DESIGN, ANALYSIS, AND IMPLEMENTATION OF EFFECTIVE NETWORK FLOW WATERMARKING SCHEMES

BY

AMIR HOUMANSADR

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Doctoral Committee:

Assistant Professor Nikita Borisov, Chair
Professor Pierre Moulin
Professor David M. Nicol
Doctor Paul Syverson

ABSTRACT

Linking network flows is an important problem in several security-related applications including intrusion detection and anonymity. The flows of interest in these applications are commonly relayed by several low-latency nodes, and each node adds an extra/different layer of encryption to the traffic payload. As a result, in such applications it is infeasible to link network flows by correlating their packet headers and payloads. Traffic analysis is a powerful tool in this scenario, as it correlates network flows based on their communication patterns, which are not changed due to encryption and low-latency communications.

Traditional traffic analysis is performed in a passive manner; flow patterns are *only* observed and correlated among different flows to find flow relations. The patterns used more commonly for traffic analysis are packet timings, sizes, and counts. The main shortcoming of passive traffic analysis is not being *scalable* to large-size applications. For a network with n ingress flows and m egress flows passive traffic analysis needs to perform $O(mn)$ flow correlations, and $O(n)$ flows need to be communicated among traffic analysis entities. To overcome this limitation, recent research introduces an active form for traffic analysis, called *flow watermarking*. Flow watermarking works by first *perturbing* network flows and then *observing* network flows, looking for the specific perturbation patterns in order to link flows. This reduces the computational overhead to $O(n)$ (from $O(mn)$ for non-targeted passive analysis) and the communication overhead to $O(1)$ (from $O(n)$ in the case of non-targeted passive analysis), since the information used for flow linking, the *watermark*, is self-contained in each flow. In addition to being more scalable (in the case of non-targeted traffic analysis), active traffic analysis can provide lower false-positive rates than passive traffic analysis in linking network flows. This is because passive analysis works by correlating flow patterns that can be intrinsically correlated among non-linked flows, whereas

active traffic analysis correlates artificial patterns that are tailored to be highly uncorrelated among non-linked flows.

In this thesis we study the design, analysis, and implementation of network flow watermarks. Our research is motivated by two main issues with existing flow watermarks: i) previous flow watermarks are not efficient, meaning that they need very long network flows in order to successfully detect watermarked flows, and, ii) previous watermarks are not *invisible*, meaning that non-watermarking entities (e.g., users, attackers) can tell if a flow has been watermarked. In particular, we design an attack, called multi-flow attack, that works on several previously proposed flow watermarks. Our multi-flow attack is able to distinguish watermarked flows and remove the watermark by observing a handful of watermarked flows.

We also design two new flow watermarking systems. We design RAINBOW, a non-blind network flow watermark. RAINBOW is able to perform very reliable traffic analysis by applying tiny perturbations to packet timings. We also design a blind flow watermark, SWIRL, that inserts watermarks that are flow-dependent. For both of the proposed schemes, we analyze their detection performance both theoretically and through experiments. We also evaluate watermark invisibility for both of the schemes.

A flow watermark inserts a single bit of information on flows, carrying the message that these flows have been observed previously. However, in some applications more information needs to be piggybacked on network flows, giving information about the location they were observed, the entity who tagged them, etc. We call such tags composed of several bits of information flow fingerprints. We design a flow fingerprinting system, called Fancy, that is able to send tens of bits of information reliably using fairly short length of network flows.

Finally, we introduce two new applications for network flow watermarking. The first application uses flow watermarks to detect botmasters and infected machines corresponding to a centralized botnet, i.e., an IRC botnet. We also propose to use our SWIRL watermark to mitigate a threat against the Tor anonymity system, the Tor congestion attack.

To my grandmother, Fatima, who loved me with no bounds.
To my parents, for their love and support.
To my wife, Saloumeh, without whom I would never have made it this far.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my adviser, Professor Nikita Borisov, who has always supported me with his excellent guidance, insight, and patience. I also express my warmest gratitude to Professor Negar Kiyavash, who gave me brilliant advice and guidance on my thesis research. I also owe a great debt of gratitude to Professor Todd Coleman, for advising me on some theoretical aspects of my thesis. It gives me great pleasure in acknowledging the support and help of Professor Matthew Caesar with my academic career.

I cannot find words to express my gratitude to my committee members, Professor Pierre Moulin, Professor David M. Nicol, and Doctor Paul Syverson. This dissertation has greatly been improved after their invaluable comments and remarks; any remaining errors are wholly mine.

I am indebted to my many colleagues and friends at UIUC who supported me through all these years. Specially, I would like to express my thanks to the members of the Hatswitch group, past and present (in alphabetic order): David Albrecht, Anupam Das, Xun Gong, Sonia Jahid, Joshua Juen, Prateek Mittal, Shishir Nagaraja, Giang Nguyen, Nabil Schear, Robin Snader, and Qiyan Wang. I am also very thankful to my friends, Sara Bahramian, Farzan Farbiz, Yashar Heidari, Mina Kamouie, Hamed Okhravi, Navid Okhravi, Thomas Riedl, and Ehsan Shafiee, just to name a few.

Many thanks are due again to my family and my lovely wife, Saloumeh, who kept me sane throughout this long process; I could not have done it without them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

IPD	Inter-packet delay
PDF	Probability distribution function
CDF	Cumulative distribution function
FP	False-positive error rate
FN	False-negative error rate

CHAPTER 1

INTRODUCTION

Traffic analysis is the practice of inferring sensitive information from communication patterns instead of traffic content, and as more traffic is becoming encrypted traffic analysis is becoming more relevant to security. Traffic analysis has been particularly well studied in the context of intrusion detection to trace back attackers relaying their traffic through intermediate machines, i.e., stepping stones [1–3]. Traffic analysis has also been used in the realm of anonymous communication systems, where features of network flows are used to link two flows and break anonymity guarantees [4, 5].

Traditional traffic analysis schemes link network flows by *only* observing the network flows, trying to correlate related flows using features like packet timings, sizes, and counts [2, 3, 6–8]; we call theses schemes *passive traffic analysis*. More recently, an active approach called *watermarking* has been studied for traffic analysis. In this approach, traffic patterns of one flow (usually packet timings) are *actively* modified to contain a special pattern. If the same pattern is later found on another flow, the two are considered linked. Watermarking significantly reduces the computation and communication costs of traffic analysis, and also leads to more precise detection with fewer false positives. Watermarking has been considered for different applications of traffic analysis, i.e., detection of stepping stones [9–11] and compromising anonymous networks [12–14]. More recently, watermarking has also been studied as a traceback mechanism for botnets [15, 16].

Watermarking schemes are evaluated based on different features. *Detection accuracy* of a watermarking scheme expresses the probability of detecting a watermarked flow; this is always considered along with the *false positive error rate*, i.e., the probability of declaring a non-watermarked flow as watermarked. A watermark should be efficiently detectable after certain amounts of modifications have been applied on the watermarked flows, which is referred to as *watermark robustness*. In this thesis we only consider robustness

to channel perturbations, e.g., network delays, but not to active attackers[1] who try to destroy the watermark by introducing additional delays. In fact, in the watermark applications discussed in this thesis such an active attack is either impractical or inefficient. Another important feature of a flow watermarking system is the *watermark invisibility*; the watermark insertion should be imperceptible for the legitimate users by not interfering with their regular communication. Moreover, in some watermark applications an attacker should not be able to distinguish watermarked and non-watermarked flows through different statistical/analytical tools. For instance, if the watermark is used by a cybercriminal to de-anonymize network flow entering an anonymity system the cybercriminal needs to keep the watermark modifications invisible to the anonymity system to conceal her activity.

In this thesis, I investigate the design, analysis, and implementation of efficient network flow watermarks for different applications. In Section 1.1, I briefly mention how watermarking is used in different traffic analysis applications. Section 1.2 reviews the literature on traffic analysis, and in Section 1.3 I overview the thesis outline.

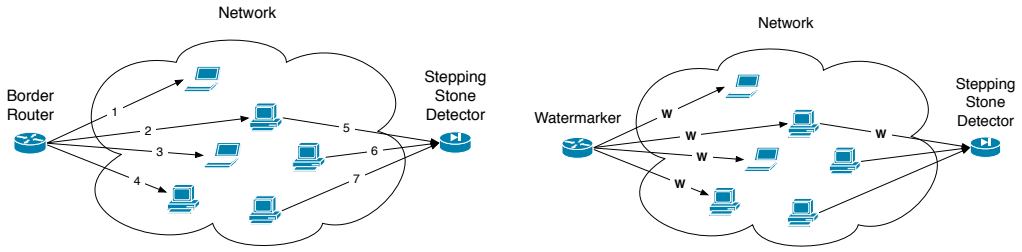
1.1 Applications of flow watermarking

1.1.1 Stepping stone detection

A stepping stone is a host that is used to relay traffic through a network to another remote destination. Stepping stones are used to disguise the true origin of an attack. Detecting stepping stones can help trace attacks back to their true source. Also, stepping stones are often indicative of a compromised machine. Thus detecting stepping stones is a useful part of enterprise security monitoring.

Generally, stepping stones are detected by noticing that an outgoing flow from an enterprise matches an incoming flow. For example, in Figure 1.1a, flow 2 will have the same characteristics as flow 5, allowing someone to link them by correlating such characteristics. Since the relayed connections are

[1] In different uses of network flow watermarking, the roles of watermarker and watermark attacker are assigned to different entities, e.g., security defenders, and security attackers. Throughout this thesis by attacker we refer to the watermark attacker, not the security attacker.



(a) A stepping stone (b) Network flow watermarks

Figure 1.1: Stepping Stone Detection

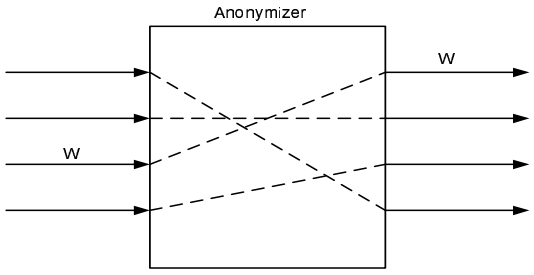


Figure 1.2: An anonymous system.

often encrypted (using SSH [17], for example), only characteristics such as packet sizes, counts, and timings are available for such detection. And even these are not perfectly replicated from an incoming flow to an outgoing flow, as they are changed by padding schemes, retransmissions, and jitter. As a result, statistical methods are used to detect correlations among the incoming and outgoing flows. Such detection can also be made in an active manner, i.e., using flow watermarks. As Figure 1.1b shows a watermarker can manipulate traffic characteristics of network flows entering the network, which are then looked for by some watermark detector.

1.1.2 Anonymous networks

At a very high level, an anonymous system maps a number of input flows to a number of output flows while hiding the relationship between them, as shown in Figure 1.2. The internal operation can be implemented in various manners, e.g., using onion routing [4], or a simple proxy [18]. The goal of an attacker, then, is to link an incoming flow to an outgoing flow (or vice versa).

A watermark can be used to defeat anonymity protection by marking cer-

tain input flows and watching for marks on the output flows. For example, a malicious website might insert a watermark on all flows from the site to the anonymizing system. A cooperating attacker who can eavesdrop on the link between a user and the anonymous system can then determine if the user is browsing the site or not. Similarly, a compromised entry router in Tor can watermark all of its flows, and cooperating exit routers or websites can detect this watermark.

Note that this does not enable a fundamentally new attack on low-latency anonymous systems: it has been long known [19] that if an attacker can observe a flow at two points, he can determine if the flow is the same, unless cover traffic is used. (In fact, deployed low-latency systems such as Onion Routing [4], Freedom [20], and Tor [21] have all opted to forego cover traffic due to it being expensive, hoping instead that it will be difficult for an attacker to observe a significant fraction of incoming and outgoing flows.) However, watermarking makes the attack much more efficient, as will be discussed in Section 1.2.2.

1.1.3 Other applications

Previous research on flow watermarking mostly focuses on the two problems of stepping stone detection and compromising anonymous networks, as introduced above. In fact, we can think of using flow watermarks in other networking scenarios that exhibit *low-latency* communication mechanisms, as high-latency communication renders flow watermarking inefficient by destroying the tiny watermark patterns. In Chapter 6 we introduce two new applications for network flow watermarks, which are botnet detection and mitigating an attack on the Tor network.

1.2 Literature review

In this section, we review the literature on passive traffic analysis and watermark-based traffic analysis schemes.

1.2.1 Passive traffic analysis

In general, passive traffic analysis techniques operate by recording characteristics of incoming streams and then correlating them with outgoing ones. The right place to do this is often at the border router of an enterprise in the case of stepping stone detection, 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 enterprise with many interconnected networks, a connection relayed through a stepping stone may enter and leave the enterprise through different points; in such cases, there is also some communication overhead for transmitting traffic statistics between border routers. This communication overhead is also essential to the entities collaborating to compromise anonymity of an anonymous network.

The passive schemes have explored using various characteristics for correlating streams. Staniford-Chen and Heberlein were the first to target detection of stepping stones through statistical analysis of character frequencies [1]. This was followed by other works, all using packet contents to match connections composing a stepping stone. Looking at the average lag between two connections, Yoda et al. devise a quadratic time deviation-based scheme trying to discriminate stepping stone connections from ordinary connections [2]. Zhang and Paxson model interactive flows as ON/OFF processes and detect linked flows by matching up their ON/OFF behavior [3]. Wang et al. focus on inter-packet delays, and consider several different metrics for correlation [6]. More recently, He and Tong used packet counts for stepping stone detection [22].

Passive traffic analysis has also been considered for anonymous communications. Syverson et al. analyse the security of onion routing [4] in [19]. Back et al. describe generic attacks that apply to Freedom [20] and PipeNet [23] anonymous networks, e.g., counting packets, and latency attack [24]. In [25] Raymond presents the traffic analysis problem with an emphasis on mix-based systems [26]. Danezis also considers the traffic analysis of continuous-time mixes in [27].

Donoho et al. were the first to consider intruder evasion techniques [7]. They defined a *maximum-tolerable-delay* (MTD) model of attacker evasion and suggested wavelet methods to detect stepping stones while being robust

to adversarial action. Blum et al. used a Poisson model of flows to create a technique with provable upper bounds on false positive error rates [8], given the MTD model. However, for realistic settings, their techniques require thousands of packets to be observed to achieve reasonable rates of false errors.

1.2.2 Network flow watermarks

To address some of the efficiency concerns of passive traffic analysis, Wang et al. proposed the use of watermarks [9]. In this scenario, a border router will modify the inter-packet delays (IPD) of the incoming flows to contain a particular pattern—the watermark. If the same pattern is present in an outgoing flow, a stepping stone is detected. This can be seen in Figure 1.1b.

Watermarks improve upon passive traffic analysis in two ways. First, by inserting a pattern that is uncorrelated with any other flow, 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^{-5} or lower, as compared to 10^{-2} with passive watermarks). Second, they can operate in a *blind* fashion; with passive traffic analysis, if one monitor observes n input flows and another observes m output flows, most current techniques [7–9] for passive traffic analysis use $O(mn)$ computation by comparing each ingress flow with each egress flow, and the best known techniques [28] improve upon that to $O(n + \sqrt{mn})$, whereas blind watermarking only needs O(n) computation. The passive schemes also require $O(n)$ communication. With blind watermarking, on the other hand, no communication needs to take place between the two monitors after they have established a shared secret key, and the computation cost is $O(n)$ and $O(m)$ at the watermarker and detector, respectively, as the watermarker marks each input flow and the detector checks each output flow for the presence of a mark. 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. Note that this is only true for "non-targeted" traffic analysis: for "targeted" traffic analysis, where an attacker tries to find the egress flow linked to a single specific ingress flow, the correlation and communication requirements of passive traffic analysis are $O(n)$ and $O(1)$, respectively, which are similar to that of blind watermarking.

The IPD-based watermark of [9] was later shown to be detectable [29], in addition to being susceptible to packet modification, e.g., packet additions/removals. To be robust against repacketization, Pyun et al. proposed an *interval-based* watermarking scheme in [10] by delaying packets of some intervals based on the watermark value. Dealing with intervals instead of packets themselves makes the detection scheme robust to packet addition/removal. Some of the schemes target anonymous communication [13,14] rather than stepping stones as the application area, but the techniques for both are comparable. In [13] Wang et al. use an interval-based watermarking scheme similar to [10] aiming to compromise anonymity of Anonymizer [18]. Yu et al. suggested another interval-based watermark [14] by changing the packet rates of flow intervals using direct sequence spread spectrum (DSSS). Recently, Jia et al. have shown how this watermark is detectable exploiting statistical attacks against DSSS [30]. Another watermark focused on anonymous networks was proposed by Wang et al. which tracks anonymous peer-to peer VOIP calls over Internet [12].

The mentioned watermarking schemes insert large watermark delays in order to provide an efficient detection. This results in watermark visibility and is used to design attacks against them. Motivated by this, we intend to design efficient watermarking schemes that are robust, invisible, and efficiently detectable.

1.3 Thesis outline

The rest of this thesis is as follows: in Chapter 2 we design an attack that targets several interval-based flow watermarks by observing a handful of watermarked flows. This attack, multi-flow attack (MFA), is shown to be effective against previous interval-based watermarks of [10, 13, 14]. We also suggest a number of countermeasures to make the existing interval-based schemes robust to our MFA attack. In Chapter 3 we design the first non-blind watermark for network flows, called RAINBOW. RAINBOW works by inserting tiny spread spectrum watermarks on packet timings, in a manner that is undetectable by third-parties using statistical tools. We find the optimum detectors for RAINBOW under different traffic models using the theory of hypothesis testing, and compare its detection performance with a

similar passive traffic analysis scheme. We also validate the analysis using an implementation running on PlanetLab. Even though RAINBOW performs a robust, yet invisible, watermarking it has scalability limitations similar to passive traffic analysis, as a side effect of being non-blind. In Chapter 4, we design SWIRL, a blind watermark that performs robust and invisible watermarking. SWIRL uses an interval-based approach to provide robustness to packet modifications, yet using a flow-dependent structure it also provides robustness to the MFA attack despite being an interval-based watermark. We assess SWIRL's performance using a theoretical analysis, as well as by performing simulations and prototype implementation. Also, we evaluate SWIRL's invisibility to different statistical metrics. In Chapter 5, we propose a flow fingerprinting scheme, called Fancy. Fancy uses a main structure similar to that of RAINBOW, but uses coding algorithms to be able to send several information bits reliably by perturbing flow patterns. This is contrast to flow watermarks that intend to send a single bit of information. We investigate the use of several popular coding algorithms in the design of Fancy. In Chapter 6, we investigate the use of flow watermarking in low-latency applications other than stepping stone detection and anonymity systems. In particular, we design a botnet-detection system that uses watermarking to trace back botmasters, and to detect machines infected by IRC-based botnets. As another novel application, we propose the use of our SWIRL watermark to mitigate a threat against the Tor anonymity network, called the Tor congestion attack. We conclude this thesis in Chapter 7 along with some insights into future research directions.

