $U_{B,k}$ is communicating with *one of* the n users observed by A, but they can not tell with which of them. Alternatively, suppose that flow fingerprinting is used by the attackers. In this case, A inserts a different, customized tag (i.e., a fingerprint) on the traffic of each of its n users, e.g., it inserts the fingerprint f_i on the traffic of user $U_{A,i}$. Now, if B observes that the traffic to one of its users, $U_{B,k}$, contains the fingerprint f_i the attackers can infer that users $U_{A,i}$ and $U_{B,k}$ are communicating through the anonymity system.



(a) Targeted attack



(b) Non-targeted attack

Fig. 1: Target and non-targeted attacks on an anonymous network.

Stepping Stone Detection A stepping stone is a host that is used to relay an attacking traffic to its victim destination, in order to hide the true origin of the attack. To defend, an enterprise network should be able to identify the ingress flows that are linked (correlated) with some egress flow. The situation is therefore

very similar to an anonymous communication system, with n flows entering the enterprise and m flows leaving. There are two objectives for active traffic analysis in this case, as described in the following; the first one is achievable using flow watermarking while the second one requires flow fingerprinting.

1) Detecting relayed flows As previous research [13, 19] suggests, flow watermarks can be used to detect relayed network flows in this scenario. Suppose that the enterprise network consists of two border routers A and B. To do so, the border router A inserts a watermark tag w on all flows that enter the enterprise network. On the other side, the border router B inspects all egress network flows, looking for the watermark w. Suppose that A intercepts n flows and B intercepts m network flows at a given time. If B detects that a network flow $F_{B,k}$ is carrying the watermark tag w, the security officer of the enterprise network infers that $F_{B,k}$ is a traffic relayed through the enterprise. However, the security officer can not tell which of the n flows observed by A is the source of $F_{B,k}$, since A inserts the same watermark tag on all intercepted flows.

2) Detecting relayed flows and their origins Flow fingerprinting can be used to not only detect the relayed flows, but also identify their sources. Consider the case in which the border router A inserts different tags (i.e., fingerprints) on each of the n intercepted flows (that is, the fingerprint f_i is inserted into the i-th flow, $F_{A,i}$). Now, suppose that B detects the fingerprint f_i on the network flow $F_{B,k}$. In this case, the security officer infers two facts: first, $F_{B,k}$ is a relayed traffic and, second, the source of this relay traffic is the network flow $F_{A,i}$. A watermark, however, is not able to identify the source of the relayed traffic.

3 Fancy Fingerprinting Scheme

In this section, we describe the design of our flow fingerprinting system, Fancy. Fancy consists of two main elements: a *fingerprinter* that embeds fingerprint messages inside intercepted flows by slightly modifying their timing patterns, and a *fingerprint extractor* (extractor in short) that analyzes the timing patterns of the intercepted flows, trying to extract the fingerprint messages. Fancy uses a *non-blind* architecture similar to the RAINBOW watermark [13]: the fingerprinter communicates with fingerprint extractors some information about the flows being fingerprinted, which is required for efficient fingerprint extraction. To perform this communication, Fancy uses a third element in its design, *IPDs registrar*, which is accessible by fingerprinters and fingerprint extractors. A Fancy fingerprinter stores some information about the intercepted flows on the IPDs registrar, which is periodically accessed by Fancy extractors. Figure 2 shows the high-level block diagram of Fancy.

3.1 Embedding Fingerprints

Figure 3 illustrates a Fancy fingerprinter. Suppose that a network flow, n, with packet timings of $\boldsymbol{t} = \{t_i | i = 1, ...\}$ enters the fingerprinter (e.g., a router) where it is to be fingerprinted. The fingerprinter generates an ℓ-bits fingerprint message,

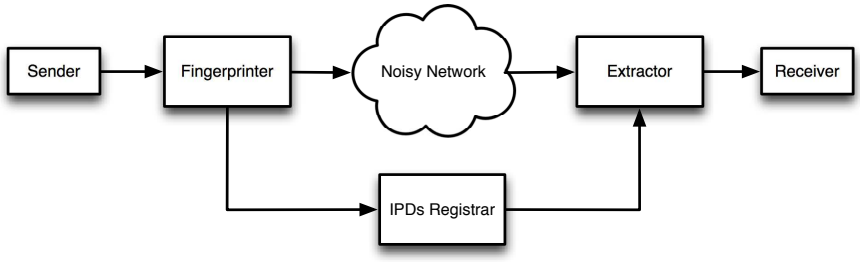


Fig. 2: The main model of Fancy fingerprinting system.

$\boldsymbol{f} = \{f_i | f_i = \pm 1, i = 1, .., \ell\}$, that especially corresponds to the intercepted flow n. That is, a different fingerprint sequence is generated for each intercepted flow, however a fingerprint sequence can be re-used for another flow once the first flow has terminated. This fingerprinter records n's fingerprint message, $\boldsymbol{f}$, along with the last ℓ^c inter-packet delays (IPDs) of n, i.e., $\boldsymbol{\tau} = \{t_i | \tau_i = t_{i+1} - t_i, i = 1, .., \ell^c\}$, in the IPDs registrar (ℓ^c is the length of the fingerprinted flows, as described later). In addition to recording the flow's fingerprint in the IPDs registrar, the fingerprinter embeds $\boldsymbol{f}$ on the intercepted flow n, as described in the following (in Section 3.3, we discuss the reason for both embedding the fingerprint into the flow and recording it in the IPDs registrar).

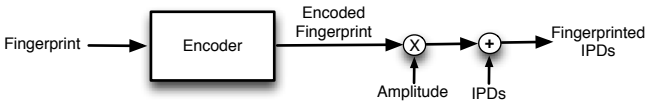


Fig. 3: Fingerprinting scheme of Fancy.

The fingerprinter embeds the fingerprint $\boldsymbol{f}$ into the intercepted flow n by delaying its packets by an amount such that the IPD of the i-th fingerprinted packet is

$$\tau_i^c = \tau_i + a \cdot f_i^c \tag{1}$$

The constant a is the *fingerprint amplitude* and $\boldsymbol{f^c} = \{f_i^c | f_i^c = \pm 1, i = 1, ..., \ell^c\}$ is the *encoded fingerprint* sequence, which is generated from $\boldsymbol{f}$ as described in the following. The value a is chosen to be small enough so that the artificial jitter caused by fingerprinting is invisible to non-fingerprinting parties and to the users.

Fingerprint generation: Suppose that Fancy intends to insert an ℓ-bits fingerprint $\boldsymbol{f} = \{f_i | i = 1, ..., \ell\}$ on a candidate flow. Since each fingerprint bit takes one of the +1 and -1 values, the number of all distinct fingerprint sequences is 2^ℓ. A fingerprinter encodes an ℓ-bits fingerprint sequence, $\boldsymbol{f}$, into an ℓ^c-bits encoded fingerprint $\boldsymbol{f}^c$ by passing $\boldsymbol{f}$ through the encoder block of Fancy fingerprinter.

For a given encoding algorithm, we define the *redundancy* of our fingerprinting system, r, to be the redundancy of the utilized encoder, i.e.,

$$r = \ell^c/\ell. \tag{2}$$

3.2 Extracting Fingerprints

Suppose that a Fancy extractor receives the fingerprinted flow n after passing the noisy network, e.g., the Internet. Let us consider that $\boldsymbol{\tau}^{c,r} = \{\tau_i^{c,r} | i = 1, .., \ell^c\}$ are the IPDs of n as observed by the extractor (the superscript c denotes being encoded/fingerprinted and the superscript r denotes being received after passing the noisy network). As described above, the Fancy fingerprinter has recoded n's IPDs before getting fingerprinted, i.e., $\boldsymbol{\tau}$, in the IPDs registrar, along with the embedded fingerprint message $\boldsymbol{f}$. The extractor uses this recorded information to perform fingerprint extraction by accessing the IPDs registrar.

The IPDs registrar contains *flow record*s, which are generated by fingerprinters. Each flow record is a pair $R_k = (\boldsymbol{\tau}^k, \boldsymbol{f}^k)$, where k is the index of the record in the IPDs registrar, $\boldsymbol{\tau}^k$ is the original IPDs sequence of a fingerprinted flow, and $\boldsymbol{f}^k$ is the fingerprint embedded into that flow. For each received flow, the extractor loops through the IPDs registrar to find the right flow record corresponding to it (if any).

For any flow record in the IPDs registrar, e.g., $(\boldsymbol{\tau}^k, \boldsymbol{f}^k)$, the extractor performs the following steps:

1. The extractor derives the following sequence:
$$f_i^{r,k} = (\tau_i^{c,r} - \tau_i^k)/a \qquad i = 1, .., \ell^c \tag{3}$$
where $\tau_i^{c,r}$ is the i-th IPD of the received flow.
2. Then, the extractor passes the ℓ^c-bits $\boldsymbol{f}^{r,k} = \{f_i^{r,k} | i = 1, .., \ell^c\}$ through a *decoder* block that outputs an ℓ-bits sequence $\boldsymbol{f}^{d,k} = \{f_i^{d,k} | f_i^{d,k} = \pm 1, i = 1, .., \ell\}$. This decoder is the corresponding decoder for the encoder block used by Fancy fingerprinter.
3. The extractor declares that the received flow contains the fingerprint sequence $\boldsymbol{f}^k$ if it is the same as the decoder's output, i.e., if we have that:
$$f_i^{d,k} = f_i^k, \qquad \forall i \in \{1, .., \ell\} \tag{4}$$
4. If (4) does not hold the extractor uses the next flow record from the IPDs registrar and repeats the steps above until the end of the database.