CHAPTER 2

MULTI-FLOW ATTACK ON NETWORK WATERMARKS

In Section 1.2, we mentioned several attacks that aim to break through flow watermarking schemes. In this chapter, we introduce a new attack that compromises a large group of flow watermarks by obtaining a handful of watermarked flows.

Early flow watermarks have been applied to two problems of attacking anonymity systems [12–14] and detecting stepping stones [9, 10]. In both contexts, many flows must be watermarked in order to learn new information. In our work, we consider whether an attacker[1] can learn enough information to defeat the watermark by observing multiple watermarked flows [31]. We apply this multi-flow threat model to the latest generation of *interval-based watermarks* [10, 13, 14]. These watermarks subdivide the flow to be marked into discrete time intervals and perform transformative operations on an entire interval of packets. This approach is more robust to packet losses, insertions, and repacketizations, than previous approaches that focused on individual packets [9, 12], because the time intervals allow the watermarker and detector to retain synchronization. However, the same synchronization property can be exploited by attackers by "lining up" multiple watermarked flows and observing the transformations that were inserted.

We show through experiments that the interval-based watermark schemes are completely vulnerable to an attacker who can collect a small number of watermarked flows—about 10. This is sufficient to not only detect that a watermark is indeed present, but also to recover the secret parameters of the watermark scheme and to be able to remove the watermark at a low cost. Furthermore, our attack works even if different watermarked flows

The research presented in this chapter is done in a collaboration with Prof. Negar Kiyavash, from UIUC.

[1] We use "attacker" here to refer to someone attacking the watermarking scheme; in the case where watermarks themselves are used by attackers, these will be the "counter-attackers."

contain different embedded "messages" with only about twice the number of watermarked flows necessary. We also analytically estimate the false-positive rates for our attack and find them to be very low.

We also consider some countermeasures to such attacks. We show that by using multiple "keys" (time interval assignments) to watermark different flows, it is possible to defeat our attack. This countermeasure comes at a cost of higher computation overhead at the detector and a higher rate of false positives. However, this increased cost is only linear, whereas the increased cost of the attacker is superexponential, thus providing an effective defense.

The rest of the chapter is organized as follows. We next introduce three recent interval-based flow watermarks that are the targets of the MFA attack presented in this chapter. Section 2.2 describes the theoretical foundation for our attack, and Section 2.3 implements the attack. We discuss potential countermeasures to the attack in Section 2.4. Section 2.5 concludes the chapter.

2.1 Interval-based watermarks

We next introduce three recent interval-based network flow watermarks.

2.1.1 Interval Centroid-based Watermarking (ICBW)

We next review the scheme proposed by Wang et al. [13]; for more details of the scheme as well as some analysis we refer the reader to [13]. The scheme is based on dividing the stream into intervals of equal lengths, using two parameters: o, the offset of the first interval, and T, the length of each interval. A subset of $2n$ of these intervals is randomly selected which is subsequently randomly divided into two further subsets A and B each consisting of $n = rl$ intervals. Each of the sets A and B are randomly divided to l subsets denoted by $\{A_i\}_{i=1}^{l}$ and $\{B_i\}_{i=1}^{l}$ each consisting of r intervals. The i-th watermark bit is encoded using the sets $\{A_i, B_i\}$. Therefore, a watermark of length l can be embedded in the flow. Figure 2.1 depicts the random selection and grouping of time intervals of packet flow for watermark insertion.

The watermarker and detector agree on the parameters o, T and use a pseudorandom number generator (PRNG) and a seed s to randomly select

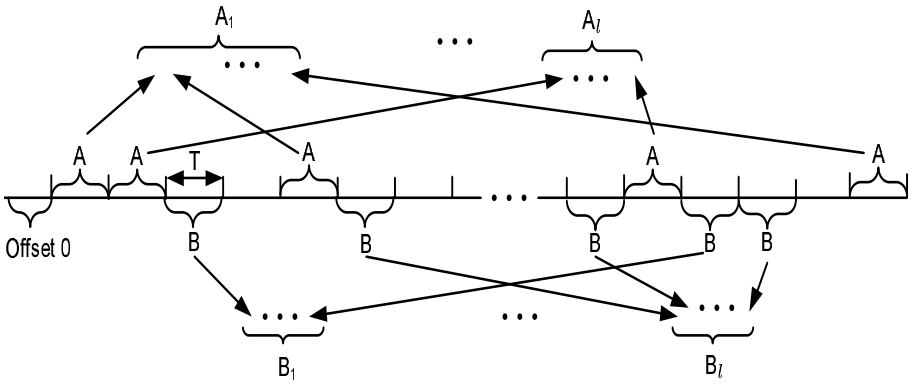


Figure 2.1: Random selection and assignment of time intervals of packet flow for watermark insertion.

and assign intervals for watermark insertion. To keep the watermark transparent, all of these parameters are kept secret. Depending on whether the i-th watermark bit is 1 or 0, the watermarker delays the arrival times of the packets at the interval positions in sets A_i or B_i respectively, by a maximum of a. Figure 2.2 illustrates the effect of this delaying strategy over the distribution of packets arrival time in an interval of size T (this operation is called "squeezing" by Wang et al.) Finally, the overall watermark embedding is illustrated in Figures 2.3 (a) and (b).

As the result of this embedding scheme, the expected value of aggregate centroid, i.e., the average of the arrival time of the packets modulo the length of the interval T, in either the intervals A_i (when watermark bit is 1) or B_i (when watermark bit is 0) corresponding to bit i is increased by $\frac{a}{2}$. The expected difference between the aggregate centroid of A_i and B_i now will be $\frac{a}{2}$ when watermark bit is 1 or $-\frac{a}{2}$ when watermark bit is 0.

The detector checks for the existence of the watermark bits. The check on watermark bit i is performed by testing whether the difference of the aggregate centroid of packet arrival times in the intervals A_i and B_i is closer to $\frac{a}{2}$ or $-\frac{a}{2}$. If it is closer to $\frac{a}{2}$, then the watermark bit is decoded as 1 and if it is closer to $-\frac{a}{2}$, the bit is declared a 0. By focusing on the arrival times of many intervals (r of them for each bit of watermark) rather than individual packet timings, ICBW approach is robust to repacketization, insertion of chaff, and mixing of data flows. Network jitter can shift packets from one interval into another, but the suggested parameters for a and T (350ms and 500ms respectively) are large enough that few packets will be affected.

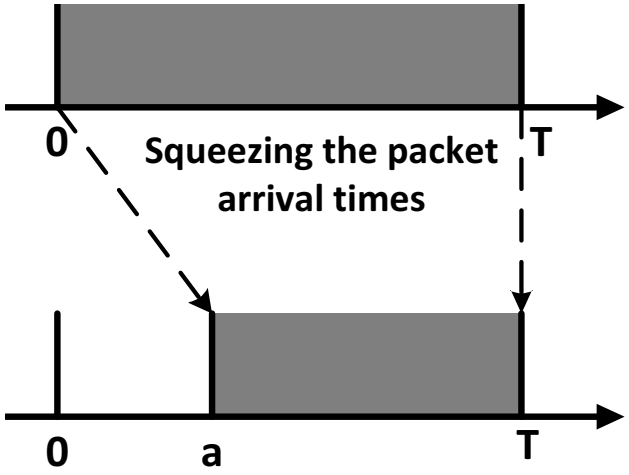
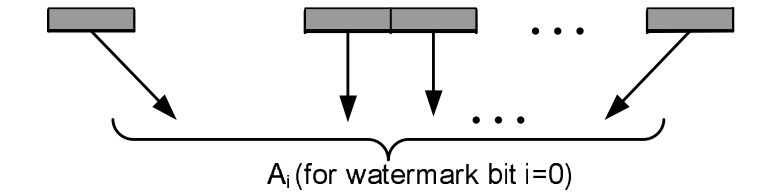


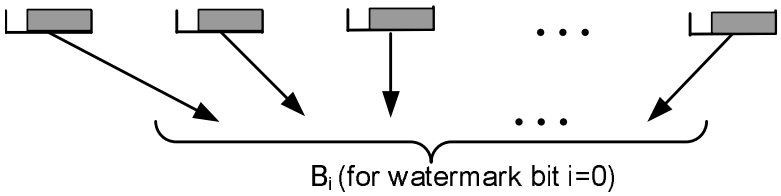
Figure 2.2: Distribution of packets arrival time in an interval of size T before and after being delayed.

The secrecy of the interval positions A_i and B_i make the mark difficult to detect or remove, as it is hard to distinguish the patterns generated by the mark from natural variation in traffic rates. We show in Sections 2.2 and 2.3, however, that a simple technique allows an observer to effectively recover the watermark positions and values. This technique is applicable to any watermarking scheme that creates periods of clear or low traffic at *specific* parts of the flows across many flows. Next, we briefly describe *Interval-Based Watermarking (IBW)*, a flow watermarking scheme proposed by Pyun et al. [10] to detect *stepping stones*. Our attacks also applies to this scheme.

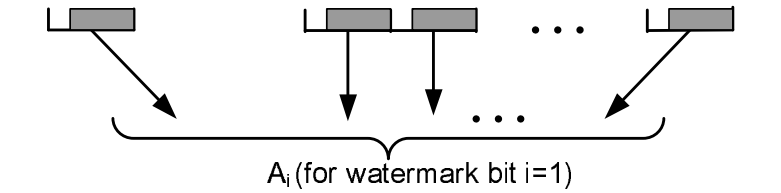
2.1.2 Interval-Based Watermarking

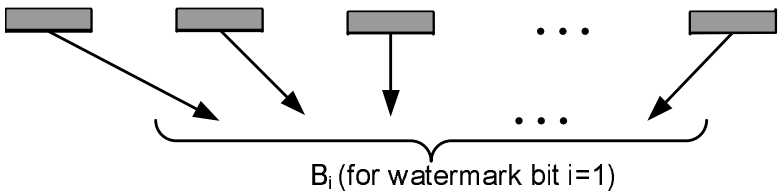
Similar to ICBW, the watermarking scheme of Pyun et al. [10] manipulates the arrival times of the packets over a set of preselected intervals. The watermark embedding is achieved by manipulating the rates of traffic in successive intervals. There are two manipulations: an interval I_i may be *cleared* by delaying all packets from interval I_i until interval I_{i+1}, or it may be *loaded* by delaying all packets from interval I_{i-1} until interval I_i. A loaded interval will therefore have twice the expected number of packets, and a cleared one will have none. To send a 0 bit in position i, the interval I_i is cleared and I_{i+1} is loaded; to send a 1, I_i is loaded and I_{i+1} is cleared. (Note that since clearing





(a) Insertion of watermark bit 0





(b) Insertion of watermark bit 1

Figure 2.3: ICBW bit insertion.

one interval implicitly loads the next, it takes 3 intervals to send a bit.)

The watermarker and detector agree on the parameters o, T and a list of positions $S = \{s_1, \ldots, s_n\}$; all of these parameters are secret. The watermarker encodes the watermark bits at the interval positions s_i and the detector checks for the existence of the watermark. The check is performed by testing whether the data rate in interval I_{s_i} differs from the rate in interval I_{s_i+1} by a factor exceeding a threshold; if it does, then a 0 or 1 bit is considered detected. By focusing on data rates rather than individual packet timings, the interval-based approach is robust to repacketization of data flows.

The detection process may generate false positives due to natural variation in packet rates, or false negatives, as delays between the watermarker and repacketization at the relay cause rates in intervals to shift. To ensure reliable transmission, each watermark bit is encoded in several positions in the stream. Pyun et al. show that this technique operates with very low false-positive and false-negative rates.

2.1.3 Spread-Spectrum Watermarking

In DSSS watermarking technique of Yu et al. [14], each bit of a length-n binary watermark is embedded in an interval of length T_s. Hence the whole watermark is inserted in some part of the flow of length nT_s. To embed a watermark bit 1, the rate of the packets in its length-T_s designated interval are manipulated according to a pseudo-noise (PN) code. The PN code is a fast varying signal that switched between $+1$ and -1; the duration of each ± 1 period is T_c. In particular, Yu et al. choose a length-7 PN code for their implementation. When PN code is $+1$, the rate of flow remains intact, but when PN code is -1, the rate of flow is decreased for a duration of T_c. The flow rate is manipulated by creating an interfering flow and relying on TCP congestion control. (Note that this approach works only with bulk flows where the sending rate is indeed limited by TCP congestion control.) On the other hand to embed a watermark bit 0, the flow is manipulated using the complement of the PN code. Figure 2.4 depicts the embedding of watermark 110 for a PN code of length 5.

The watermarker and detector agree on the parameter T_s, the watermark,

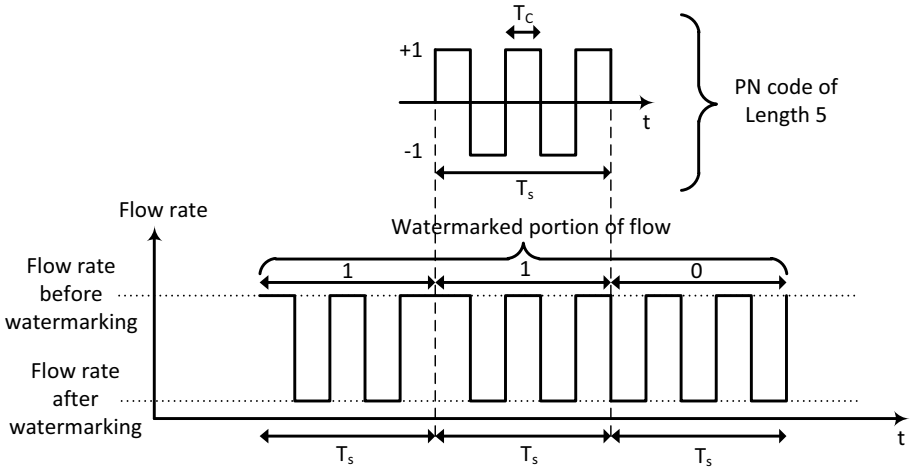


Figure 2.4: A length-5 PN code and insertion of DSSS watermark 110.

and a pseudo-noise code. The detector recovers the watermark by first applying a high-pass filter to the received signal and subsequently passing it through despreading and a low-pass filter. The details of the detector's structure are inconsequential to our attack and the interested reader is referred to [14].

Given that the watermark insertion technique in DSSS reduces the flow rates over certain intervals across all flows, it is vulnerable to our averaging attack, which is analysed in this paper. More recently, Huang et al. suggest to change the DSSS watermark to use different PN codes for watermarking different flows in order to defend against the multi-flow attack presented in this paper [32]. This approach results in increasing the false-positive rates of the watermark detection as well as the complexity of the watermark detector, since a detector needs to correlate any received flow against all possible PN codes that might have been used for watermarking; unfortunately, this has not been considered by the authors.

2.2 Attack analysis

In this section, we present a probabilistic analysis of our attack using a model for interactive traffic. Though some watermarked traffic may consist of non-interactive bulk transfer traffic, we will show in Section 2.3.1 that interactive

traffic presents a more difficult case for our attack, and thus we analyze it here. As DSSS watermarks work well only against non-interactive traffic, our analysis here applies only to IBW and ICBW, but as we demonstrate experimentally, our attack will work on DSSS watermarks as well.

2.2.1 Probabilistic model of interactive traffic

We first present a model for interactive traffic, as it is essential to our analysis. Let f_m denote the m-th flow in a pool of interactive traffic flows. Given that the traffic might be encrypted, we do not consider the content of the packets; likewise, the sizes of packets representing keystrokes are likely to be uniform. We thus consider only the arrival time of the packets in the flow, allowing us to model the flow as a point process.

Suppose we observed packet arrivals at times $t_1 < t_2 < \cdots < t_n$ in a fixed interval $(0, \tau]$ such that t_i is the time the i-th packet arrived. The collection of arrival times $\mathbf{t}_m = (t_1, t_2, \ldots, t_n)$ specifies a flow f_i. Furthermore, we model the interactive connection as a Markov-modulated Poisson process (MMPP) [33, 34]. The set of possible states are $\{0, 1\}$, where state 0 corresponds to user typing characters and state 1 corresponds to periods of silence. Figure 2.5 depicts this two-state MMPP.

Let $X(t)$ denote the state of the process at time t. When the process is at state 0, packet arrivals are modeled as a renewal process; i.e., the interarrival times are independent and identically distributed (i.i.d.). In case of interactive traffic flow this renewal process is often modeled as Poisson [7, 8]. The Poisson assumption means that the interarrival time of the packets, denoted by θ, are exponentially distributed. Hence its probability density function (PDF) is given by:

$$f_\theta(t) = \lambda e^{-\lambda_0 t}$$

where λ_0 denotes the rate of the Poisson process. When the process is in state 1, the arrivals are again modeled as Poisson but with rate $\lambda_1 < \lambda_0$. Given that state 1 corresponds to a period of silence (no packet arrivals), as soon as a packet arrives the embedded Markov chain transitions to state 0. Therefore, the transition probabilities $\{P_{ij}, i \geq 0, j \geq 0\}$ of the embedded

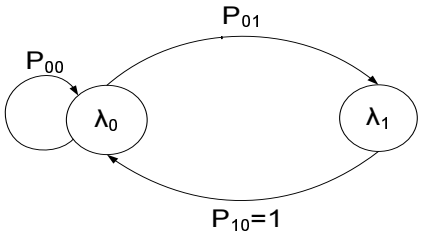


Figure 2.5: The embedded two-state Markov chain.

Markov chain $\{X_n, n \geq 0\}$ are as follows:

$$\begin{aligned} P_{00} + P_{01} &= 1, \\ P_{01} = 1, P_{11} &= 0 \end{aligned} \tag{2.1}$$

and the embedded Markov chain is defined by the matrix:

$$\begin{bmatrix} P_{00} & 1 \\ 1 - P_{00} & 0 \end{bmatrix}$$

The steady state probabilities π_0, π_1 of the embedded chain X_n are given by:

$$\begin{bmatrix} \pi_0 \\ \pi_1 \end{bmatrix} = \begin{bmatrix} P_{00} & 1 \\ 1 - P_{00} & 0 \end{bmatrix} \begin{bmatrix} \pi_0 \\ \pi_1 \end{bmatrix}$$

or:

$$\pi_0 = \frac{1}{2 - P_{00}}, \qquad \pi_1 = \frac{1 - P_{00}}{2 - P_{00}}$$

The steady state probabilities P_0, P_1 of the Markov process $X(t)$ are given by ([34]):

$$P_i = \frac{\frac{\pi_i}{\lambda_i}}{\sum_k \frac{\pi_k}{\lambda_k}}$$

or:

$$P_0 = \frac{\lambda_1}{\lambda_1 + (1 - P_{00})\lambda_0}, \qquad P_1 = \frac{(1 - P_{00})\lambda_0}{\lambda_1 + (1 - P_{00})\lambda_0} \tag{2.2}$$

The significance of the steady state probabilities of (2.2) is that they capture the probability of each of the states 0 and 1 at any given point in time. Recall that ICBW encodes the watermark bits "1" or "0" by delaying the arrival times of the packets at the set of intervals A_i or B_i respectively and IBW encodes the watermark bits "0" or "1" by transferring the traffic of an

interval of length T to some adjacent interval. Therefore, they both create periods of time with no arrivals in the flow. This period for ICBW is of length a and for IBW is of length T. When the embedded Markov chain is in state i, we can compute the probability of zero occurring in a period of length ℓ starting at any given point as:

$$P_{f_m^i}(0;\ell) = e^{-\lambda_i \ell} \tag{2.3}$$

since the waiting times are exponentially distributed and therefore memoryless.