We claim that the described algorithm results in reliable extraction of Fancy fingerprints. Let us consider the following two cases:

Case 1 (True match): Suppose that the extractor has picked the flow record that corresponds to the received flow, i.e.,$\boldsymbol{\tau}^k = \boldsymbol{\tau}$ and $\boldsymbol{f}^k = \boldsymbol{f}$. We have that

$$\tau_i^{c,r} = \tau_i^c + \delta_i \tag{5}$$

where δ_i is the network jitter applied to the i-th IPD and τ_i^c is the fingerprinted IPD. In this case, using (1), (5), and (3) we get that:

$$f_i^{r,k} = (\tau_i^{c,r} - \tau_i^k)/a \tag{6}$$
$$= (\tau_i + \delta_i + a f_i^c - \tau_i)/a \tag{7}$$
$$= f_i^c + \delta_i/a \tag{8}$$

In other words, using the right flow record from the registrar the step 1 of our extraction process will generate a perturbed version of the coded fingerprint. As a result, passing this perturbed coded fingerprint through the decoder (step 2) is likely to return the embedded fingerprint, depending on the noise conditions and the decoder performance (in Section 4 we design decoders that perform very well in our application). Since this decoded fingerprint is the same as the one contained in the flow record, step 3 of the algorithm will result in a correct fingerprint extraction.

Case 2 (False match): Now, let us consider a case where the extractor is using a non-relevant flow record from the registrar. In this case, the second step of the above algorithm will result in an arbitrary fingerprint sequence. The odds that this arbitrary fingerprint is the same as the one contained in the flow record is $2^{-\ell}$. For the values of ℓ used in our design (e.g., 25) this results in a tiny false extraction rate, so we neglect this case in our performance evaluation. Note that the overall probability for a received flow getting false matched with some flow record is $2^{-\ell} \times Y$, where Y is the number of flow records matched against from the IPDs registrar; the following filtering process minimizes Y.

Filtering flow records: In order to speed up the extraction process, also to decrease the possibilities of false matches, for a received flow the extractor leaves out a large number of flow records in the registrar by using simple filters. One very simple, yet efficient, filter is the packet count of the registered flows: the extractor does not consider flow records whose corresponding flows have packet counts much larger or much smaller than the packet count of the received flow. A second filter that we use is the *IPDs-distance* metric, which leaves out the records with non-similar IPDs. We define the IPDs-distance metric between a received flow with IPDs $\boldsymbol{\tau}^r$ and a flow record $R_k = (\boldsymbol{\tau}^k, \boldsymbol{f}^k)$ to be:

$$d(\boldsymbol{\tau}^r, \boldsymbol{\tau}^k) = \frac{1}{\ell^c} \sum_{i=1}^{\ell^c} (\tau_i^r - \tau_i^k - f_i^c) \tag{9}$$

If R_k is the right record corresponding to the received flow this distance metric will return the average network jitter on the path, which is a small number. The extractor considers only those records from the registrar whose distance from the received flow are smaller than a threshold, η. By putting η equal to four times the standard deviation of the network jitter the odds that the right record is left out is approximately 10^{-4} (we model the network jitter as Laplace distribution [13]).

3.3 Alternative Designs

As described above, to fingerprint a flow n with the fingerprint message $\boldsymbol{f}$, the fingerprinter performs two tasks: a) it records the fingerprint $\boldsymbol{f}$ in the IPDs registrar, along with the IPD values of the flow, and, b) it embeds $\boldsymbol{f}$ into the network flow n. Alternatively, one could suggest to only record the fingerprint in the registrar, or to only embed it into the flow. In the following, we back our design decision by discussing the performance degradation of the two alternatives.

Passive fingerprinting An alternative approach to Fancy is to only record fingerprint sequences in the IPDs registrar along with their corresponding IPDs sequences, *without* embedding the fingerprints into the flows. In fact, this approach is passive traffic analysis, which has extensively been studied in the literature [3, 9, 25]. Unfortunately, this approach may result in high rates of false detection, especially when the evaluated flows are cross-correlated. The common examples for correlated network flows are web traffic (to the same destination), and file transfers. To validate this, we simulate an optimum passive traffic analysis scheme [14] on real web traffic that are generated by different users to the same websites. Our results (Table 1) show that even this optimum passive traffic analysis scheme produces very large false positive errors in linking the correlated web traffic.

Table 1: False positive error rates of the optimum passive traffic analysis [14] in linking web traffic that are generated by different users to the same websites (N is the flow length).

Website	N=25	N=50	N=100
baidu.com	0.29	0.07	0.08
blogger.com	0.97	0.63	1
facebook.com	0.91	0.97	0.96
live.com	1	1	0.38
wikipedia.org	0.94	0.44	0.46
yahoo.co.jp	0.66	0.33	0.05
yahoo.com	1	1	0.23
yandex.com	0.89	0.08	0.02

Only embedded into flows As another alternative, one could only embed fingerprints in network flows *without* recording them in the IPDs registrar. This, also, results in high rates of false extraction errors. For a received flow at the extractor, the use of a non-corresponding flow record from the registrar will most likely lead the extractor to retrieve some *valid* fingerprint sequence that is different from the embedded fingerprint. By recording the fingerprints in the registrar as well the extractor can detect this by simply comparing the extracted fingerprint with the one recorded in the IPDs registrar.

4 Code Design and Simulations

We investigate the use of different coding algorithms as the encoder/decoder block of Fancy. In particular, we investigate the use of several linear block codes [21] considering our communication channel. Based on our measurements over Planetlab [1] the standard deviation of network jitter (δ) between randomly selected nodes varies between $6ms$ and $12ms$. For a fingerprinting amplitude of $a = 10ms$, the SNR [2], given by $SNR = 20log(a/\delta)$, varies between -1.5836 and 4.4370 (i.e., an average of 1.4267). Also, we aim at having a flow length of around $n \approx 100$ for fingerprinting, since larger lengths would take longer to extract. For these parameters, we look for appropriate coding algorithms to be used by Fancy's encoder. Dolinar et al. [8] compare the performance of several block codes for different lengths of information bits, along with the theoretical capacity limits. In particular, they illustrate the appropriate block size values for different coding algorithms, i.e., the range of block sizes that a coding algorithm performs close enough to the channel capacity. Based on such evaluations (Figure 12 in [8]) we identify several codes that are expected to work well for our system parameters. In particular, we investigate the use of three types of linear codes in our simulations, which are Reed-Solomon (RS) Codes, convolutional codes, and turbo codes. The simulations are done in Matlab using network traces gathered over Planetlab [1], and by using Matlab's built-in coding functions and the CML coding library [4].

Evaluation metrics – We define the following metrics to evaluate the extraction performance of Fancy.

- **Extraction Rate (P_E):** This metric is the ratio of the number of fingerprinted flows successfully extracted by a Fancy extractor to the number of all fingerprinted flows.
- **Miss Rate (P_M):** This is the ratio of the number of fingerprinted flows declared as non-fingerprinted by a Fancy extractor to the number of all fingerprinted flows. We have that $P_M = 1 - P_E$.

The goal of a Fancy extractor is to maximize the extraction rate (i.e., minimize the miss rate).

4.1 Reed-Solomon (RS) Codes

Reed-Solomon (RS) codes [21] are a class of linear block codes that are maximum distance separable (MDS), i.e., they meet the equality criteria of the *singleton bound* [21]. In fact, the RS codes are the only known instances of the MDS codes. The encoding structure of the RS codes makes them suitable for M-ary communication schemes where the noise is applied in bursts over a message bit stream (e.g., satellite communications). This makes them a good candidate for applications where bursty noise may happen to inter-packet delays, e.g., due to network congestion. We use the notation (n, k)-RS for an RS code that encodes each k message symbols into n encoded symbols, where each symbol is m bits

and $m = \log_2(n+1)$ (e.g., an n-bit RS coded message consists of $m \times n$ binary bits).

We design a Fancy fingerprint, called Fancy-RS, that utilizes RS encoders as part of its encoding algorithm. More specifically, Fancy-RS generates an ℓ^c-bits coded fingerprint $\boldsymbol{f}^c$ from an ℓ-bits fingerprint sequence $\boldsymbol{f}$ by passing $\boldsymbol{f}$ through an (n,k)-RS encoder. We have that $\ell^c = n/k\ell$.

We simulated Fancy-RS in Matlab. We use traces of network jitter gathered over Planetlab [1] to simulate the effect of the noisy network over the fingerprinting performance. Note that we do not include the original IPDs in our simulations, since as discussed in Section 3.2 the extractor is able to reliably pick the original IPD sequence from the IPDs registrar and subtract it from the received flow before performing the extraction. In the first experiment, we measure and compare the performance of our fingerprint extractor for different parameters of the (n,k)-RS encoder. We set $a = 10ms$ and $p = 2$. We also set $\ell = mk$ (generally, ℓ should be an integer multiplication of mk) and vary the m and k parameters of our RS encoder (each experiment is run for 1000 times with different randomly generated fingerprints and different network jitter). Figure 4 shows the extraction rate (P_E) for different values of m and k (the bars show the 95% confidence intervals). As can be seen, for a given m, decreasing k improves the extraction performance since it increases the redundancy of our RS encoder, i.e., $(2^m - 1)/k$. Figure 5 shows the coding redundancy of Fancy-RS for different parameters of the RS code.

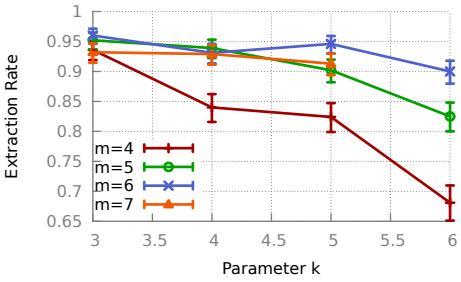


Fig. 4: Extraction rate of Fancy-RS for different RS encoders. ($a = 10ms$, and $\ell = mk$)

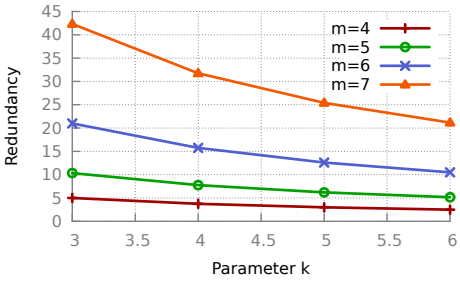


Fig. 5: Coding redundancy of Fancy-RS for different RS encoders. ($a = 10ms$, $p = 2$, and $\ell = mk$)

Finally, the number of distinct fingerprints, N, that can be embedded and extracted reliably by Fancy-RS is given by

$$N = 2^{mk} \tag{10}$$

For instance, for $k = 5$, and $m = 5$ (i.e., $\ell = 25$) we have that $N \approx 10^7$.