In general given a flow f_m generated from an MMPP, from (2.3) probability of having a period of length ℓ with no arrivals $P_{f_m}(0;\ell)$ is:

$$\begin{aligned} P_{f_m}(0;\ell) &= P_0 P_{f_m^0}(0;\ell) + P_1 P_{f_m^1}(0;\ell) \\ &= P_0 e^{-\lambda_0 \ell} + P_1 e^{-\lambda_1 \ell} \end{aligned} \tag{2.4}$$

where the steady state probabilities $\{P_0, P_1\}$ are given by (2.2).

A good watermarking scheme requires that the watermarked stream should not reveal any clues of the presence of the watermark to unauthorized observer. Therefore, it is desirable that $P_{f_m}(0;\ell)$ above should be reasonably large so that presence of silent periods does not give away the watermark. We next present parameters of our two-state MMPP and show that for those parameters the watermark indeed cannot be detected with observing a single stream watermarked with ICBW or IBW. However, we will show that if the attackers have access to multiple copies of a marked signal, they can defeat the two watermarking schemes both when multiple flows are watermarked with the same key and when they are watermarked using various keys.

2.2.2 Parameter selection and goodness of fit

We estimated the parameters P_{00}, λ_0, and λ_1 of our MMPP model by using network traces of SSH connections taken at a wireless access point in our institution. For a trace, we first estimated the underlying state of the embedded Markov chain by choice of a threshold η. If the interarrival time between two packets exceeded the threshold η, we assumed that the process was in state 1 and if the interarrival time between two packets was less than

the threshold η, we assumed that the user was typing and therefore she/he was in state 0. Once the states $\{X_n, n \geq 0\}$ of the underlying chain are determined, by concatenation of the parts of the interactive traffic that came from same underlying state, we could extract two Poisson flows with rates λ_0 and λ_1 from the original flow.

Given that the expected number of arrivals of a Poisson process distribution with parameter λ in time interval $(0, t]$ is λt, we estimated the rate λ_0 and λ_1 by calculating the arrival rates of each of the two extracted flows. Parameter P_{00} was estimated as the portion of the time the chain spent at state 0. Our estimated values for the transition probability P_{00} and the rates λ_0 and λ_1 were as follows:

$$P_{00} = .96 \qquad \lambda_0 = 5.6 \qquad \lambda_1 = 0.57. \tag{2.5}$$

To assess the goodness of fit of our MMPP with parameters of (2.5), we used a quantile-quantile (q-q) plot [35]. Using the theoretical CDF of the model, the observations are mapped into values in interval $[0, 1]$. If the underlying statistical model of the data is consistent with the observations, the values obtained from the mapping are uniformly distributed in the interval $[0, 1]$. To assess the uniformity of the mapped values or equivalently assessing the goodness of the theoretical model an empirical CDF of the mapped values is compared against the theoretical CDF of a uniform distribution which is a 45-degree reference line. The closer the CDF to this reference line, the greater the evidence that the statistical model captures the underlying phenomenon. The q-q plot of Figure 2.6 for our model shows that the MMPP model for the interactive traffic with parameters (2.5) provides a good fit for the data and significantly outperforms a simpler Poisson model, or a Pareto distribution that has been previously proposed to fit interactive traffic [36].

2.2.3 Multi-flow attack

Regardless of whether the ICBW or IBW watermarking schemes are implemented using the same message across all interactive flows or they use multiple message for different flows, they are subject to an averaging attack. This is because both schemes embed watermarks by emptying the same parts across various flows. Next, we will explain our attack for both the single-

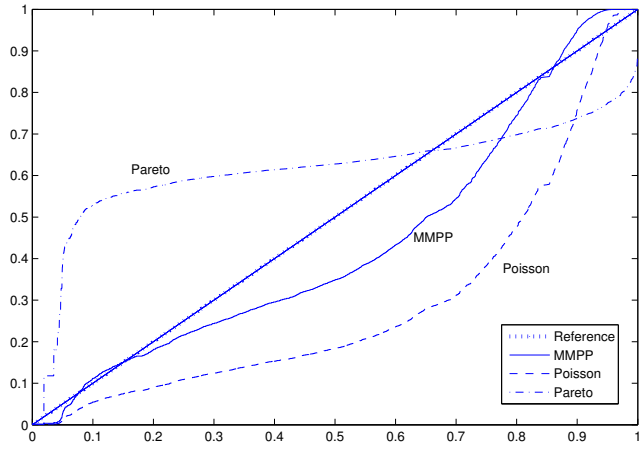


Figure 2.6: Q-Q plot of Poisson and MMPP models with our sample data.

message and multiple-message watermarks.

Averaging Attack against Single-Message Watermarks

When ICBW or IBW watermarking schemes are implemented using the same message across all interactive flows, if the attacker has access to k watermarked flows, he can form an aggregate of all the flows by taking the sorted union of all the arrival times of packets in all flows. We denote this aggregated stream by $\overline{f_k}$, where the subscript k denotes the total number of streams involved in forming the aggregate flow.

Given that each interactive stream is independent of all the other streams, the probability of having a period of length T with no arrivals in the flow $\overline{f_k}$ is given by:

$$P\{N_{\overline{f_k}}(t_a + \ell) - N_{\overline{f_k}}(t_a) = 0\} \quad = \quad \prod_{i=1}^{k} P_{f_i}(0;\ell) = P_{f_m}(0;\ell)^k \qquad (2.6)$$

Equation (2.6) shows that probability of having period of length ℓ with no arrivals decreases exponentially in k, the total number of copies used to form the aggregate flow $\overline{f_k}$. Therefore, if the streams are not watermarked there is a very small probability that the aggregate stream has periods of no arrivals. However, if ICBW or IBW use the same key and message across all interactive

flows, the aggregated copy of the watermarked flows always exhibits patterns of no arrivals of length ℓ that give away the location of the watermark as well as the maximum delay parameter a of ICBW and the period T of IBW.

Substituting the parameters of (2.5) into (2.4), assuming $\ell = 350\,\text{ms}$, as suggested by Wang et al. [13], we have $P_{f_m}(0; 0.35) = 0.33$. Therefore, in an aggregate of as few as 10 flows probability of a periods of 350 ms without any arrivals is as low as $P_{f_m}(0; 0.35)^{10} = 1.6 \times 10^{-5}$. Similarly for $\ell = 900\,\text{ms}$, as used by Pyun et al. [10], we have $P_{f_m}(0; 0.9) = 0.17$ and $P_{f_m}(0; 0.9)^{10} = 2.4 \times 10^{-8}$.

This, of course, shows us the probability of finding an empty interval in a particular spot; we next consider the possibility of finding empty intervals at *any* position in the flows. To do so, we use a discrete approximation. Given an aggregate flow of length L, we are interested in finding the probability of having an empty interval of length ℓ at any position. For this, we divide the aggregate flow into non-overlapping intervals with length $\ell_M = \ell/M$ (a total of $N = \lfloor L/\ell_M \rfloor$ intervals). Finding M (or $M-1$) consecutive empty intervals of length ℓ_M gives lowerbound (upperbound) of this probability.

Since $P_{00} = 0.96$, the process is nearly memoryless and we can approximate the discrete version of the problem as a Bernoulli process, where each interval is empty with probability $p_M = P_{\overline{f_k}}(0; L_M)$. For a total of n intervals let us refer to the probability of finding s consecutive empty intervals as $P_E(s, p_M, n)$. We can compute this using a recurrence.[2] Let $y[n] = P_E(s, p_M, n)^c$, i.e., the probability of finding *no* consecutive runs of s empty intervals among the first n intervals. Then, for $n \geq s$, we have:

$$y[n] = y[n-1] - (1 - p_M)p_M^s \cdot y[n-s-1]$$

This is because the probability of having no runs of s empty intervals among n is the probability that there aren't any empty intervals among the first $n-1$, less the probability that there is exactly one run among the last s intervals. This recurrence has the characteristic polynomial:

$$p(x) = x^{s+1} - x^s + (1 - p_M)p_M^s$$

Any solution to the recurrence can be expressed in terms of the roots of the polynomial $p(x)$; given roots r_i with respective multiplicities m_i, we have

[2]This solution is adapted from [37].

that:

$$y[n] = \sum_i p_i(n) r_i^n$$

where p_i is a polynomial of degree at most $m_i - 1$. (See, for example, [38, Theorem 4.5.6].) Note that $y[n] = 1$ for $n < s$, which allows us to solve for the coefficients of the polynomials. Finally, we compute $P_E(s, p_M, n) = 1 - y[n]$.

Note that the schemes above will create *multiple* blank intervals, so we compute the probability of finding e blank intervals of length ℓ in a flow of length L, $P'_E(L, \ell, e)$. Modeling the process as approximately memoryless, we can see that, for $e > 1$ and $L > \ell$:

$$P'_E(L, \ell, e) = P'_E(L, \ell, 1) P'_E(L - \ell, \ell, e - 1)$$

Therefore:

$$P'_E(L, \ell, e) = \begin{cases} \prod_{i=0}^{e-1} P'_E(L - i\ell, \ell, 1) & L \geq e\ell \\ 0 & \text{otherwise} \end{cases}$$

where $P'_E(L - i\ell, \ell, 1)$ is approximated by the method above, i.e.,

$$P_E(M-1, p_M, \lfloor (L - i\ell)/\ell_M \rfloor) \leq P'_E(L - i\ell, \ell, 1) \leq P_E(M, p_M, \lfloor (L - i\ell)/\ell_M \rfloor)$$

We apply these computations to parameters, taken from the evaluation of the ICBW scheme by Wang et al. [13]. They used a 32-bit watermark, with a redundancy between 12 and 20, and flow lengths between 394 and 650 seconds. In Figure 2.7, we plot $P_{E'}(394\,\text{s}, 350\,\text{ms}, 12 \times 32)$ as a function of the number of aggregated flows. We can see that, even for small numbers of flows, the false-positive probability of our attack is quite low. This graph was computed using $M = 40$, which is sufficient to give an approximate error of less than 10^{-45} for $k > 4$, computed by comparing the upper and lower bounds.

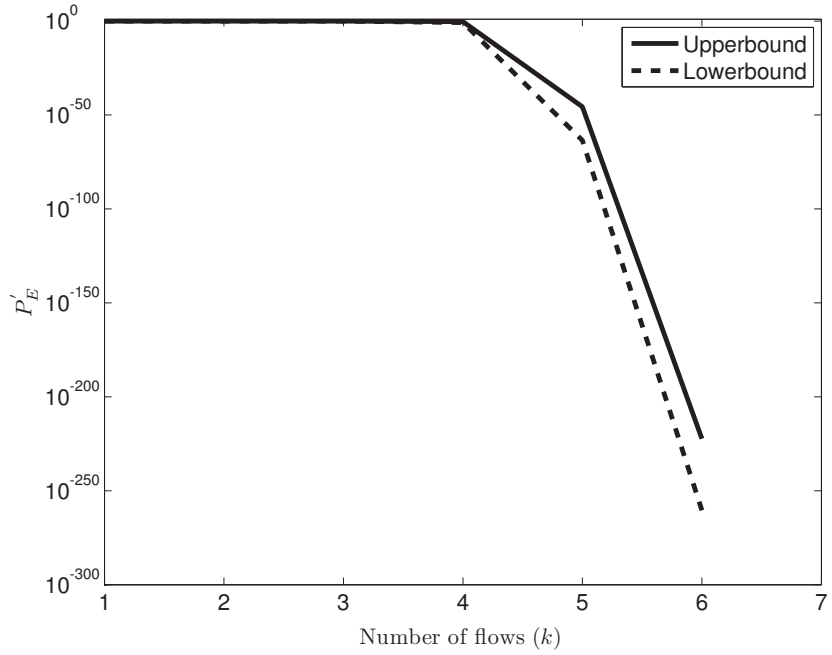


Figure 2.7: False positive errors of MFA for different number of aggregated flows.

2.2.4 Impact of timing perturbations

Our analysis above assumed that the attacker sees the timings of the watermarked stream directly. In reality, these timings will be perturbed by network delays. As a result, the intervals cleared by the watermark may have some packets from previous intervals shifted into them and no longer appear completely empty. Note that what is relevant here is not the magnitude of the network delay but its variance, or *jitter*, since delaying all packets by an equal amount does not affect our attack. And if the jitter is much less than ℓ, our attack will work equally well: if jitter is $< \epsilon$ with high probability, then we will find clear intervals of length at least $\ell - \epsilon$ in the k averaged watermarked streams, whereas the probability of seeing such an interval in unwatermarked streams is $P_{f_m}(0; \ell - \epsilon)^k \approx P_{f_m}(0; \ell)^k$, which is vanishingly small. We observe that the studied parameters of the ICBW and IBW schemes have $\ell = 350\,\text{ms}$ or $900\,\text{ms}$, in order to resist traffic perturbations, repacketization, etc. The network jitter, on the other hand, is two orders of magnitude smaller. Our experiments on PlanetLab [39] show it to be on the order of several milliseconds for geographically distributed hosts, and this matches the results of previous studies [40]. Therefore, it is indeed

the case that the jitter is $< \epsilon \ll \ell$, and so it will not significantly affect our attack.

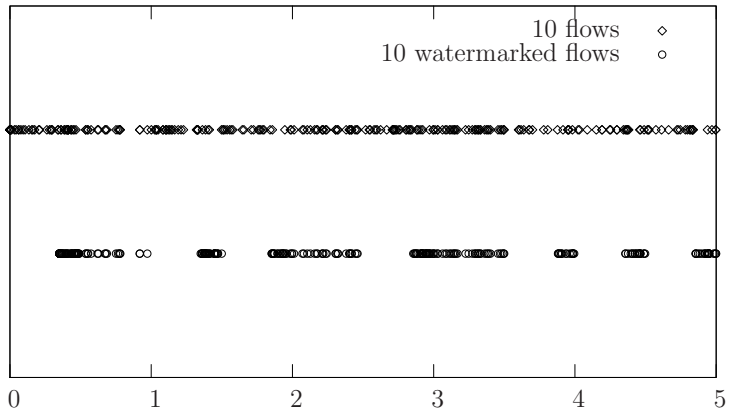
2.3 Implementation

Having shown the theoretical background behind our attack, we now show the result of implementing it in practice. We developed algorithms to detect the presence of a watermark, recover the secret parameters, and to remove the watermark from new streams. We evaluated the algorithms using both real flows gathered from traces and synthetic flows generated using our MMPP model, presented in Section 2.2.1. We first present our attacks for same-value watermarks, and then extend it to multi-valued watermarks.

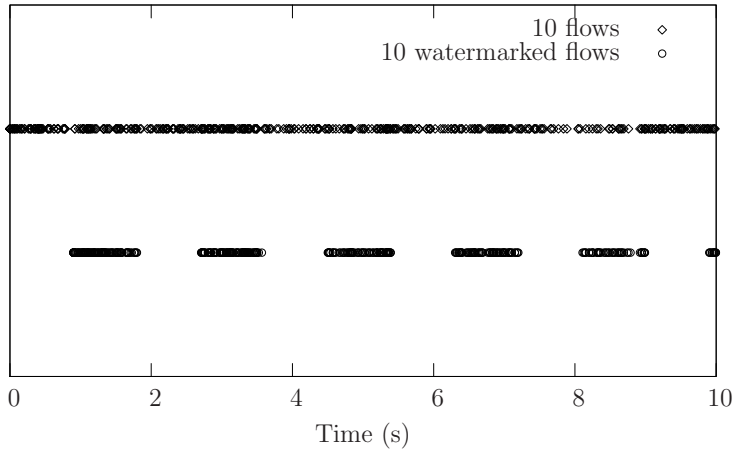
2.3.1 Watermark detection

As above, our attack relies on collecting a series of flows that are watermarked with the same value. These flows are combined into a single flow and examined for large gaps between packets. Figure 2.8a shows the packet arrivals for 10 combined flows before and after an ICBW watermark has been applied. The watermark pattern is clearly visible in the combined flows, alerting about the watermark presence. Figure 2.8b shows the same process working with the IBW watermark scheme.

We also performed the same analysis for non-interactive, bulk transfer traffic by applying the watermark to packet traces we collected from web downloads across a DSL connection. Figure 2.9a shows the packet timings for 10 combined flows before and after a watermark. Bulk transfers have a somewhat more regular behavior, since they are controlled by the TCP algorithms, rather than by individual users. This can be seen at the beginning of the 10 combined flows before watermark: the TCP slow start period results in a much lower rates for the first few seconds of the connection. However, this regularity quickly gets out of sync due to irregular network delay and response times. In the graph of 10 watermarked flows, the intervals squeezed by the watermark are readily visible. In fact, because data transfer flows are much more dense than interactive flows, the watermark is visible even on a single flow (Figure 2.9b).

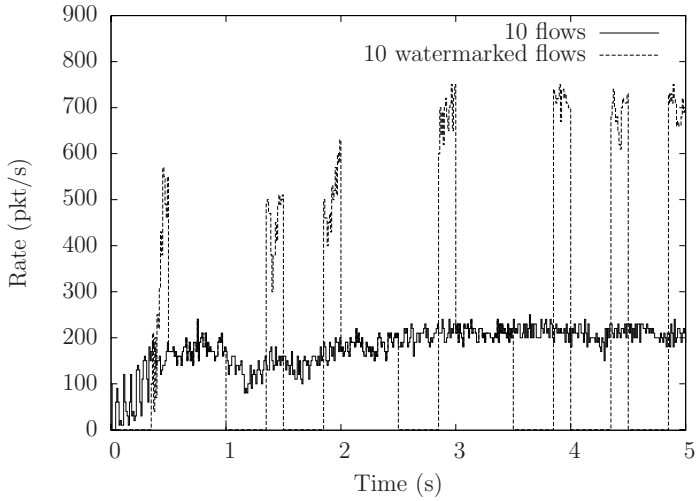


(a) Interval-Centroid Based Watermark

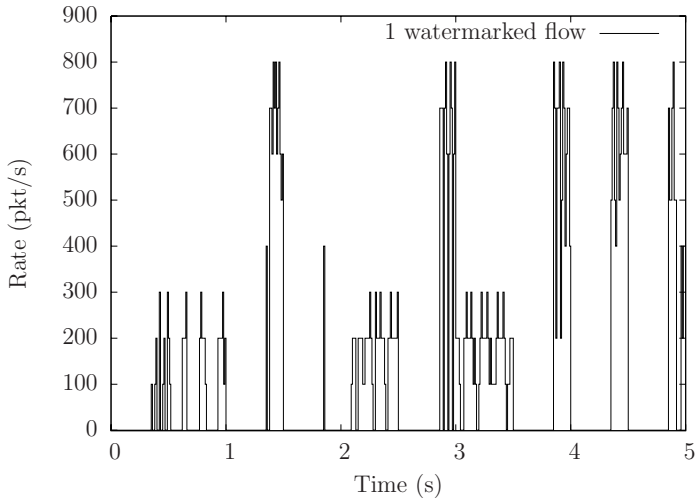


(b) Interval-Based Watermark

Figure 2.8: 10 flows before and after watermarking.