In order to evaluate the effect of the fingerprint amplitude (a), we measure the extraction rate for different values of a. This is illustrated in Figure 6, where

$m = 5$, and $k = 5$. As intuitively expected, increasing a rapidly improves the true detection and miss rate such that for $a = 20ms$ we have that $P_T = 1$ and $P_M = 0$. Note that increasing the fingerprint amplitude a makes the fingerprint less invisible, as discussed in [13].

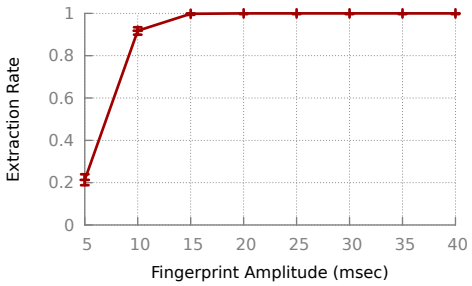


Fig. 6: Extraction rate of Fancy-RS for different fingerprint amplitudes. ($m = 5$, $k = 5$, i.e., the redundancy is 6.2)

4.2 Convolutional Codes

Convolutional codes are another class of linear error-correcting codes that have use in several different applications [21]. An (n, k) convolutional code, (n, k)-Conv, is a function with k inputs and n outputs. The input stream, i.e., $\boldsymbol{f} = \{f_i | f_i \in \{0, 1\}, i = 1, 2, ...\}$, is split into k streams entering the inputs of the encoder. Each of the n output streams of this encoder is evaluated by convolving some of the input streams with a generator function $\boldsymbol{G}$. The length of the generator function is called the *constraint length* v, and $u = v - 1$ is the memory of the encoder. An easy-to-implement decoder for convolutional codes is an ML decoder based on the Viterbi algorithm [21].

We design a variant of Fancy fingerprint, called Fancy-Conv, that uses convolutional coding for its encoding process. More specifically, an ℓ-bits fingerprint sequence $\boldsymbol{f}$ passes through the (n, k)-Conv encoder of Fancy-Conv, which generates the final encoded fingerprint $\boldsymbol{f}^c$ consisting of ℓ^c bits.

We implemented Fancy-Conv in Matlab using the CML [4] coding libraries. We use a constraint length of $v = 9$, and a randomly created generator function G. We run several experiments to measure the performance of Fancy-Conv for different parameters, where each experiment is run for 1000 randomly generated fingerprints. In the first experiment, we measure the effect of our encoder's redundancy on the fingerprinting performance, where $a = 10ms$, and $\ell = 24$. Figure 7 shows the extraction rate (with bars showing the 95% confidence intervals) for different redundancies of (n, k)-Conv. As can be seen, increasing the redundancy, r, improves the extraction rate; this, however, increases the flow length required to embed the fingerprint, which is linear with redundancy, i.e., $\ell^c = r\ell$.

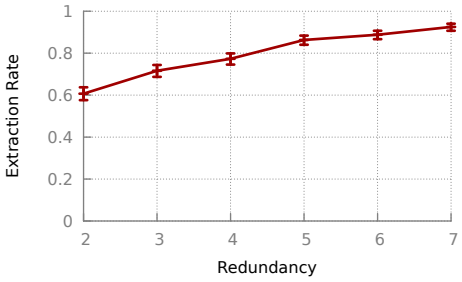


Fig. 7: Extraction rate of Fancy-Conv for different encoder redundancies. ($a = 10ms$)

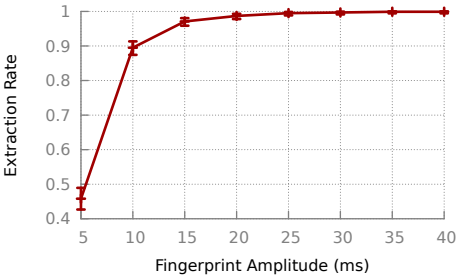


Fig. 8: Extraction performance of Fancy-Conv for different fingerprint amplitudes. ($\ell = 18$, and the convolutional code's redundancy is 8)

We also measure the effect of the fingerprint amplitude on the detection performance. As can be seen from Figure 8, increasing the fingerprint amplitude improves the extraction rate ($\ell = 18$, and the convolutional code's redundancy is 8). This comes at the price of less fingerprint invisibility. A fingerprint amplitude of $a = 20ms$ results in a very good extraction rate, while at the same time provides a promising invisibility.

Finally, the number of distinct fingerprints, N, that can be embedded and extracted reliably by Fancy-Conv is given by $N = 2^{\ell}$. For instance, for $\ell = 24$, we have that $N \approx 10^7$.

4.3 Turbo Codes

Turbo codes are a class of high-performance error correction codes and are the first practical capacity-approaching codes [18]. A turbo code is generated by concatenating two or more *constituent* codes, where each constituent code can be a convolutional or a block code. Usually some *interleaver* reorders the data at the input of the inner encoders. Turbo codes are decoded through iterative schemes. There are two types of Turbo codes: *Block Turbo Codes* (BTC), and *Convolutional Turbo codes* (CTC).

In this paper, we consider the use of BTC codes in the design of Fancy fingerprints. A BTC code works by encoding a $k_x \times k_y$ matrix of data, D, into a $n_x \times n_y$ matrix C as follows: a (n_x, k_x) systematic code encodes each row of D, a block interleaver reorders the rows of the resulted matrix, and finally, a (n_y, k_y) systematic code encodes the columns of the resulted matrix to generate the final $n_x \times n_y$ dimensional matrix C. The systematic codes used for BTC codes are the cyclic codes, e.g., Hamming, Single Parity Check, and Extended Hamming [18]. The constituent block codes can be generated using polynomials.

We design Fancy-BTC, a fingerprint that uses BTC codes as its encoding block. Our BTC code uses two convolutional codes as its horizontal and vertical constituent codes. We use the CML [4] coding library to simulate Fancy-BTC in

Matlab. We randomly create the generator functions of the convolutional codes that constitute our BTC encoder (with constraint lengths of v_x and v_y for the horizontal and vertical codes, respectively). To encode a fingerprint $\boldsymbol{f}$ with length ℓ our encoder reorders $\boldsymbol{f}$ into a $k_x \times k_y$ matrix D, where $k_x = k_y = round(\sqrt{\ell})$ and $round(\cdot)$ rounds to the nearest larger number. The encoder also fills the first $B = k_x \times k_y - \ell$ bits of this matrix with zeros. We use an iterative detector that stops only if either a maximum number of iterations has reached, or the additional iterations do not change the decoded fingerprint.

In our first experiment, we measure the effect of the number of iterations on the extraction performance (each experiment is run for 1000 randomly generated fingerprints). We use a fingerprint amplitude of $a = 10ms$, a fingerprint length of $\ell = 25$, and our code is designed such that $v_x = v_y = 6$, $k_x = k_y = 5$, $B = 0$ and the code redundancy is 5.95. Figure 9 shows the extraction rate for different values of the maximum decoder iteration, along with the 95% confidence intervals. As can be seen, even though the detection performance improves rapidly for small numbers of iterations it does not change significantly after several iterations. Considering the added processing overhead for more iterations, we choose 8 as the maximum number of iterations performed by our detector, being used in all of our consecutive simulations.

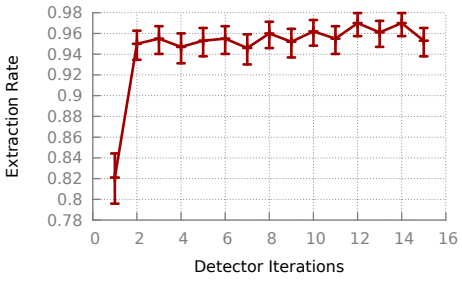


Fig. 9: Extraction rate of Fancy-BTC for different values of maximum decoder iteration. ($a = 10ms$, $\ell = 25$, and $r = 5.95$)

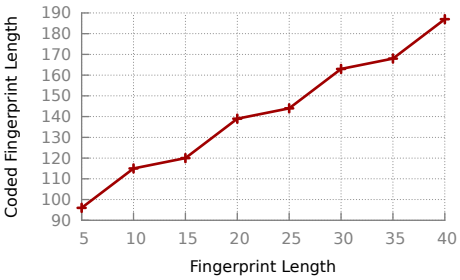


Fig. 10: The length of the coded fingerprint for different fingerprint lengths for Fancy-BTC.

In the second experiment, we keep $v_x = v_y = 6$ and $a = 10ms$, but vary the fingerprint length ℓ. Figure 10 shows the length of the coded fingerprint for different values of the fingerprint length. As before, the number of distinct fingerprints, N, is exponential with ℓ. As can be seen from the figure, increasing ℓ only linearly increases the length of the encoded fingerprint, while it exponentially increases N. Note that in the figure the code redundancy is around 6, but varies a bit with ℓ since our BTC encoder can not produce all redundancy values for any given ℓ.

In the third experiment, we evaluate the performance of Fancy-BTC using BTC codes with different redundancies. More specifically, we set $a = 10ms$,

$\ell = 25$, and $k_x = k_y = 5$ and try BTC codes with different constraint lengths (v_x and v_y), resulting in various redundancies. Figure 11 shows the extraction rate for codes with different redundancies. As can be seen, Fancy-BTC does not perform well for small values of code redundancies, however increasing the code redundancy rapidly improves its performance. In fact, such an improved performance comes at the price of longer fingerprinted flows.