(a) Watermark on 10 flows



(b) Watermark on 1 flow

Figure 2.9: Watermark detection on bulk traffic.

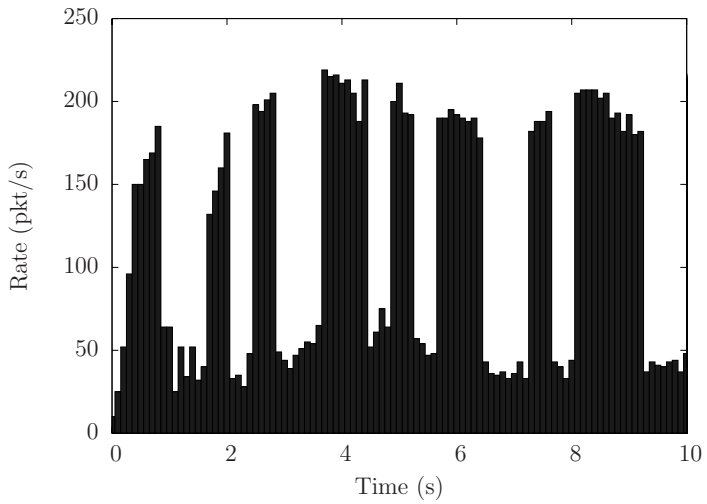


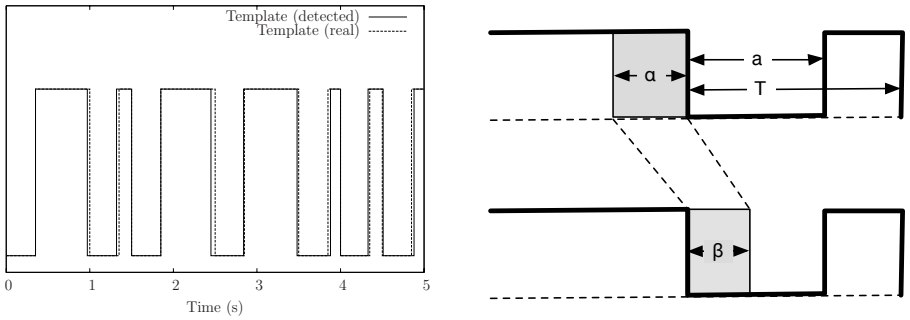
Figure 2.10: Average rate of 10 flows after DSSS watermark.

The DSSS watermark is intended to be applied to bulk transfer traffic such as FTP, since it interferes with traffic rate, rather than changing packet timings. A similar muiti-flow attack works against DSSS as well, as shown in Figure 2.10. (We used the parameters of chip length 0.4s, chip sequence length of 7, and code length of 7.) In this case, periods of high interference are clearly seen as low-rate periods in the flows, allowing one to recover the chip sequence and then decode the watermark.

2.3.2 Watermark removal

Based on the combined graphs, it is easy to recover the watermark parameters as well. We can build a template of clear intervals by selecting all intervals larger than a threshold; for example, Figure 2.11a shows the template derived from 10 flows watermarked by ICBW. The estimated template is somewhat imprecise, due to network jitter, as well as the fact that small (10–20ms) gaps may precede or follow the clear intervals even when 10 flows are combined. However, this imprecision is not a problem since the watermark can still be effectively removed. The template also lets us estimate the values of T and a. We can average the lengths of clear intervals and the distance between two consecutive clear intervals to obtain a relatively precise estimate. Armed with this information, we can then modify a new flow to remove the watermark.

For ICBW, we have two choices: we can either shift traffic into the clear



(a) Template of clear intervals (b) Shift to remove watermark

Figure 2.11: Watermark removal.

intervals in the template, thereby negating the squeezing action of the watermark, or find intervals that have not been squeezed and squeeze them. We decided to implement the former approach since it does not require as precise an estimate of T. Also, it leaves the flow looking more natural. Our shift is implemented as shown in Figure 2.11b, by shifting all packets in a period α before the clear interval into an interval of length β inside the clear interval. Larger values of α and smaller values of β will more significantly shift the interval centroid back in a different direction; however, very small values of β may not have the desired effect, since the template is imprecise and too many packets may get shifted without arriving into the correct interval. Experimentally, we found that $\alpha = 0.9(\hat{T} - \hat{a})$ and $\beta = 0.8(\hat{T} - \hat{a})$ provides best results, where $\hat{T}$ and $\hat{a}$ are estimated values of T and a.

Table 2.1 shows the results of watermark removal. We reimplemented the ICBW detection mechanism and computed the Hamming distance of the encoded watermark to the detected one, collected over 100 flows. (We show the average distance, with range shown in parentheses.) With as few as 10 flows, we are able to get a reasonably good estimate of T and a and remove the watermark in most cases—the ICBW detection scheme uses a Hamming distance threshold of 5–8 to decide when a watermark has been detected. With 15 flows, we get a more accurate template and estimate, and all 100 flows will clear the template.

A similar approach can be used to attack the IBW watermark; by delaying packets so that they fall into the clear intervals, the clear intervals become indistinguishable from loaded ones. Table 2.2 shows the effect of applying our attack on the IBW watermark, where 24 bits are encoded at different levels

Table 2.1: Results for removing ICBW watermarks

Num flows	$\hat{T}$	$\hat{a}$	**Hamming not watermarked**	**Hamming watermarked**	**Hamming attacked**	**Ave. delay**	**Max delay**
10	365 ($\sigma = 10.7$)	492 ($\sigma = 15.2$)	17.9 (13–24)	2.67 (1–7)	13.9 (2–20)	33.6	164
15	353 ($\sigma = 0.60$)	504 ($\sigma = 1.62$)	17.6 (13–25)	2.74 (0–6)	16.1 (12–21)	42.6	188.2
20	346 ($\sigma = 0.30$)	504 ($\sigma = 0.50$)	17.2 (12–21)	2.68 (0–5)	16.4 (11–20)	45.4	194.3

Table 2.2: Watermark bits detected before and after applying the attack (watermark length is 24).

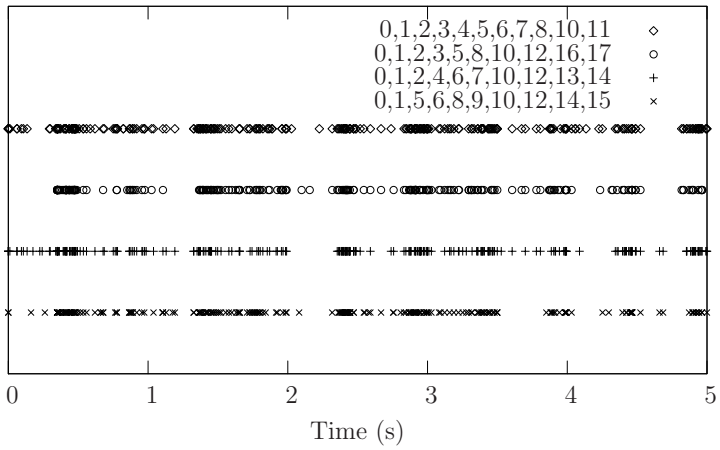
Rep.	Bits detected		Marked packets
	Before attack	After attack	
1	7	3	53
5	14	5	156
10	24	4	505
15	24	2	754
20	24	2	967
24	24	2	1209
30	24	2	1440
35	24	2	1724
41	24	2	2008
45	24	2	2307
50	24	2	2697
55	24	2	3083
60	24	2	3296
65	24	2	3623
70	24	2	3876
75	24	2	4090
80	24	2	4343

of redundancy. Even with a redundancy of 80, most bits are not recovered correctly. These results were obtained by using the code provided by the authors of [10].

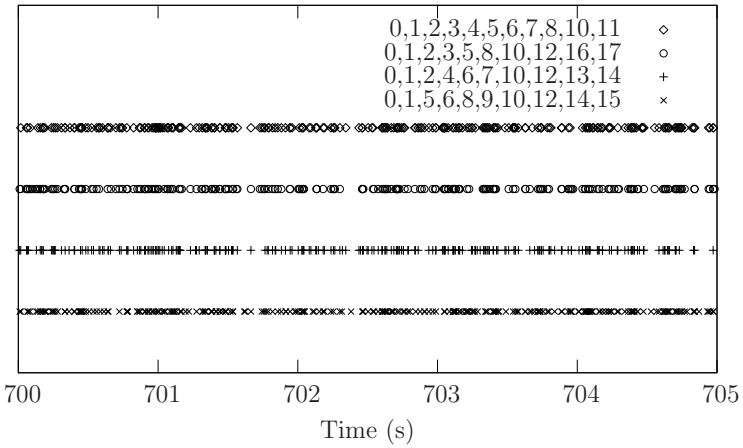
We expect a similar technique should work against DSSS watermarks; a template of low rates can be inferred from several flows. An attacker can then decrease rates in the non-interference section of the template by dropping packets, or increase the rate in the high-interference section by delaying packets into the template. We do not have experimental results for DSSS since the detection algorithm is fairly complex and we did not have access to an implementation of it.

2.3.3 Multiple values

So far we have assumed that the watermarks on all of the aggregated flows are the same. Here, we consider the case where each watermark uses multiple, different values. We can still execute our attack by relying on the fact that



(a) Watermarked flow subsets



(b) Un-watermarked flow subsets

Figure 2.12: Searching subsets of flows by MFA.

within a collection of $2k - 1$ flows, for any given bit b, we can find k flows where this bit has the same value (we have further discussed this in [31] and [41]).

Figure 2.12a plots the result of such a subset search. By inspection, we can see that in the first subset of flows, the interval (4.5,4.85) has been cleared. In the second subset, this interval remains cleared and the interval (0,0.35) becomes clear as well. The third subset has no packets in (2.0,2.35) and the fourth in (3.5,3.85). Note that this pattern immediately lets us detect the presence of a watermark; Figure 2.12b shows the same flow subsets on an unwatermarked section.

Recovery of the secret parameters can proceed largely as in the single-

Table 2.3: Blank intervals from subset of flows

(a) All blank intervals

Start	End
2.08	2.32
3.50	3.85
4.03	4.25
5.13	5.33
11.59	11.85
18.14	18.37
19.56	19.79
25.58	25.82
30.06	30.34
34.08	34.35
...	...

(b) Largest blank intervals

Start	End
130.98	131.35
140.49	140.86
151.99	152.36
161.99	162.35
235.99	236.37
306.49	306.86
334.49	334.86
368.49	368.86
43.99	44.36
51.98	52.35
...	...

value case. One difficulty is that with the flow subsets, we may encounter large intervals that are not precisely aligned with the interval positions. For example, Table 2.3a lists the blank intervals longer than 0.2s in the last subset. There are a lot of wrong-size intervals that result from the case when 8 or 9 of the flows in the subset have had an interval squeezed, but the last one or two add a few packets to the mix. To address this concern, we can select the largest empty intervals in any subset, as shown in Table 2.3b. These will correspond to intervals that have been squeezed on every flow. This can be used to recover the watermark parameters of T and a.

Once these are obtained, the next step is to scan through all subsets and determine which intervals are always squeezed at the same time and call such lists S_i; these will correspond to either A_b or B_b for some bit b. Then, for each S_i, we find S_j such that S_i and S_j are never squeezed at the same time. This will tell us that S_i and S_j correspond to the same bit. Armed with this knowledge, we can remove the watermark by observing the watermarked stream for a short while, and when we see intervals from S_i that are being squeezed, we proceed to artificially squeeze intervals in S_j (or unsqueeze further intervals in S_i, or both).

2.4 Countermeasures

We next consider several countermeasures to our attack.

2.4.1 Multiple offsets

A watermark can be inserted at an offset o from the start of the stream. This offset is picked randomly from the range $[0, o_{max}]$; [10] suggested to use $o_{max} = T$. An offset watermark can still be detected by enumerating different offsets and choosing the one with the highest detection result. This will increase the false positives, in proportion to o_{max}, but overall [10] reports that such a scheme still has good performance.

Since an offset is chosen randomly for each stream, it complicates the multi-flow attack because the watermark insertion points no longer line up with one another. It becomes necessary to search for optimal alignments by trying multiple offsets for different streams. A simple approach is to select a step value δ and choose offset values from: $(0, \delta, 2\delta, ..., \lceil o_{max}/\delta \rceil \delta)$. The attacker will need to enumerate through each of these values for each stream out of k, evaluating $(\lceil o_{max}/\delta \rceil + 1)^k$ possibilities in all.[3]

Each target alignment might be imperfect, but it is easy to see that, for some choice of offset for each stream, the misalignment will be bounded by $\delta/2$. Therefore, we must search for clear intervals of length $\ell = T - \delta/2$. We can therefore bound the probability of false positives in the overall process by:

$$P_{FP} \leq \left(\left\lceil \frac{o_{max}}{\delta} \right\rceil + 1 \right)^k P'_E(L, \ell, e) \tag{2.7}$$

where L is the maximum length of the streams and e is number of required empty intervals for watermark detection ($P'_E(L, \ell, e)$ is analyzed in Section 2.2.3).

Figure 2.13 illustrates the corresponding false positive error rate for different number of flows k when the maximum offset value is $o_{max} = 10T$ and the step value is $\delta = T$. Comparing with using only a single offset (Figure 2.7),

[3]The computational requirements can be reduced by eliminating from consideration any combinations that can be shown to lack the necessary clear intervals in a subset of all streams. E.g., if the first two streams have no intersecting clear intervals that are long enough with offsets $(0, 0)$, it is not necessary to consider combinations with other stream at offsets $(0, 0, \ldots)$.

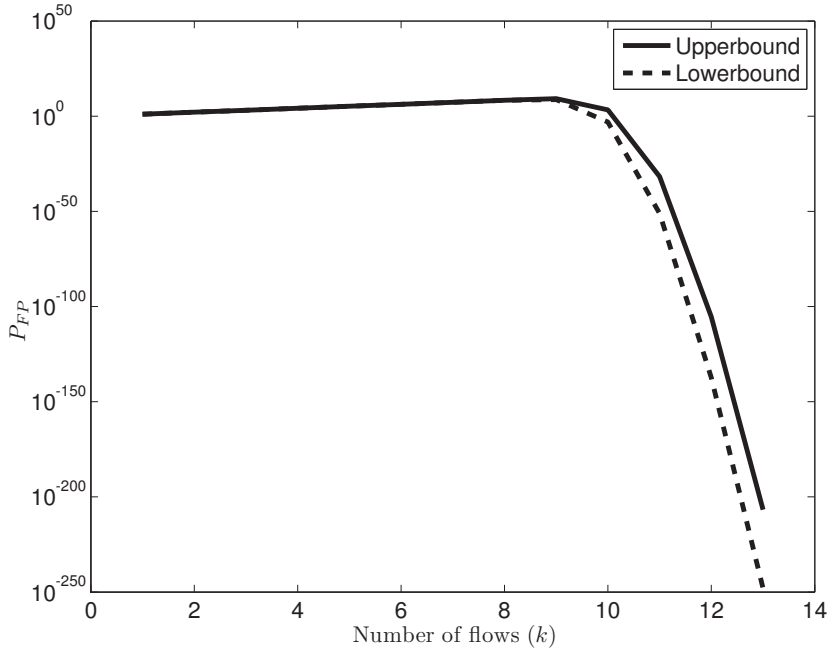


Figure 2.13: False positive errors of MFA for different number of aggregated flows when multiple offsets are used ($o_{max} = 10T$, $\delta = T$).

we can see that the multi-flow attack is still effective, at the cost of more computation for the attacker and requiring more (approximately twice) watermarked flows for the same performance. It should also be mentioned that this also increases the false positive of the watermark detector by a factor of $o_{max}/\delta = 10$. Note that larger o_{max} increases the attacker's false positive and also the computation, but requires longer flows to insert the watermark.

2.4.2 Multiple positions

Another alternative is to choose different positions, in the case of ICBW and IBW, and different PN codes in the case of DSSS [41]. Let us consider the case of ICBW. A watermarker and detector must use the same assignment of intervals to the sets A_i and B_i, as determined by the random seed s, in order for the watermark to be successfully recovered. However, a watermarker may decide to use multiple seed values, $s_1, \ldots, s_n$, and pick one of them at random for each flow.

To deal with this, the detector would need to try to recover the watermark with each possible s_i and pick the best match. Once again, the probability of

error grows with n, but increased redundancy can again be used to make up for it. Note that the probability of error falls exponentially with increased redundancy, but grows only roughly linearly with n.

We can once again use the subset attack to try to find k flows that use the same seed value s_i; however, the complexity grows quickly out of control. The probability of a given set of k flows using the same seed is $\left(\frac{1}{n}\right)^{k-1}$, which falls quite quickly even when $k = 10$. By the pigeon hole principle, within $n(k-1)+1$ flows we can always find a subset of k flows with the same seed, but the search space of all $\binom{n(k-1)+1}{k}$ subsets grows superexponentially in n. For example, with $n = 6$ and $k = 10$, $\binom{51}{10} > 10^{10}$, resulting in an infeasible number of subsets to enumerate.

The same principle can apply to IBW, by picking multiple sets of positions $\{s_i\}$, and to DSSS by using multiple PN codes [32].

2.5 Conclusions

We have demonstrated an attack on three recent network flow watermarking schemes that is highly successful, while requiring a low amount of resources. Our attack, MFA, is based on a solid theoretical grounding, and has been validated with a prototype implementation tested against the original prototypes. MFA can detect the presence of the watermark on a watermarked flow and remove it successfully. Additionally, in case of IBW scheme we can also recover the watermark parameters and values, allowing us to modify the watermark or insert it into other streams, confusing the detector. We have also suggested two countermeasures to our attack — switching bit positions and using different offset values. These countermeasures can impose a very high computation cost and therefore disable the attack.

While the use of network flow watermarking techniques for various security applications is quite new [9–11, 13, 14, 42], digital watermarking and specifically multimedia watermarking is a nearly mature field. Indeed most of network flow watermarking schemes are inspired by multimedia watermarks. To name a few: Wang and Reeves's [9] scheme is a special instance of QIM watermarking, a well-understood multimedia watermarking technique [43]. IBW scheme of Pyun et al. [10] that we have broken is based on patchwork watermark of Bender et al. [44] and the scheme of Yu et al. [14] is based on

spread spectrum watermarking [45].

The current approach for designing network flow watermarks suffers from the fact that while watermarking schemes are inspired by the digital watermarking schemes, little attention is given to the entirety of the watermarking design problem. For example, statistical characteristics of the underlying media are always an important consideration in digital watermarks, but network watermark research does not adequately model the effect that network traffic characteristics have on watermarks; as we showed, the density of bulk traffic makes it very difficult to insert a transparent watermark. Likewise, digital watermarks have long considered the possibility that multiple watermarked documents can be used to attack watermarks [45, 46], but we are unaware of previous work looking at the multi-flow threat model for watermarking. We thus hope that future work on watermarks will be informed by our work and perform a broader analysis.

CHAPTER 3

NONBLIND WATERMARKING

Linking network flows is an important problem in intrusion detection as well as anonymity. Passive traffic analysis can link flows but requires long periods of observation to reduce errors. Watermarking techniques allow for better precision and blind detection, but they do so by introducing significant delays to the traffic flow, enabling attacks that detect and remove the mark, while at the same time slowing down legitimate traffic. We propose a new, non-blind watermarking scheme called RAINBOW that is able to use delays hundreds of times smaller than existing watermarks by eliminating the interference caused by the flow in the blind case. As a result, our watermark is invisible to detection, as confirmed by experiments using information-theoretic detection tools.

We analyze the error rates of our scheme based on a mathematical model of network traffic and jitter. We also validate the analysis using an implementation running on PlanetLab. We find that our scheme generates orders of magnitude lower rates of false errors than passive traffic analysis, while using only a few hundred observed packets. We also extend our scheme so that it is robust to packet drops and repacketization and show that flows can still be reliably linked, though at the cost of somewhat longer observation periods.

3.1 Introduction

Internet attackers commonly relay their traffic through a number of (usually compromised) hosts in order to hide their identity. Detecting such hosts, called stepping stones, is therefore an important problem in computer secu-

The research presented in this chapter is done in a collaboration with Prof. Negar Kiyavash, from UIUC.

rity. The detection proceeds by finding correlated flows entering and leaving the network. Traditional approaches have used patterns inherent in traffic flows, such as packet timings, sizes, and counts, to link an incoming flow to an outgoing one [1,3,6–8]. More recently, an active approach called *watermarking* has been considered [9,10]. In this approach, traffic characteristics of an incoming flow are actively perturbed as they traverse some router to create a distinct pattern, which can later be recognized in outgoing flows. These techniques also have relevance to anonymous communication, as linking two flows can be used to break anonymity, and both passive traffic analysis [27,47] and active watermarking [12–14] have been studied in that domain as well.

The choice between passive and active techniques for traffic analysis exhibits a tradeoff. Passive approaches require observing relatively long-lived network flows, and storing or transmitting large amounts of traffic characteristics. Watermarking approaches are more efficient, with shorter observation periods necessary. They are also *blind*: rather than storing or communicating traffic patterns, all the necessary information is embedded in the flow itself. This, however, comes at a cost: to ensure robustness, the watermarks introduce large delays (hundreds of milliseconds) to the flows, interfering with the activity of benign users, and making them subject to attacks [29,31].

Motivated by this, we develop a new scheme for linking flows, called RAINBOW. As with passive techniques, our scheme will record traffic timings of incoming flows and correlate them with outgoing flows. However, we also insert a watermark value by delaying some packets. As the watermark is generated independently of the flows, this will diminish the effect of natural similarities between two unrelated flows, and allow a flow linking decision to be made over a much shorter time period. We use spread-spectrum techniques to make our delays much smaller than previous work. We use delays that are on the order of only a few milliseconds; this means that our watermarks not only do not interfere with traffic patterns of normal users, they are also virtually *invisible*, since the delays are of the same magnitude as natural network jitter.

We thoroughly analyze the detection performance of the RAINBOW non-blind watermark, and compare it with that of passive traffic analysis schemes. By using *hypothesis testing* mechanisms from the detection and estimation theory [48], we find the optimum detection schemes for RAINBOW as well as the optimum passive approach for different models of the network traffic.

Modeling the real-world network traffic is a complicated problem as it depends on many different parameters; as a result, we only consider two extreme models of the network traffic depending on the degree of traffic correlations: 1) independent flows where each flow is modeled as a Poisson process (traffic model A), and 2) fully correlated flows where all flows are considered to have similar timing patterns (traffic model B). As models A and B correspond to the most correlated and the least correlated traffic types, we assume that any real-world traffic, e.g., web traffic, lies in the middle of these two extreme models. Since false positive error rates of traffic analysis depend on intrinsic flow correlations, we argue that the detection performance of a given traffic analysis scheme is upper-bounded by its performance under traffic models A and lower-bounded by its performance under traffic model B. Hence, in this chapter we only analyze a scheme's performance under two models A and B. Our analysis shows that even though the optimum passive detector performs as well as a watermark detector for uncorrelated traffic (traffic model A), it results in large false positive error rates in the case of correlated flows (model B). This is while the optimum watermark detector performs well for both of the traffic models A and B. This *justifies the use of non-blind watermarks over passive traffic analysis*, as a main conclusion of this research. We validate our analysis through simulating the detection schemes on real network traces. In particular, we show that for highly correlated traffic, e.g., same webpage downloads, passive traffic analysis performs very poorly while a RAINBOW watermark is highly effective.

We validate our analysis by building a prototype implementation of our scheme. We test it by generating flows with timings taken from real SSH [17] traffic traces, and linking flows that traversed the Internet between PlanetLab [49] nodes. Our scheme performed quite well in this setting as well. Note that PlanetLab introduces significantly more jitter than would be present in an enterprise network, so in practice, much lower watermark delays, or smaller packet sizes, can be used. We also analyze the invisibility of our scheme by subjecting it to several information-theoretic detection tools [29, 50].

We also extend our scheme to handle dropped or inserted packets. Such changes to flows will occur naturally due to packet losses, retransmissions, or repacketization. By adjusting our scheme to perform *selective correlation*, where packets that do not match up between the incoming and outgoing

flows are dropped, our scheme can be made robust to packets being inserted and deleted, though at the cost of either longer observation periods or higher watermark amplitude.

The rest of this chapter is organized as follows: The proposed RAINBOW scheme is presented in Section 3.2. In Section 3.3, we use the *detection and estimation theory* to analyze performance of the proposed scheme. Section 3.4 provides simulations results for RAINBOW, and we provide the implementation results in Section 3.5, validating the analysis. In Section 3.6 we extend RAINBOW by introducing *selective correlation* to make it robust to flow modifications. Discussions on watermark invisibility are presented in Section 3.7, and the chapter concludes in Section 3.8 along with some future research directions.

3.2 RAINBOW Watermark

We next present the design of a new watermark scheme we call RAINBOW, for Robust and Invisible Non-Blind Watermark. Our scheme is robust (to passive interference) and invisible. However, to achieve invisibility while maintaining detection efficiency, we make the scheme *non-blind*; that is, incoming flow timings are recorded and compared with the timings of outgoing flows. This allows us to make a robust watermark test with even low-amplitude watermarks.

The RAINBOW watermark embedding process is shown in Figure 3.1. Suppose that a flow with the packet timing information $\{t_i^u | i = 1, .., n+1\}$ enters the border router where it is to be watermarked (we use the superscript u to denote an "unwatermarked" flow). Before embedding the watermark, the inter-packet delays (IPDs) of the flow, $\tau_i^u = t_{i+1}^u - t_i^u$ are recorded in an IPD database, which is accessible by the watermark detector. The watermark is subsequently embedded by delaying the packets by an amount such that the IPD of the ith watermarked packet is $\tau_i^w = \tau_i^u + w_i$. The watermark components $\{w_i\}_{i=1}^n$ take values $\pm a$ with equal probability. The value a is chosen to be small enough so that the artificial jitter caused by watermark embedding is invisible to ordinary users and attackers.[1]

[1]Throughout this chapter, by attacker we mean the attacker to the watermarking scheme.

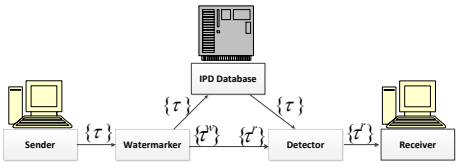


Figure 3.1: Model of RAINBOW network flow watermarking system.

In order to apply watermark delays on the flow, output packet t_i is delayed by $w_0 + \sum_{j=1}^{i-1} w_i$, where w_0 is the initial delay applied to the first packet. This results in $\tau_i^w = \tau_i^u + w_i$, as desired. Since we cannot delay a packet for a negative amount of time, w_0 must be chosen large enough to prevent this from happening. Since the sequence w_i is generated from a random seed, the watermarker can calculate all of the partial sums $\sum_{j=1}^{i-1} w_i$ in advance and adjust w_0 accordingly. If a particular random seed requires a very large initial delay w_0, a different seed can be chosen.

As the flow traverses the network, it accumulates extra delays. Let d_i be the delay that the packet accumulates by the time it reaches the watermark detector; i.e., the packet is received at the detector at time $t_i^r = t_i^w + d_i$. The IPD values at the detector are then:

$$\tau_i^r = t_{i+1}^r - t_i^r = \tau_i^u + w_i + \delta_i \tag{3.1}$$

where $\delta_i = d_{i+1} - d_i$ is the jitter present in the network.

As mentioned before, the RAINBOW scheme is non-blind and therefore the detector has access to the IPD database where the unwatermarked flows are recorded. Given an observed flow at the detector with IPDs $\tau^{\mathbf{r}}$ and a previously recorded flow $\tau^{\mathbf{u}}$, the detector must decide whether the two flows are linked or not. In the next section we derive the optimum detectors for the RAINBOW watermaks according to the LRT ruls. We also derive the optimum passive detectors, showing that the RAINBOW watermark performs significantly better than passive traffic analysis for correlated network flows.

3.3 Detection approaches

RAINBOW is the first *non-blind* flow watermarking scheme. Non-blind watermarking inherits similar scalability issues from the passive traffic analysis. In this section, we show how non-blind watermarking improves the traffic analysis performance as compared to the traditional passive traffic analysis.

We derive *optimum* Likelihood Ratio Test (LRT) detectors for the RAINBOW watermarking scheme for different traffic models, and compare its detection performance with those of optimum passive detectors. We show that RAINBOW outperforms passive traffic analysis for different traffic models; this confirms what we expect intuitively from information theory, as a non-blind watermark detector has access to more information (the watermark and the IPDs), compared to a passive detector which only has access to the IPDs. We also show that the RAINBOW detector is reliable in different models, while the optimum passive detector fails in some scenarios.

As the extreme models, we perform our detection analysis for two traffic models:

- traffic model A: independent flows with i.i.d. inter-packet delays, and,
- traffic model B: completely-correlated flows.

As it is infeasible to evaluate the detection performance for all different traffic models, we discuss the detection performance for these two traffic models, and consider any real-world network flow to lie between these two extreme models. We show that an active detector, i.e., RAINBOW, is reliable for different models, while a passive detector fails for certain traffic models.

3.3.1 Detection primitives

We use *hypothesis testing* [48] to analyze the detection performance of active and passive detectors. For an active detector, we aim to distinguish between the two following hypotheses:

- H_0 (*null hypothesis*): the received flow with IPDs $\tau^{\mathbf{r}}$ is a new, unwatermarked flow, unlinked to the flow with IPDs τ, *and*,

- H_1: $\tau^{\mathbf{r}}$ is the result of a flow with original IPDs τ being watermarked and passed through the network.[2]

Also, for a passive detector we consider the following hypothesis testing problem:

- H_0 (*null hypothesis*): the received flow with IPDs $\tau^{\mathbf{r}}$ is a new flow, unlinked to τ (the IPDs of another received flow), *and*,
- H_1: $\tau^{\mathbf{r}}$ is the result of τ passing through the network.

We find the *optimum* likelihood-ratio tests (LRT) of these hypothesis testing problems. For any received flow with $\tau^{\mathbf{r}}$ IPDs, an LRT test evaluates a test metric for the IPDs, $T[\tau^{\mathbf{r}}]$, and compares it with a *detection threshold* η; if $T[\tau^{\mathbf{r}}] \geq \eta$, the received flow is said to be *linked* to the one in the detector's database (with IPDs of τ). We can therefore express the false positive and false-negative rates of the detector as:

$$P_{\text{FP}} = P(T[\tau^{\mathbf{r}}|\mathbf{H_0}] \geq \eta) \tag{3.2}$$

$$P_{\text{FN}} = P(T[\tau^{\mathbf{r}}|\mathbf{H_1}] < \eta) \tag{3.3}$$

3.3.2 Network jitter model

We will model network delays as i.i.d. exponential, which implies that the jitter (difference of two delays) is i.i.d. according to a zero-mean Laplace distribution denoted by $Lap(0, b_\delta)$, where $2b_\delta^2$ is the variance of the jitter. Of course, in a real network, delays will have some correlation; we compare the probability density function (PDF) of real observed jitter on a connection over PlanetLab [49] with a best-fit Laplace distribution in Figure 3.2. We can see that the real PDF has greater support at 0, and the Laplace distribution has a heavier tail. This means that our analysis of error rates will be conservative, since 0 jitter will result in no error for our detection scheme. We have also conducted similar experiments with the same results on Tor anonymous network [5] to consider the other application of watermarking.

[2]Note that there is another possibility, namely that $\tau^{\mathbf{r}}$ is a *watermarked* flow, but not corresponding to τ. However, we ignore this case because errors in this scenario do not matter: if the flow is said to be watermarked, then the detection algorithm is correct, and if it is said to be unwatermarked, it will later be tested against the correct τ.

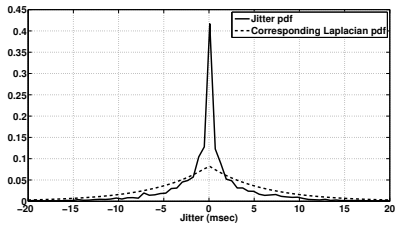


Figure 3.2: A comparison of observed jitter and a fitted Laplace distribution.

3.3.3 Traffic model A: independent flows, i.i.d. IPDs

In this model, we assume that the candidate flows are independent. Also, each flow has i.i.d. IPDs, i.e., the flow is modeled with a Poisson process. This represents a good model for non-interactive network flows.

Passive detection (PASSV scheme)

In this section, we find the optimum likelihood ratio (LRT) passive detector for the traffic model A. Suppose that the flow with IPDs τ is known to the detector. Detector will need to check if it is correlated with some received flow τ^*, where τ and τ^* are independent. So, in this case the hypothesis testing problem is:

$$\begin{cases} H_0: & \tau_i^r = \tau_i^* + \delta_i^0 \\ H_1: & \tau_i^r = \tau_i + \delta_i^1 \end{cases} \tag{3.4}$$

where δ^0 and δ^1 represent the network jitter. Based on our measurements over the Planetlab we model the network jitter with an i.i.d. Laplace distribution $Lap(0, b)$ (see Section 3.3.2).

In order to find the optimum LRT detector, we first need to find the PDF function of τ_i^r in different hypotheses, i.e., $p_i(\cdot)$ for hypothesis H_i . As the model A suggests, we model the IPDs τ^* as i.i.d. exponential distribution. So, in hypothesis H_0 the received signal τ_i^r is the summation of a Laplace and an exponential random variable. We have that the summation of an exponential random variable $X \sim Exp(\lambda)$ and a Laplace distribution $Y \sim$

$Lap(0, b)$, i.e., $Z = X + Y$, is given by [51]:

$$f_Z(z) = \begin{cases} \frac{\lambda}{2(\lambda b-1)} e^{-\frac{z}{b}} + \frac{\lambda}{1-\lambda^2 b^2} e^{-\lambda z} & z \geq 0 \\ \frac{\lambda}{2(\lambda b+1)} e^{\frac{z}{b}} & z < 0 \end{cases} \tag{3.5}$$

We use this to find $p_0(\cdot)$:

$$p_0(\tau_i^r) = \begin{cases} \frac{\lambda}{2(\lambda b-1)} e^{-\frac{\tau_i^r}{b}} + \frac{\lambda}{1-\lambda^2 b^2} e^{-\lambda \tau_i^r} & y_i \geq 0 \\ \frac{\lambda}{2(\lambda b+1)} e^{\frac{\tau_i^r}{b}} & y_i < 0 \end{cases} \tag{3.6}$$

In the case of H_1, since the τ_i is known to the detector, we can model τ_i^r as a Laplace distribution with mean τ_i. So:

$$p_1(\tau_i^r) = \frac{1}{2b} e^{-\frac{|\tau_i^r - \tau_i|}{b}} \tag{3.7}$$

Note that even though the real-world IPDs can never be negative, the densities p_0 and p_1 return a non-zero density for negative values of the IPDs. In fact, this is due to the approximation we make in modeling the network jitter as a two-sided Laplace distribution, and its effect is very small for ordinary network flows based on our simulations [11].

Having the densities p_0 and p_1, we derive the optimum detector based on the likelihood ratio test to be:

$$L(\tau^{\mathbf{r}}) \gtrless_{\mathbf{H_0}}^{\mathbf{H_1}} \mathbf{e}^{\eta} \tag{3.8}$$

where η is the LRT detection threshold and

$$L(\tau^{\mathbf{r}}) = \prod L_i(\tau_i^r) \tag{3.9}$$

$$L_i(\tau_i^r) = \frac{p_1(\tau_i^r)}{p_0(\tau_i^r)} \tag{3.10}$$

We define $\eta_n = \eta/n$ as the *normalized detection threshold*. A value of of $\eta_n = 0$ results in a MiniMax detector.