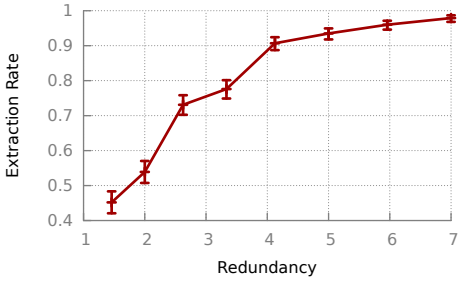


Fig. 11: Extraction rate of Fancy-BTC for different redundancies of the BTC code. ($a = 10ms$, $\ell = 25$)

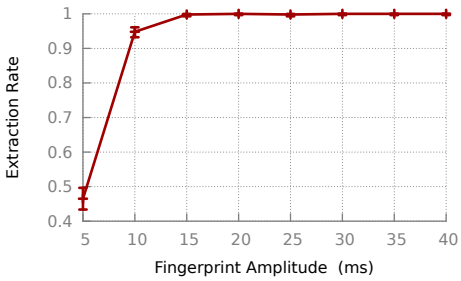


Fig. 12: Extraction rate of Fancy-BTC for different values of fingerprint amplitude. ($\ell = 25$, and $r = 4$)

Finally, we illustrate the effect of the fingerprint amplitude on the extraction performance. As can be seen from Figure 12, increasing a rapidly improves the extraction, such that $a = 20ms$ results in an extraction rate of in 1000 runs of the experiment.

4.4 Comparison

We also compare the performance of the three versions of Fancy, i.e., Fancy-RS, Fancy-Conv, and Fancy-BTC. Figure 13 shows the extraction rate of the three schemes for different values of encoder redundancy (for all three schemes we have that $\ell = 25$, and $a = 10ms$). As can be seen, for very small redundancies the Fancy-RS outperforms the other two, even though all of the schemes perform poorly for such small values of redundancy. As the redundancy increases, all schemes improve their performance and, in particular, the Fancy-BTC outperforms the other two schemes for high redundancies.

We also compare the three schemes for different fingerprint amplitudes. Figure 14 shows the extraction performance of Fancy-RS, Fancy-Conv, and Fancy-BTC for different values of a, with $\ell = 25$. Also, the redundancies of Fancy-RS, Fancy-Conv, and Fancy-BTC are 6.2 , 6, and 5.96, respectively (note that it is not possible to produce an exact value of r for any given ℓ). As intuitively expected, increasing the fingerprint amplitude significantly improves the extraction performance at the cost of larger perturbations applied to the fingerprinted flows.

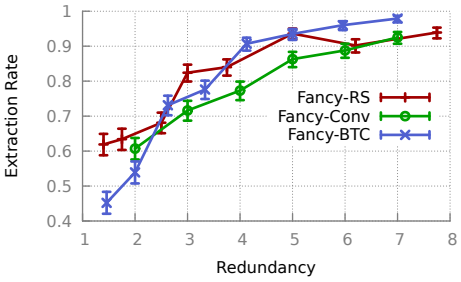


Fig. 13: Comparing the extraction performance of Fancy-RS, Fancy-Conv, and Fancy-BTC for different code redundancies.

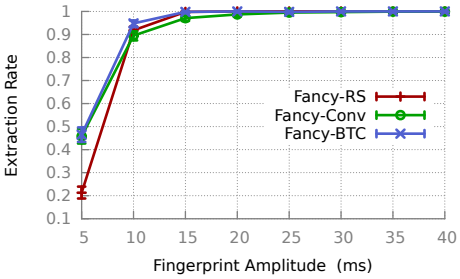


Fig. 14: Comparing the extraction performance of Fancy-RS, Fancy-Conv, and Fancy-BTC for different fingerprint amplitudes (with a similar redundancy of around 6).

5 Related Work

Traffic analysis has long been studied for linking network flows. Early traffic analysis schemes [5, 9, 20, 24, 26, 28] work in a passive manner, i.e., they record the communication characteristics of incoming flows and correlate them with that of the observed outgoing flows. The right place to do this is often at the border router of an enterprise, so the overhead of this technique is the space used to store the stream characteristics long enough to check against correlated relayed streams, and the CPU time needed to perform the correlations. In a complex application with many interconnected networks, a relayed connection, e.g., through a stepping stone, may enter and leave the monitored network through different points; in such cases, there is also an additional communications overhead for transmitting traffic statistics between the border routers.

To address some of the efficiency concerns of passive traffic analysis, researchers have suggested to perform traffic analysis in an active manner. Active traffic analysis improves upon passive traffic analysis in two ways. First, by inserting a pattern that is uncorrelated with any other flows, they can improve the detection efficiency, requiring smaller numbers of packets to be observed (hundreds instead of thousands) and providing lower false-positive rates (10^{-4} or lower, as compared to 10^{-2} with passive watermarks) [14]. Second, they can operate in a *blind* fashion [25]: after an incoming flow is watermarked, there is no need to record or communicate the flow characteristics, since the presence of the watermark can be detected independently. The detection is also potentially faster as there is no need to compare each outgoing flow to all the incoming flows within the same time frame.

Wang et al. borrowed the QIM watermarking idea from the multimedia literature to perform active traffic analysis [25]. This approach, however, is fragile to packet-level modifications, e.g., a single dropped packet would completely destroy the watermark pattern. To provide robustness to such packet-level perturbations several proposals apply the watermark modifications on the timing

"intervals" of network flows [19,23,27]. Kiyavash et al. [12,15] demonstrated that applying the interval-based watermarks identically on different flows can give away the embedded watermark to an attacker who observes several watermarked flows. This affected the design of the successor watermarking schemes [10, 13], which apply the watermark patterns to network flows depending on the features of the candidate flow. Note that such schemes do not insert different watermark tags on different flows, but they apply the same watermark tag differently on different flows. Houmansadr et al. [11] use repeat-accumulate codes to improve the detection performance of watermarks.