Detection performance Let us consider the case where the detector uses the PASSV detection scheme in order to link a received flow with IPDs $\boldsymbol{\tau^r}$ to a known flow with IPDs $\boldsymbol{\tau}$, i.e., a registered flow. Considering the assumptions

made in the traffic model A, i.e., the IPDs being i.i.d., we use the Chernoff bound for i.i.d. signals [48] to find the false positive (P_{FP}) and false negative (P_{FN}) error rates of the PASSV detector:

$$P_{FP}^{\boldsymbol{\tau}} \leq \prod_{i=1}^{n} e^{-(s\eta_n - \mu_{0,i}^{\tau_i}(s))} \tag{3.11}$$

$$P_{FN}^{\boldsymbol{\tau}} \leq \prod_{i=1}^{n} e^{-((s-1)\eta_n - \mu_{0,i}^{\tau_i}(s))} \tag{3.12}$$

where $0 < s < 1$ and:

$$\mu_{0,i}^{\tau_i}(s) = \ln \int p_0^{1-s}(\tau_i^r) p_1^s(\tau_i^r) d\tau_i^r \tag{3.13}$$

The error probabilities of $P_{FN}^{\boldsymbol{\tau}}$ and $P_{FP}^{\boldsymbol{\tau}}$ correspond to a fixed known IPDs sequence, $\boldsymbol{\tau}$. The overall false errors are evaluated by averaging $P_{FP}^{\boldsymbol{\tau}}$ and $P_{FN}^{\boldsymbol{\tau}}$ with respect to $\boldsymbol{\tau}$:

$$\begin{aligned} P_{FP} &= E_{\boldsymbol{\tau}}\{P_{FP}^{\boldsymbol{\tau}}\} && (3.14)\\ &\leq \prod_{i=1}^{n} E_{\tau_i}\left\{e^{-(s\eta_n - \mu_{0,i}^{\tau_i}(s))}\right\} && (3.15)\\ &= \left(\int_0^{\infty} e^{-(s\eta_n - \mu_{0,1}^{\tau_1}(s))} \lambda e^{-\lambda\tau_1} d\tau_1\right)^n && (3.16) \end{aligned}$$

$$\begin{aligned} P_{FN} &= E_{\boldsymbol{\tau}}\{P_{FN}^{\boldsymbol{\tau}}\} && (3.17)\\ &\leq \prod_{i=1}^{n} E_{\tau_i}\left\{e^{-((s-1)\eta_n - \mu_{0,i}^{\tau_i}(s))}\right\} && (3.18)\\ &= \left(\int_0^{\infty} e^{-((s-1)\eta_n - \mu_{0,1}^{\tau_1}(s))} \lambda e^{-\lambda\tau_1} d\tau_1\right)^n && (3.19) \end{aligned}$$

We can represent the upper bounds of these false errors as:

$$\begin{aligned} P_{FP} &\leq e^{-n \cdot E_{FP}(s,\eta_n)} && (3.20)\\ P_{FN} &\leq e^{-n \cdot E_{FN}(s,\eta_n)} && (3.21) \end{aligned}$$

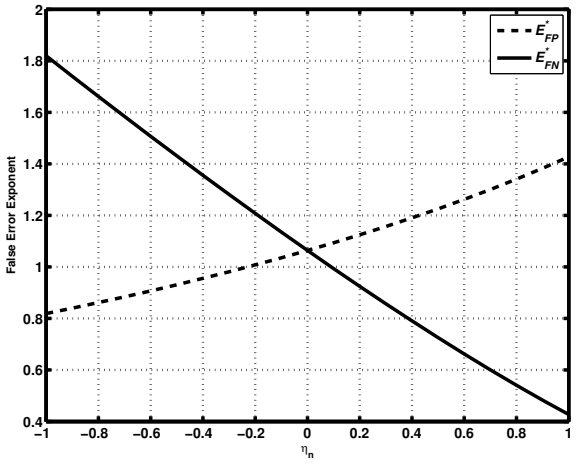


Figure 3.3: Error exponents $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ of the PASSV detection scheme for different values of η_n (traffic model A). ($b = 10^{-2}sec$, $\lambda = 5pps$)

where

$$E_{FP}(s, \eta_n) = -\ln\left(\int_0^\infty e^{-(s\eta_n - \mu^{\tau_1}_{0,1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right) \tag{3.22}$$

$$E_{FN}(s, \eta_n) = -\ln\left(\int_0^\infty e^{-((s-1)\eta_n - \mu^{\tau_1}_{0,1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right) \tag{3.23}$$

$$(0 < s < 1)$$

For each detection threshold η_n, we find the tightest exponent bounds $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ such that:

$$E^*_{FP}(\eta_n) = \max_{0<s<1} E_{FP}(s, \eta_n) \tag{3.24}$$

$$E^*_{FN}(\eta_n) = \max_{0<s<1} E_{FN}(s, \eta_n) \tag{3.25}$$

Analysis results We use Mathematica 7.0 to evaluate the false error exponents of (3.24) and (3.25). The parameters used for the simulations are $b = 10^{-2}sec$ and $\lambda = 5pps$, borrowed from [11]. Figure 3.3 plots the tightest bounds for the error exponents of $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ for different thresholds of η_n. Note that the optimum s varies with the decision threshold. For $\eta_n = 0$ the false positive and false negative errors are equal; we name this error rate as the *Cross-Over Error Rate* (COER). For the mentioned setting of the variables the COER exponent of the PASSV detector is equal to 1.06396.

Active detection (ACTV scheme)

In this section, we find the optimum LRT detector for the RAINBOW non-blind watermark for the traffic model A. We have the following hypothesis testing problem:

$$\begin{cases} H_0 : \tau_i^r = \tau_i^* + \delta_i \\ H_1 : \tau_i^r = \tau_i + w_i + \delta_i \end{cases} \tag{3.26}$$

where τ_i's are the IPDs registered in the IPD database, and τ_i^*'s are the IPDs of an independent flow. As before, in order to find the optimum LRT detector we need to find the distribution of τ_i^r in different hypotheses. Using (3.5) we find the corresponding PDF function under H_0 as:

$$p_0(\tau_i^r) = \begin{cases} \frac{\lambda}{2(\lambda b-1)} e^{-\frac{\tau_i^r}{b}} + \frac{\lambda}{1-\lambda^2 b^2} e^{-\lambda \tau_i^r} & \tau_i^r \geq 0 \\ \frac{\lambda}{2(\lambda b+1)} e^{\frac{\tau_i^r}{b}} & \tau_i^r < 0 \end{cases} \tag{3.27}$$

Since τ_i and w_i are known to the detector, we find the PDF in hypothesis H_1 as the following:

$$p_1(\tau_i^r) = \frac{1}{2b} e^{-\frac{|\tau_i^r - \tau_i - w_i|}{b}} \tag{3.28}$$

So, the optimum detector based on the likelihood ratio test is:

$$L(\tau^{\mathbf{r}}) \gtrless_{\mathbf{H_0}}^{\mathbf{H_1}} \mathbf{e}^{\eta} \tag{3.29}$$

where η is the LRT detection threshold and

$$L(\tau^{\mathbf{r}}) = \prod L_i(\tau_i^r) \tag{3.30}$$

$$L_i(\tau_i^r) = \frac{p_1(\tau_i^r)}{p_0(\tau_i^r)} \tag{3.31}$$

Detection performance As before, considering the independence of the IPDs and also the watermark bits we use the Chernoff bound [48] to find the

error probabilities of the ACTV detector for a given τ and $\mathbf{w}$:

$$P_{FP}^{\boldsymbol{\tau},\boldsymbol{w}} \leq \prod_{i=1}^{n} e^{-(s\eta_n - \mu_{0,i}^{\tau_i,w_i}(s))} \tag{3.32}$$

$$P_{FN}^{\boldsymbol{\tau},\boldsymbol{w}} \leq \prod_{i=1}^{n} e^{-((s-1)\eta_n - \mu_{0,i}^{\tau_i,w_i}(s))} \tag{3.33}$$

where $0 < s < 1$, and:

$$\mu_{0,i}^{\tau_i,w_i}(s) \;\; = \;\; \ln \int p_0^{1-s}(\tau_i^r) p_1^s(\tau_i^r) d\tau_i^r \tag{3.34}$$

As $P_{FN}^{\tau,\mathbf{w}}$ and $P_{FP}^{\tau,\mathbf{w}}$ correspond to a fixed IPDs sequence τ and the watermark $\mathbf{w}$, we evaluate the overall false errors by averaging $P_{FP}^{\tau,\mathbf{w}}$ and $P_{FN}^{\tau,\mathbf{w}}$ with respect to τ and $\mathbf{w}$:

$$P_{FP} = E_{\mathbf{w}} E_{\tau}\{P_{FP}^{\tau,\mathbf{w}}\} \tag{3.35}$$

$$\leq \prod_{i=1}^{n} E_{w_i} E_{\tau_i}\left\{e^{-(s\eta_n - \mu_{0,i}^{\tau,\boldsymbol{w}}(s))}\right\} \tag{3.36}$$

$$= \left(\frac{1}{2}\sum_{w_1=0}^{1}\int_0^{\infty} e^{-(s\eta_n - \mu_{0,1}^{\tau_1,w_1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right)^n \tag{3.37}$$

$$P_{FN} = E_{\mathbf{w}} E_{\tau}\{P_{FN}^{\tau,\mathbf{w}}\} \tag{3.38}$$

$$\leq \prod_{i=1}^{n} E_{w_i} E_{\tau_i}\left\{e^{-((s-1)\eta_n - \mu_{0,i}^{\tau,\boldsymbol{w}}(s))}\right\} \tag{3.39}$$

$$= \left(\frac{1}{2}\sum_{w_1=0}^{1}\int_0^{\infty} e^{-((s-1)\eta_n - \mu_{0,1}^{\tau_1,w_1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right)^n \tag{3.40}$$

The approximated upperbounds can be formulated as:

$$P_{FP} \leq e^{-n\cdot E_{FP}(s,\eta_n)} \tag{3.41}$$

$$P_{FN} \leq e^{-n\cdot E_{FN}(s,\eta_n)} \tag{3.42}$$

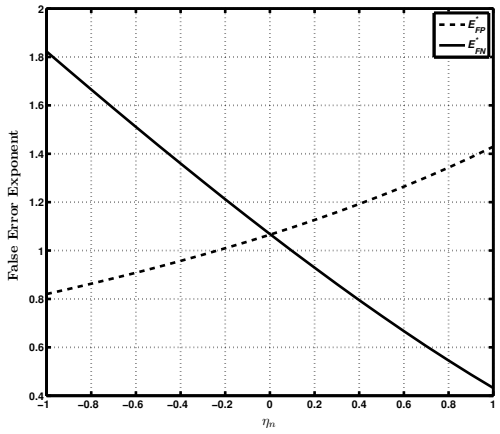


Figure 3.4: Error exponents $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ of the ACTV detection scheme for different values of η_n (traffic model A). ($b = 10^{-2}sec$, $\lambda = 5pps$)

where

$$E_{FP}(s, \eta_n) = -\ln\left(\frac{1}{2}\sum_{w_1=0}^{1}\int_0^{\infty} e^{-(s\eta_n - \mu_{0,1}^{\tau_1, w_1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right)$$

$$E_{FN}(s, \eta_n) = -\ln\left(\frac{1}{2}\sum_{w_1=0}^{1}\int_0^{\infty} e^{-((s-1)\eta_n - \mu_{0,1}^{\tau_1, w_1}(s))}\lambda e^{-\lambda\tau_1} d\tau_1\right)$$

$$(0 < s < 1)$$

Finally, the tightest bounds for each η_n are found by maximizing the error exponents with respect to the parameter s:

$$E^*_{FP}(\eta_n) = \max_{0<s<1} E_{FP}(s, \eta_n) \tag{3.43}$$

$$E^*_{FN}(\eta_n) = \max_{0<s<1} E_{FN}(s, \eta_n) \tag{3.44}$$

Analysis results Using Mathematica 7.0 we evaluate the false error exponents of (3.43) and (3.44). As before, we use the parameters $b = 10^{-2}sec$, $a = 10^{-2}sec$, and $\lambda = 5pps$ for the simulations. Figure 3.4 plots the tightest bounds for the error esponents of $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ for different thresholds of η_n. The COER exponent occurs for $\eta_n = 0$ and is equal to 1.06828, which is slightly better compared to that of the PASSV detector evaluated before, (1.06396).

3.3.4 Traffic model B: correlated flows, correlated IPDs

As the other extreme of traffic models we investigate the traffic model with correlated IPDs. We consider the case where all of the network flows have the same IPDs, e.g., for any two flows with IPDs τ^* and τ we have that $\tau_i^* = \tau_i = C_i$ for all i. In particular, this model captures the behavior of a number of widely used traffic types, including file transfers, browsing the same websites, etc.

Passive detection

In this model, a passive detection faces the following hypothesis testing problem:

$$\begin{cases} H_0: & \tau_i^r = \tau_i^* + \delta_i \\ H_1: & \tau_i^r = \tau_i + \delta_i \end{cases} \tag{3.45}$$

where $\tau_i^* = \tau_i = C_i$. The optimum LRT detector for this problem is the random guessing:

$$L(\tau^{\mathbf{r}}) = RND \tag{3.46}$$

where RND is a uniform random variable. The detection rule is:

$$L(\tau^{\mathbf{r}}) \gtrless_{\mathbf{H_0}}^{\mathbf{H_1}} \mathbf{e}^{\eta} \tag{3.47}$$

Detection performance Since the detector is based on random guessing, the false errors are as followed:

$$P_{FP} = p \tag{3.48}$$

$$P_{FN} = 1 - p \tag{3.49}$$

where $0 \leq p \leq 1$ is determined by the choice of η.

Active detection (SLCorr scheme)

In this case, we have the following hypothesis testing problem:

$$\begin{cases} H_0: & \tau_i^r = \tau_i^* + \delta_i \\ H_1: & \tau_i^r = \tau_i + w_i + \delta_i \end{cases} \tag{3.50}$$

Since $\tau_i^* = \tau_i = C_i$, this can be reduced to the following hypothesis testing:

$$\begin{cases} H_0: & y_i = \delta_i \\ H_1: & y_i = w_i + \delta_i \end{cases} \tag{3.51}$$

where $y_i = \tau_i^r - \tau_i$. The optimum LRT detector for this problem can be found considering the distribution of y_i in different hypotheses:

$$p_0^i(y_i) = \frac{1}{2b} e^{-\frac{|y_i|}{b}} \tag{3.52}$$

$$p_1^i(y_i) = \frac{1}{2b} e^{-\frac{|y_i - w_i|}{b}} \tag{3.53}$$

So, we can derive the LRT detection metric as:

$$L_i(y_i) = \frac{p_1^i(y_i)}{p_0^i(y_i)} \tag{3.54}$$

which can be expressed as:

$$\ln L_i(y_i) = \frac{1}{b}(|y_i| - |y_i - w_i|) \tag{3.55}$$

$$= \frac{2}{b} f^{SL}\left(y_i - \frac{w_i}{2}\right).sgn(w_i) \tag{3.56}$$

$f^{SL}(\cdot)$ is a *soft-limiter* with breakpoints at $-\frac{a}{2}$ and $+\frac{a}{2}$ (a is the watermark amplitude as defined before):

$$f^{SL}(x) = \begin{cases} +\frac{a}{2} & x \geq +\frac{a}{2} \\ x & -\frac{a}{2} < x < +\frac{a}{2} \\ -\frac{a}{2} & x \leq -\frac{a}{2} \end{cases} \tag{3.57}$$

Figure 3.5: Block diagram of the SLCorr detection scheme.

We can reformulate the optimum detection rule as:

$$D(\mathbf{y}) \gtrless_{H_0}^{H_1} \eta \tag{3.58}$$

where

$$D(\mathbf{y}) = \sum_{i=1}^{n} D_i(y_i) \tag{3.59}$$

and

$$\begin{aligned} D_i(y_i) &= \frac{b}{2} \ln L_i(y_i) \\ &= f^{SL}\left(y_i - \frac{w_i}{2}\right).sgn(w_i) \end{aligned} \tag{3.60}$$

We call this detector *SLCorr*, as it is composed of a soft limiter followed by a correlation block. From a communications point of view, the soft-limiter is useful in reducing the signal detection noise in channels with a Laplacian distributed noise. We will use this as the detection scheme for the RAINBOW watermark, as would be discussed later. Figure 3.5 shows the block diagram of the *SLCorr* detector. *SLCorr* is a MiniMax detector for a detection threshold of $\eta = 0$.

Detection performance The SLCorr test metric is given in (3.58) to (3.60). Let us define $f_0^i(\cdot)$ and $f_1^i(\cdot)$ as the PDF of $x_i = y_i - \frac{w_i}{2}$ in hypothesis H_0 and H_1, respectively. We have that:

$$f_0^i(x_i) = \frac{1}{2b} e^{-\frac{|x_i + \frac{w_i}{2}|}{b}} \tag{3.61}$$

$$f_1^i(x_i) = \frac{1}{2b} e^{-\frac{|x_i - \frac{w_i}{2}|}{b}} \tag{3.62}$$

Based on these, we can evaluate $p_0(\cdot)$ and $p_1(\cdot)$, namely the PDF of $D_i(y_i)$

under hypothesis H_0 and H_1, respectively:

$$p_0(D_i) = \begin{cases} \frac{1}{2}e^{-\frac{a}{b}} & D_i = +\frac{a}{2} \\ \frac{1}{2b}e^{-\frac{D_i+\frac{a}{2}}{b}} & -\frac{a}{2} < D_i < \frac{a}{2} \\ \frac{1}{2} & D_i = -\frac{a}{2} \end{cases} \tag{3.63}$$

$$p_1(D_i) = \begin{cases} \frac{1}{2} & D_i = +\frac{a}{2} \\ \frac{1}{2b}e^{\frac{D_i-\frac{a}{2}}{b}} & -\frac{a}{2} < D_i < \frac{a}{2} \\ \frac{1}{2}e^{-\frac{a}{b}} & D_i = -\frac{a}{2} \end{cases} \tag{3.64}$$

Considering that the distributions $p_0(D_i)$ and $p_1(D_i)$ are i.i.d. with i we use the Chernoff bound [48] to find the error probabilities of the SLCorr detector:

$$P_{FP} \leq e^{-n(s\eta_n - \mu_0(s))} \qquad (\forall s > 0) \tag{3.65}$$

$$\mu_0(s) = \mu_{D_i|H_0}(s)$$

$$P_{FN} \leq e^{-n(s\eta_n - \mu_1(s))} \qquad (\forall s < 0) \tag{3.66}$$

$$\mu_1(s) = \mu_{D_i|H_1}(s)$$

where $\eta_n = \eta/n$ is the normalized detection threshold. We have that:

$$\begin{aligned} \mu_0(s) = \mu_{D_i|H_0}(s) &= \ln \int_{-\infty}^{\infty} e^{sx} p_0(x) dx \\ &= \ln \left[\frac{sb}{2(sb-1)} e^{-\frac{a}{b}} e^{s\frac{a}{2}} + \frac{sb-2}{2(sb-1)} e^{-s\frac{a}{2}} \right] \end{aligned} \tag{3.67}$$

and,

$$\begin{aligned} \mu_1(s) = \mu_{D_i|H_1}(s) &= \ln \int_{-\infty}^{\infty} e^{sx} p_1(x) dx \\ &= \ln \left[\frac{sb}{2(sb+1)} e^{-\frac{a}{b}} e^{-s\frac{a}{2}} + \frac{sb+2}{2(sb+1)} e^{s\frac{a}{2}} \right] \end{aligned} \tag{3.68}$$

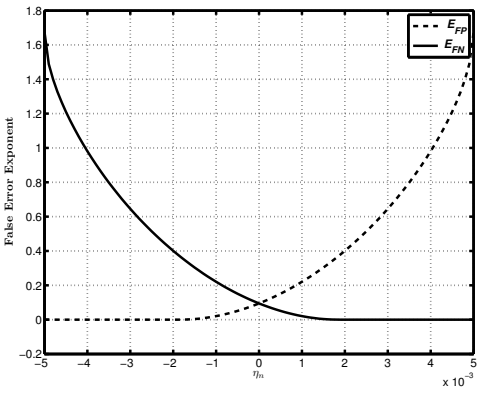


Figure 3.6: Error exponents $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ of SLCorr for different values of η_n (traffic model B). ($b = 10^{-2}sec$, $a = 10^{-2}sec$)

We can express the above P_{FP} and P_{FN} false errors as:

$$P_{FP} \leq e^{-n \cdot E_{FP}(s,\eta_n)} \tag{3.69}$$

$$P_{FN} \leq e^{-n \cdot E_{FN}(s,\eta_n)} \tag{3.70}$$

where

$$E_{FP}(s,\eta_n) = s\eta_n - \mu_0(s) \qquad (s > 0) \tag{3.71}$$

$$E_{FN}(s,\eta_n) = s\eta_n - \mu_0(s) \qquad (s < 0) \tag{3.72}$$

Finally, the tightest bounds for each η_n are found by maximizing error exponents with respect to the s parameter:

$$E^*_{FP}(\eta_n) = \max_{s>0} E_{FP}(s,\eta_n) \tag{3.73}$$

$$E^*_{FN}(\eta_n) = \max_{s<0} E_{FN}(s,\eta_n) \tag{3.74}$$

Analysis results We use Mathematica 7.0 to evaluate the false error exponents of (3.73) and (3.74). The parameters used for the simulations are $b = 10^{-2}sec$ and $a = 10^{-2}sec$. Figure 3.6 plots the tightest bounds for the error exponents of $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ for different thresholds of η_n. The COER exponent occurs for $\eta_n = 0$ and is equal to 0.0945.

3.3.5 Discussion

Above, we derived the optimum passive and active detectors for the traffic analysis problem and evaluated their performance by finding the Chernoff upperbounds of their false error rates. In this section, we use the *asymptotic relative efficiency* (ARE) as a tool to compare their detection performances.

The asymptotic relative efficiency (ARE) is a measure for comparing two discrete-time detection schemes. For two discrete detection schemes S_1 and S_2 the ARE metric is defined as $ARE_{S_1,S_2} = \lim_{n\to\infty} n_2/n$, where n is the number of S_1's samples. The n_2 parameter is the smallest number of S_2 samples that results in S_2's error rate to be smaller than or equal to the error rate of S_1 (with n samples). An ARE metric of $ARE_{S_1,S_2} > 1$ depicts that S_1 is asymptotically more efficient than S_2. Chernoff [52] finds the ARE metric of two detectors S_1 and S_2 using their Chernoff error upperbounds as:

$$ARE_{S_1,S_2} = E_1/E_2 \tag{3.75}$$

where E_1 and E_2 are the error exponents of the Chernoff upperbounds for S_1 and S_2 detectors, respectively.

Using the analysis results from Sections 3.3.3 and 3.3.4 we can derive the ARE metric of the optimum passive and active detectors for the two traffic models as:

$$ARE_{PASSV,ACTV}|A = 1.06396/1.06828 \approx 0.996 \tag{3.76}$$

$$ARE_{RND,SLCorr}|B = 0/0.0945 = 0 \tag{3.77}$$

This asserts that the optimum active detector outperforms the optimum passive detector in both traffic models A and B (which is intuitively expected from information theory). As an important observation, we see that the active detector's advantage is very small for the traffic model A, however, the active detector significantly outperforms the optimum passive detector in traffic model B, i.e., the correlated traffic. In other words, the active detector provides very good detection performance for different traffic models, however, the passive detection is very poor for the more correlated network traffic.

In the rest of this section we analyze the performance of the SLCorr scheme under the traffic model A, showing that even though SLCorr is not the opti-

mum detector for the traffic model A, however, it provides very good detection performance under this model. Based on this, we choose SLCorr as the sole detector for RAINBOW, regardless of the behavior of the network flows. This simplifies the watermark detection, as real-world traffic are combinations of the models A and B, and the detection can be performed regardless of the type of the received traffic. We also analyze the performance of PASSV and ACTV detectors under traffic model B, showing their inefficiency in this model.

SLCorr Detection performance for traffic model A

The SLCorr scheme is the optimum active detector for traffic model B, but not the traffic model A. In this section we show that SLCorr achieves a good detection performance even under traffic model A, allowing a system designer to use it as the sole detection scheme regardless of the type of the traffic. SLCorr faces the following hypothesis testing under the traffic model A:

$$\begin{cases} H_0 : \tau_i^r = \tau_i^* + \delta_i \\ H_1 : \tau_i^r = \tau_i + w_i + \delta_i \end{cases} \tag{3.78}$$

Considering SLCorr's detection metric, given in (3.58) to (3.60), one can rewrite the hypothesis testing problem as:

$$\begin{cases} H_0 : y_i = \tau_i^* + \delta_i - \tau_i \\ H_1 : y_i = w_i + \delta_i \end{cases} \tag{3.79}$$

where $y_i = \tau_i^r - \tau_i$. Let us assume $f_i^0(\cdot)$ and $f_i^1(\cdot)$ as the PDF functions of $y_i|H_0$ and $y_i|H_1$, respectively. We have that:

$$y_i|H_1 \sim Lap(w_i, b) \tag{3.80}$$

$$f_i^1(y_i) = \frac{1}{2b} e^{-\frac{|y_i - w_i|}{b}} \tag{3.81}$$

Also, based on the summation of two Laplace distributions given in [51]

we have that:

$$\delta_i \sim Lap(0, b) \tag{3.82}$$

$$(\tau_i^* - \tau_i) \sim Lap(0, 1/\lambda) \tag{3.83}$$

$$f_i^0(y_i) = \frac{b\lambda}{2(1-b^2\lambda^2)} \left(\frac{1}{b} e^{-\lambda|y_i|} - \lambda e^{-\frac{1}{b}|y_i|} \right) \tag{3.84}$$

Now, let us define $p_0(\cdot)$ and $p_1(\cdot)$ as the PDF functions of $D_i(y_i)$ under hypotheses H_0 and H_1, respectively. We derive $p(\cdot)$ as:

$$p_0(D_i) = \begin{cases} \frac{b^2\lambda^2}{2(1-b^2\lambda^2)} \left(\frac{1}{b^2\lambda^2} e^{-\lambda a} - e^{-\frac{a}{b}} \right) & D_i = +\frac{a}{2} \\ \frac{b\lambda}{2(1-b^2\lambda^2)} \left(\frac{1}{b} e^{-\lambda(D_i + a/2)} \right. & \\ \quad \left. -\lambda e^{-\frac{1}{b}(D_i + a/2)} \right) & -\frac{a}{2} < D_i < \frac{a}{2} \\ \frac{1}{2} & D_i = -\frac{a}{2} \end{cases} \tag{3.85}$$

Also, using (3.81) we derive $p_1(\cdot)$ as:

$$p_1(D_i) = \begin{cases} \frac{1}{2} & D_i = +\frac{a}{2} \\ \frac{1}{2b} e^{\frac{D_i - \frac{a}{2}}{b}} & -\frac{a}{2} < D_i < \frac{a}{2} \\ \frac{1}{2} e^{-\frac{a}{b}} & D_i = -\frac{a}{2} \end{cases} \tag{3.86}$$

Based on the $p_0(\cdot)$ and $p_1(\cdot)$ distributions and using the Chernoff bound [48] for signal detection we find the error probabilities of the detector to be:

$$P_{FP} \leq e^{-n(s\eta_n - \mu_0(s))} \qquad (\forall s > 0) \tag{3.87}$$

$$\mu_0(s) = \mu_{D_i|H_0}(s)$$

$$P_{FN} \leq e^{-n(s\eta_n - \mu_1(s))} \qquad (\forall s < 0) \tag{3.88}$$

$$\mu_1(s) = \mu_{D_i|H_1}(s)$$

where we have:

$$\mu_0(s) = \mu_{D_i|H_0}(s) = \ln \int_{-\infty}^{\infty} e^{sx} p_0(x) dx \tag{3.89}$$

$$= \ln \left[\frac{b^2\lambda^2}{2(1-b^2\lambda^2)} \left[\frac{s}{b^2\lambda^2(s-\lambda)} e^{sa/2} e^{-\lambda a} + \frac{sb}{1-sb} e^{sa/2} e^{-a/b} + \frac{-2\lambda bs + 2\lambda + sb^2\lambda^2 - b^2\lambda^3 + s^2 b - s}{(s-\lambda)(sb-1)b^2\lambda^2} e^{-sa/2} \right] \right] \tag{3.90}$$

and,

$$\mu_1(s) = \mu_{D_i|H_1}(s) = \ln \int_{-\infty}^{\infty} e^{sx} p_1(x) dx \tag{3.91}$$

$$= \ln \left[\frac{sb}{2(sb+1)} e^{-\frac{a}{b} - s\frac{a}{2}} + \frac{sb+2}{2(sb+1)} e^{s\frac{a}{2}} \right] \tag{3.92}$$

As before, we can express the above P_{FP} and P_{FN} false errors as:

$$P_{FP} \leq e^{-n \cdot E_{FP}(s,\eta_n)} \tag{3.93}$$

$$P_{FN} \leq e^{-n \cdot E_{FN}(s,\eta_n)} \tag{3.94}$$

where

$$E_{FP}(s, \eta_n) = s\eta_n - \mu_0(s) \qquad (s > 0) \tag{3.95}$$

$$E_{FN}(s, \eta_n) = s\eta_n - \mu_0(s) \qquad (s < 0) \tag{3.96}$$

Finally, the tightest bounds for each η_n are found by maximizing the error exponents with respect to the parameter s:

$$E^*_{FP}(\eta_n) = \max_{s>0} E_{FP}(s, \eta_n) \tag{3.97}$$

$$E^*_{FN}(\eta_n) = \max_{s<0} E_{FN}(s, \eta_n) \tag{3.98}$$

Analysis results We use Mathematica 7.0 to evaluate the false error exponents of (3.97) and (3.98). The parameters used for the simulations are $b = 10^{-2} sec$, $\lambda = 5 pps$ and $a = 10^{-2} sec$. Figure 3.7 plots the tightest bounds for the error exponents of $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ for different thresholds of η_n. The COER exponent occurs for $\eta_n = 9.6 \times 10^4 s$ which is equal to 0.0228.

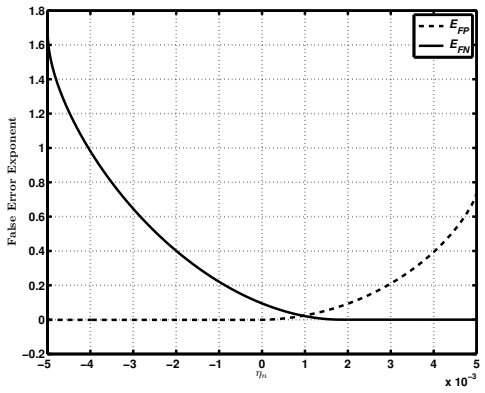


Figure 3.7: Error exponents $E^*_{FP}(\eta_n)$ and $E^*_{FN}(\eta_n)$ of SLCorr for different values of η_n (traffic model A). ($b = 10^{-2}sec$, $\lambda = 5pps$, $a = 10^{-2}sec$)

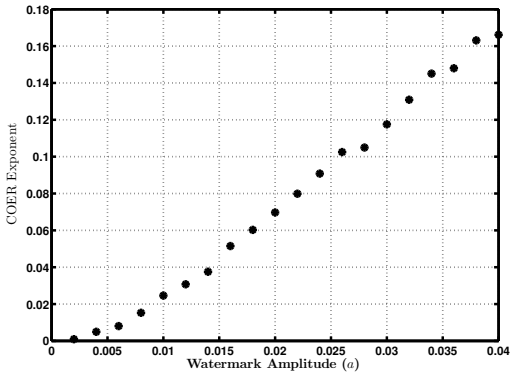


Figure 3.8: The COER error exponent of SLCorr in traffic model A for different watermark amplitudes.

Also, Figure 3.8 shows the COER exponent with respect to different values of the watermark amplitude, a. As we can see, increasing the watermark amplitude improves the detection performance (but reduces the watermark invisibility as discussed in [11]).

Detection performance of PASSV and ACTV schemes for the traffic model B: As derived before, the PASSV and ACTV schemes are the optimum passive and active detectors for the traffic model A. We show that PASSV and ACTV perform very poorly under the traffic model B, i.e., the correlated traffic. This is unlike the SLCorr detector that works well for both of the traffic models.

Under the traffic model B, the PASSV detector faces the hypothesis testing

problem of (3.45) with $\tau_i^* = \tau_i = C_i$. One can see that in this case the PASSV detection rule described in Section 3.3.3 is exactly the same for both $H0$ and $H1$ hypotheses. This means that the false positive error rate of PASSV scheme for correlated flows is equal to its true positive rate, which makes the PASSV scheme equivalent to a random guessing detector. Similarly, for the traffic model B the ACTV scheme deals with the hypothesis testing problem of (3.50) with $\tau_i^* = \tau_i = C_i$. Our analysis and simulations on Mathematica confirm that the ACTV detection metric results in very close values for the two hypothesis of $H0$ and $H1$, rendering the ACTV detection scheme ineffective for network flows in traffic model B (we skip the details due to the space constraints).

3.4 Simulation results

In this section, we evaluate the performance of the three detection schemes introduced before, i.e., SLCorr, ACTV, and PASSV, through simulating them over real-world traffic. We show that SLCorr outperforms the other detectors dealing with real-world network flows, due to the intrinsic correlations among the real-world network flows. We use the CAIDA network traces gathered January 2009 [53] for our simulations. For our simulations, we have implemented the detection schemes in C++. From the CAIDA traces we extract three types of network flows for our simulations: TCP ports of 443 (HTTPS), 25 (SMTP), and 22 (SSH). We only select flows with rates lower than 30pps (this is because the parameters of the optimum detectors depend on the rate of the flows). In all of the simulations, the detectors use the detection thresholds derived through analysis in the previous sections, i.e., 0.001 for SLCorr, 0 for ACTV, and 0 for PASSV.

In the first set of our simulations, we evaluate the false positive error rate of the three detection schemes for network flows mentioned above. For each detection scheme, we run the detection algorithm for 10000 different pairs of network flows. In order to show the effect of number of packets in the detection performance, we run the experiments for four different values of the N parameter, i.e., 25, 50, 100, and 200. Tables 3.1, 3.2, and 3.3 show the false-positive rates of the experiments along with some statistics on the detection metrics for three TCP ports of 443, 25, and 22, respectively. Re-

Table 3.1: False-positive rate of different detection schemes for port 443 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	Detection metric			False Positive
		Min	Avg	Max	
25	SLCorr	-0.005	-0.0031	0.00012	0.0068
	ACTV	-457.385	-37.8203	14.1698	0.0151
	PASSV	-245.249	-35.6167	2.8426	0.0054
50	SLCorr	-0.005	-0.0039	0.0012	0.0002
	ACTV	-503.655	-36.8637	3.9970	0.0159
	PASSV	-567.917	-45.5303	2.8297	0.0009
100	SLCorr	-0.005	-0.0042	-0.0004	0
	ACTV	-515.555	-33.2478	-2.2095	0
	PASSV	-555.857	-44.0783	2.9567	0.0023
200	SLCorr	-0.005	-0.0042	-2.5E-5	0
	ACTV	-608.838	-33.5721	0.9735	0.0005
	PASSV	-559.164	-43.2514	2.9535	0.0018

sults show that in most of the cases the SLCorr scheme results in smaller false positive errors compared to the ACTV and PASSV schemes. This is because the real network flows are deviated from the Poisson model of the traffic, due to the intrinsic dependencies among the packets of real network flows. The SLCorr detector, on the other hand, is the optimum detector for correlated network flows, which also results in reasonable detection performance for Poisson-modeled network flows. Comparing the results for the three different traffic types (Tables 3.1, 3.2, and 3.3), we observe that the ACTV and PASSV schemes perform the worst for the SSH traffic (TCP port 22); we explain this by the fact that SSH flows are more correlated compared to HTTPS and SMTP flows, as they are based on the typing behaviors of the human entities. Another general observation from the simulations is that the detection performance improves as the number of packets, N, increases.

In the second set of experiments, we run the simulated detection schemes to measure the false negative error rates. Again, we use the detection thresholds derived through the analysis in previous sections. In each simulation of the SLCorr and ACTV schemes, the candidate network flow is watermarked using the RAINBOW scheme (Section 3.2) and then a network delay is randomly selected and applied to that flow from a large pool of network delays measured over the Planetlab infrastructure [49] (the average standard deviation of the network delay is around 10ms). Likewise, for the PASSV

Table 3.2: False-positive rate of different detection schemes for port 25 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	Detection metric			False Positive
		Min	Avg	Max	
25	SLCorr	-0.005	-0.0039	0.0018	0.0008
	ACTV	-461.182	-50.3404	6.1398	0.0003
	PASSV	-364.275	-49.6125	1.8952	0.003
50	SLCorr	-0.005	-0.0042	0.0004	0
	ACTV	-359.413	-35.2567	-0.3314	0
	PASSV	-364.652	-53.7937	1.5171	0.0015
100	SLCorr	-0.005	-0.0037	-0.0007	0
	ACTV	-352.581	-31.3738	0.0420	0.0001
	PASSV	-368.304	-55.4709	1.4271	0.0013
200	SLCorr	-0.005	-0.0041	-0.0014	0
	ACTV	-190.366	-29.6399	-1.2917	0
	PASSV	-375.012	-56.3069	1.3936	0.0012

Table 3.3: False-positive rate of different detection schemes for port 22 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	Detection metric			False Positive
		Min	Avg	Max	
25	SLCorr	-0.005	-0.0029	0.0026	0.0024
	ACTV	-495.125	-18.3825	6.8506	0.0269
	PASSV	-88.1381	-8.7786	3.3239	0.1031
50	SLCorr	-0.005	-0.0038	0.0011	0.0001
	ACTV	-628.45	-20.1249	4.5654	0.0144
	PASSV	-80.5081	-9.3516	3.3204	0.0879
100	SLCorr	-0.005	-0.0037	0.0005	0
	ACTV	-522.241	-23.434	2.8119	0.0142
	PASSV	-101.337	-9.8241	3.3202	0.0861
200	SLCorr	-0.005	-0.0039	1.67E-5	0
	ACTV	-487.594	-26.357	4.7264	0.0212
	PASSV	-104.547	-9.7138	3.3195	0.0896

Table 3.4: False-negative rate of different detection schemes for port 443 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	False Negative			
		10 ms	15 ms	20 ms	30 ms
25	SLCorr	0.039	0.005	0.0004	0.0003
	ACTV	1E-04	1E-04	0	0.0004
	PASSV	0.0002			
50	SLCorr	0.0137	0.0004	0	0
	ACTV	0	0	0	0
	PASSV	0			
100	SLCorr	0.0028	0	0	0
	ACTV	0	0	0	0
	PASSV	0			
200	SLCorr	0.000977	0	0	0
	ACTV	0	0	0	0
	PASSV	0			

simulations the candidate network flow is delayed similarly to simulate the network interference. The delayed flow is then correlated with the original flow (non-delayed, and non-watermarked) using each of the detection schemes. Tables 3.4, 3.5, and 3.6 show the false negative of the experiments for the three different detection schemes, evaluated for three different TCP ports. For the watermark detection schemes of SLCorr and ACTV the experiments are repeated for four different values of the watermark amplitude, i.e., $a = 10ms, 15ms, 20ms, 30ms$. Also, all of the simulations are run for different values of the watermark length, N. Results show that by choosing reasonable parameters for the RAINBOW watermark, the SLCorr and ACTV detection schemes result in very small false-negative rates, comparable to those of the passive detection. Again, we see that increasing N improves the detection performance.