While most of the proposals for flow watermarking use timing patterns for their modifications, other traffic patterns are also used for watermark insertion. For instance, Yu et al. [27] design a watermarking system that tags flows by modifying their packet rates in different time intervals. As another example, Ling et al. [17] propose a watermarking attack on Tor [7] that works by modifying packet sizes. The attack works by modifying the counts of Tor cells [6] carried by network packets.

6 Conclusions

In this paper, we shed light on an unexplored, yet important, variant of active traffic analysis, flow fingerprinting. We designed the first flow fingerprinting scheme, Fancy, and demonstrated its reliability in tagging large numbers of distinct flows. We explored the use of different linear codes in the design of Fancy and compared their performance. In particular, we showed that Fancy can reliably tag millions of distinct network flows using flows as short as tens of packets.

Acknowledgements

The authors would like to thank Roger Dingledine and anonymous reviewers for their insightful comments. This work was supported in part by the National Science Foundation grant CNS 0831488 and the Boeing Trusted Software Center at the Information Trust Institute at the University of Illinois.

References

1. A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating Systems Support for Planetary-Scale Network Services. In *NSDI*, 2004.
2. S. Benedetto and E. Biglieri. *Principles of Digital Transmission: With Wireless Applications*. Information Technology: Transmission, Processing, and Storage. Kluwer Academic/Plenum Press, 1999.
3. A. Blum, D. X. Song, and S. Venkataraman. Detection of Interactive Stepping Stones: Algorithms and Confidence Bounds. In *RAID*, 2004.
4. The coded modulation library (cml). http://www.iterativesolutions.com/Matlab.htm.

5. G. Danezis. The Traffic Analysis of Continuous-Time Mixes. In *PETS*, 2004.
6. R. Dingledine and N. Mathewson. Tor Protocol Specification. https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt.
7. R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *USENIX Security Symposium*, 2004.
8. S. Dolinar, D. Divsalar, and F. Pollara. Code Performance as a Function of Block Size. Technical report, TMO Progress, 1998.
9. D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. Multi-scale Stepping-stone Detection: Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. In *RAID*, 2002.
10. A. Houmansadr and N. Borisov. SWIRL: A Scalable Watermark to Detect Correlated Network Flows. In *NDSS*, 2011.
11. A. Houmansadr and N. Borisov. Towards Improving Network Flow Watermarks using the Repeat-accumulate Codes. In *ICASSP*, 2011.
12. A. Houmansadr, N. Kiyavash, and N. Borisov. Multi-Flow Attack Resistant Watermarks for Network Flows. In *ICASSP*, 2009.
13. A. Houmansadr, N. Kiyavash, and N. Borisov. RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows. In *NDSS*, 2009.
14. A. Houmansadr, N. Kiyavash, and N. Borisov. Non-blind Watermarking of Network Flows. CRR, arXiv:1203.2273v1, 2012.
15. N. Kiyavash, A. Houmansadr, and N. Borisov. Multi-Flow Attacks Against Network Flow Watermarking Schemes. In *USENIX Security Symposium*, 2008.
16. Z. Lin and N. Hopper. New Attacks on Timing-based Network Flow Watermarks. In *USENIX Security*, 2012.
17. Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia. A New Cell Counter Based Attack Against Tor. In *CCS*, New York, New York, USA, 2009.
18. D. J. C. Mackay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, first edition edition, June 2003.
19. Y. Pyun, Y. Park, X. Wang, D. S. Reeves, and P. Ning. Tracing Traffic through Intermediate Hosts that Repacketize Flows. In *INFOCOM*, 2007.
20. S. Staniford-Chen and L. T. Heberlein. Holding Intruders Accountable on the Internet. In *IEEE S&P*, 1995.
21. J. H. van Lint. *Introduction to Coding Theory (third edition)*. Springer Verlag, Berlin (D), 1998.
22. X. Wang, S. Chen, and S. Jajodia. Tracking Anonymous Peer-to-peer VoIP Calls on the Internet. In *CCS*, 2005.
23. X. Wang, S. Chen, and S. Jajodia. Network Flow Watermarking Attack on Low-Latency Anonymous Communication Systems. In *IEEE S&P*, 2007.
24. X. Wang, D. Reeves, and S. F. Wu. Inter-Packet Delay Based Correlation for Tracing Encrypted Connections Through Stepping Stones. In *ESORICS*, 2002.
25. X. Wang and D. S. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Interpacket Delays. In *CCS*, 2003.
26. K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *ESORICS*, 2000.
27. W. Yu, X. Fu, S. Graham, D.Xuan, and W. Zhao. DSSS-Based Flow Marking Technique for Invisible Traceback. In *IEEE S&P*, 2007.
28. Y. Zhang and V. Paxson. Detecting Stepping Stones. In *USENIX Security Symposium*, 2000.