In the third set of experiments, we evaluate the false positive error rate of the three detection schemes over highly correlated network flows. More specifically, we use flow traces corresponding to web browsing activities of human entities that target the *same* destination websites at different times and from different network locations.[3] Table 3.7 shows the false positive error rates for different detection schemes for different websites and for different values of N (each simulation is averaged over 100 runs). As can be seen, in

[3]The traces are generated and provided to us by Xun Gong from UIUC

Table 3.5: False-negative rate of different detection schemes for port 25 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	False Negative			
		10 ms	15 ms	20 ms	30 ms
25	SLCorr	0.0346	0.0035	0.0007	0
	ACTV	0.0003	0.0002	0.0004	0.0002
	PASSV	0.0001			
50	SLCorr	0.0154	0.0005	0.0003	0
	ACTV	0	0	0	0.0006
	PASSV	0			
100	SLCorr	0.002636	0	0	0
	ACTV	0	0	0	0
	PASSV	0			
200	SLCorr	0	0	0	0
	ACTV	0	0	0	0
	PASSV	0			

Table 3.6: False-negative rate of different detection schemes for port 22 network flows. Each experiment is run for 10000 different pairs of flows.

N	Scheme	False Negative			
		10 ms	15 ms	20 ms	30 ms
25	SLCorr	0.028879	0.001775	0	0
	ACTV	0	0	0.00062	0.005727
	PASSV	0.0002			
50	SLCorr	0.009671	0	0	0
	ACTV	0	0	0	0
	PASSV	0			
100	SLCorr	0	0	0	0
	ACTV	0	0	0	0
	PASSV	0			
200	SLCorr	0	0	0	0
	ACTV	0	0	0	0
	PASSV	0			

most of cases, the ACTV and PASSV detection schemes result in very high false-positive rates, while the SLCorr scheme results in *no false positive error in all of the cases*. This confirms what we expect intuitively: *the PASSV and ACTV scheme are optimum passive and active detection schemes for independent network traffic models, but they perform poorly as the network flows get more correlated*. The SLCorr scheme, however, is the optimum detection scheme for correlated network flows, *and* it also performs good enough in the case of independent network flows.

3.5 Implementation results

We implemented the watermarking scheme and tested it by using replayed SSH connections, using timings collected from real traces at the North Carolina State University, as well as at the University of Illinois. Our tests were carried out over the PlanetLab infrastructure.

In the first experiment, we watermarked SSH flows between two specific nodes for different values of watermark amplitude $(1, 3, 5, 7, 10, 20msec)$. We show the test statics for both true correlation (hypothesis one) and false correlation (hypothesis zero), along with their standard deviations in Figure 3.9 (each experiment is run for 20 times and the average jitter standard deviation over the link is about $\delta_b = 10msec$). As we expect from analysis, false detection metric has a mean of around zero, and a variance steadily constant (because n is fixed). For hypothesis one, the statistic mean increases linearly with watermark amplitude (recall that mean of true correlation is $\frac{a}{\sqrt{2}b_\delta}$), and variance shows not much change for different experiments.

In the second experiment we watermarked 100 SSH flows of length $N = 5000$ packets with fixed watermarked amplitude of $a = 10ms$ between two specific nodes (and also the same watermark bits). The high number of flows helps to measure the variance of metrics with more confidence. Figure 3.10a shows true detection metric, and false detection metric along with their standard deviation for different number of packets n. Mean of true correlation does not vary with n because watermark amplitude is fixed and network jitter does not vary that much; mean of false correlation is almost zero as we expect from analysis. Fortunately, false correlation shows a slightly smaller standard deviation which results in even fewer false positives. This

Table 3.7: False positive error rate of different detection schemes for network flows generated by browsing the same websites.

Website	N=25			N=50			N=100		
	SLCorr	ACTV	PASSV	SLCorr	ACTV	PASSV	SLCorr	ACTV	PASSV
baidu.com	0	0.08	0.29	0	0.12	0.07	0	0.12	0.08
blogger.com	0	0.56	0.97	0	0.89	0.63	0	0.34	1
facebook.com	0	0.95	0.91	0	0.9	0.97	0	0.59	0.96
live.com	0	0.81	1	0	0.33	1	0	0.08	0.38
wikipedia.org	0	0.44	0.94	0	0.44	0.44	0	0.39	0.46
yahoo.co.jp	0	0.08	0.66	0	0.03	0.33	0	0	0.05
yahoo.com	0	1	1	0	0.02	1	0	0	0.23
yandex.com	0	0.11	0.89	0	0.02	0.08	0	0	0.02

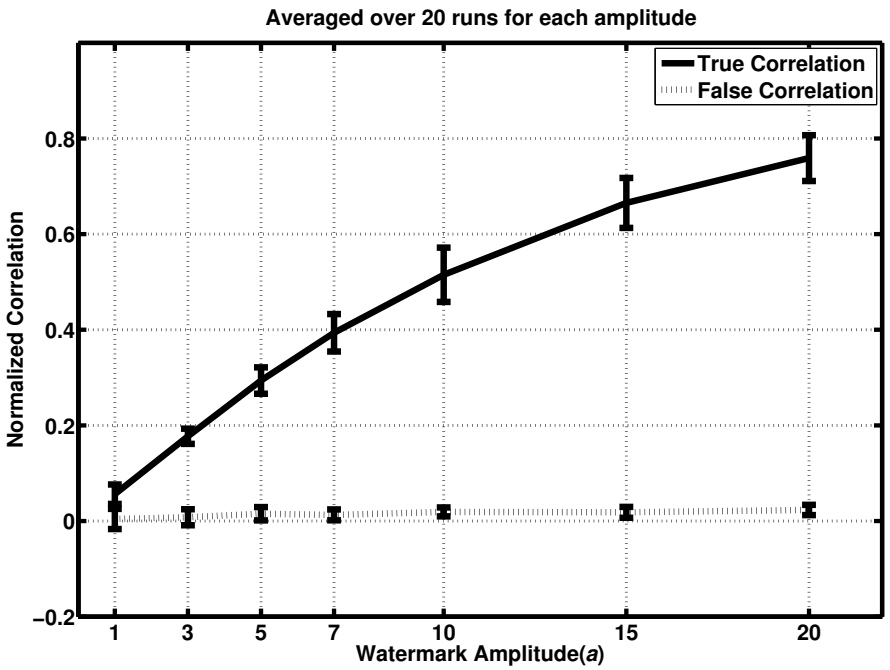
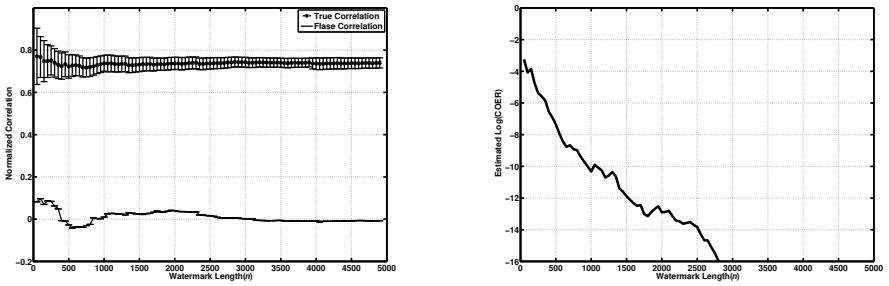


Figure 3.9: Normalized correlation test statistic for different watermark amplitudes.

is because we considered the worst case in analysis, i.e., equal rate unwatermarked flows. Figure 3.10b shows the COER estimated by fitting the errors rates to a Laplace distribution. The COER exponent varies linearly with n, as expected from the analysis. Based on this, we can achieve the tiny COER of 10^{-6} with fewer than 400 packets, which means that a typical SSH connection can be classified as a stepping stone or not within about 3 minutes. Similar passive and watermarking schemes require much more time to achieve similar error rates.

Comparing error rates of RAINBOW with those of previous passive schemes and blind watermarking schemes, RAINBOW outperforms them by orders of magnitude. The passive scheme of [6], which uses similar correlation mechanisms as RAINBOW, achieves false errors of 10^{-2} for different parameters. IPD-based watermarking scheme of [9] achieves false-negative rates of 10^{-2} and false-positive rates of at most 10^{-5}. These are far worse than what RAINBOW achieves. It should be mentioned that such a superior performance is expected from RAINBOW as its non-blind detector has access to important side information, i.e., original IPDs, that are not available to blind detectors.



(a) Normalized correlation metric (b) Estimated COER

Figure 3.10: Experimental detection performance for different watermark lengths.

3.5.1 Resource constraints

In this section we evaluate the required resources for RAINBOW in the case of stepping stone configurations. Of course, the resources required will be dependent on the number of low-rate connections, which in turn will depend on the size of the organization. We estimate the parameters needed to detect stepping stones in an organization such as the Coordinated Science Laboratory (CSL) at the University of Illinois at Urbana–Champaign. CSL has about 400 members, so we will assume as a worst-case that each member is performing a low-rate connection from the outside. Using a C++ implementation of RAINBOW, running on a 1.6 GHz Linux server with 1 GB of RAM, we can perform selective correlation (a more resource-intensive method that is robust to packet deletions and insertions, discussed in the next section) with 400 flows, using a watermark length of $n = 5000$, in 0.4μs. Table 3.8 lists the storage requirements for the IPD table for various choices of n.

Given the small size of the CPU and memory constraints, and the fact that they scale linearly with the number of flows, it is easy to see that much larger organizations can be supported using a commodity PC. For extremely large organizations, stepping stone detection can be partitioned among routers within sub-networks; e.g., in an organization such as the University of Illinois, each department can run its own stepping stone detection.

The choice of n presents a tradeoff between detection accuracy, watermark amplitude, and resource constraints. However, as we saw earlier, RAINBOW is effective with only a few hundred packets, whereas other passive schemes require many more packets [1, 3, 6–8], hence the resource constraints

Table 3.8: Maximum memory usage of the RAINBOW watermarking system for a medium-size network.

n parameter	Memory (MB)
50	0.15
100	0.3
200	0.6
500	1.5
1000	3.1

of RAINBOW will be significantly lower.

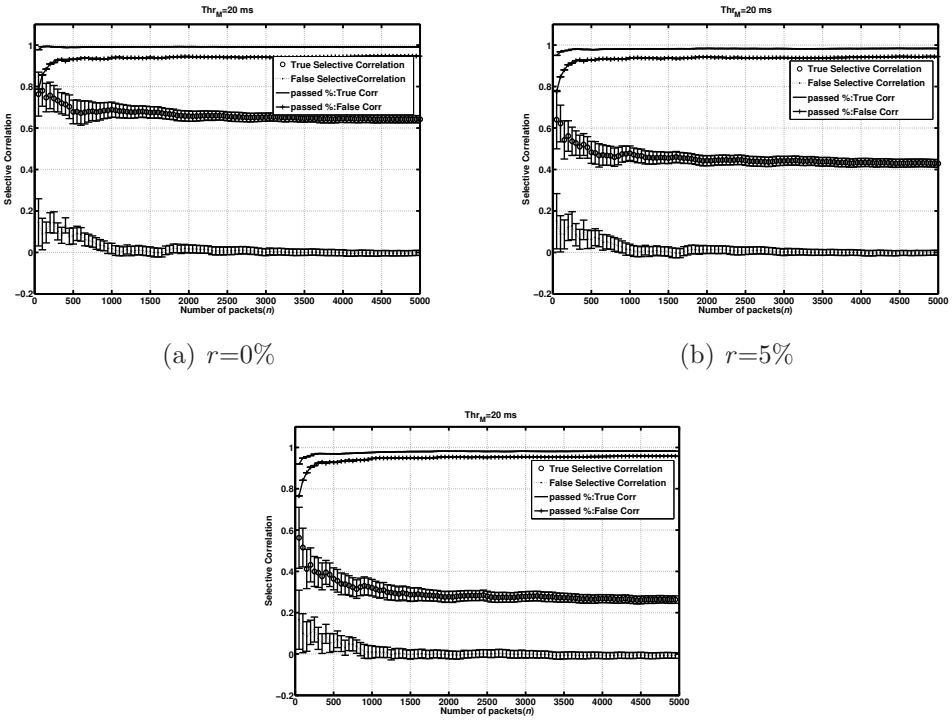
3.6 Selective correlation

In the previous sections we analyzed the performance of the detector based on normalized correlation. In our analysis and implementation, we assumed that there is a one-to-one relation between packets of watermarked flow and received flow; i.e., no packets are added to or removed from the flow between watermark insertion and watermark detection. This is often not the case, however, as real-world implementations introduce several causes for packets removal and insertion. For example, retransmissions at the TCP layer will introduce packets into one of the streams.[4] Applications may also repacketize flows while relaying them. Setup packets, such as TCP SYN/ACK packets and packets sent to initialize an SSH connection will also show up in only one of the two flows.

So, a practical watermark detector should be robust to packet addition and removal, i.e., work efficiently despite them. Among existing work, only recent schemes have considered repacketization and other natural perturbations [10, 14], while other work has looked at the presence of adversarial packet insertion and removal, or *chaff* [7, 8, 13]. Our normalized correlation scheme analyzed thus far is fragile to packet addition and removal, but with a modification we call *Selective Correlation* it shows promising performance dealing with packet addition and removal carried out at a relatively high rate.

Selective Correlation scheme: For selective correlation, we add a *match-*

[4]Though proper parsing of the TCP packets can be used to detect such retransmissions and remove them from consideration.



(a) r=0% (b) r=5%

(c) r=10%

Figure 3.11: Selective correlation performance for different ratio of add/drop packets (r).

ing step to the detector, which will pre-process $\tau^{\mathbf{u}}, \tau^{\mathbf{r}}$, and $\mathbf{w}$, before they are passed to the normalized correlation step. The aim of this step is to find and remove packets that do not have a corresponding match in the other flow.

The main idea is to use sliding windows to match IPD values of one flow by those of the other flow. For any IPD value of the received flow, τ_i^r, if the absolute difference from the corresponding IPD in database, τ_j^u, is smaller than η_M, packets are passed through as matched, along with the corresponding watermark bit. If not, the matching block tries to find an IPD in a $[j-L, j+L]$ window of $\tau^{\mathbf{u}}$ with the smallest IPD difference from τ_i^r that is also smaller than η_M. If no match is found the packet is dropped. L is the maximum expected change in number of packets and η_M depends on jitter variance.

To account for the case where too many packets are not matched, the detector also monitors the percentage of matched packets and declares the received flow as unwatermarked if this number is smaller than a threshold η_R.

Implementation results: We implemented selective correlation scheme over the same watermarked connections in PlanetLab, after adding and removing different percentages of packets to the flows. We set η_M to be twice the average standard deviation of jitter; i.e., $\eta_M = 20ms$, and L twice the maximum number of packets expected to be added or removed. Figure 3.11 illustrates true selective correlation and false selective correlation along with the percentage of packets matched in each case. If the percentage of matched packets falls below some reasonable η_R, the detector decides that the flow is not watermarked.

For the case that we do not have packet count changes (Figure 3.11a), selective correlation outperforms the simple correlation scheme (Figure 3.10a). This is because selective correlation removes IPDs with high jitter added. As fraction of packets added and/or removed increases, mean of true selective correlation decreases as shown in Figures 3.11b and 3.11c. This leads detection performance to decrease, but even for 20 percent of packets changed (10% added and 10% removed), detection can be performed efficiently.

3.7 Watermark invisibility

An efficient network flow watermarking scheme needs to be invisible to prevent the watermark from being detected and possibly removed by an active attacker. This also prevents the watermark from interfering with normal users traffic. Because of embedding large amplitude watermarks, previous flow watermarking schemes are not invisible; several interval-based watermarking schemes [10, 13, 14] have shown to be subject to detection and removal [31] (it should be mentioned that changing some watermarking parameters, e.g. interval length, in these schemes from the original values in the corresponfing papers improves invisibility, but drastically ruins false detection errors which make the schemes practically useless). Peng et al. [29] show how the *Kolmogrov–Smirnov test* (K–S test) is efficient in detecting large amplitude *QIM* watermarks applied to inter-packet delays. We use the Kolmogorov–Smirnov test to discuss invisibility of RAINBOW flow watermarking scheme.

The K-S test is used to determine whether two samples from a sequence of two observations (or one observation and samples drawn from a references

probability distribution function) belong to the same distribution by measuring the maximum distance between empirical distribution functions (or the empirical distribution function and the reference distribution function). In case of a given reference distribution function $F(x)$, the value of the K–S test is:

$$\sup_x |F_n(x) - F(x)|,$$

where $F_n(x)$ denotes the empirical distribution function from a sample of n observations.

In the first experiment we ran the K–S test against the non-watermarked and watermarked version of a SSH flow, transmitted in the same network (with similar network delay). The average K–S distance between them (averaged over 10 connections) is 0.0082 which results in a 98% confidence in declaring them to be from the same distribution. In other words, watermark presence on the flows would be transparent to normal users and (limited) attackers.

In the second experiment we considered a more intelligent/powerful attacker who sends a flow to the watermarker and receives it on another compromised host. Since the attacker has the original flow, he only needs to discriminate between $w + \delta_i$ and δ_j , where δ_i and δ_j are different jitters (measured over PlanetLab). We compared two scenarios: K–S test between $w + \delta_1$ and δ_2 and K–S test between δ_3 and δ_4. Figure 3.12 shows the difference between K–S statistics in the two different scenarios for different values of γ. As γ decreases, the attacker loses his chance to distinguish between watermarked and unwatermarked flows. Comparing with results of Section 3.3, we see that there is a tradeoff between different watermarking attributes, i.e., invisibility and robustness. A similar K–S experiment on other flow watermarking schemes returns much higher differences, which makes them suspect to attacks [31].

Gianvecchio et al. use information theory tools to invent new metrics for efficient detection of covert timing channels [50]. We use their entropy-based tools, *EN* and *CCE* tests, on a number of watermarked SSH flows (each 5000 packets) and their corresponding unwatermarked (but jittered) flows. Table 3.9 shows the averaged test metrics for regular (unwatermarked) and watermarked SSH flows. As results show, even for large values of γ, watermarking does not change *EN* and *CCE* test results significantly (the

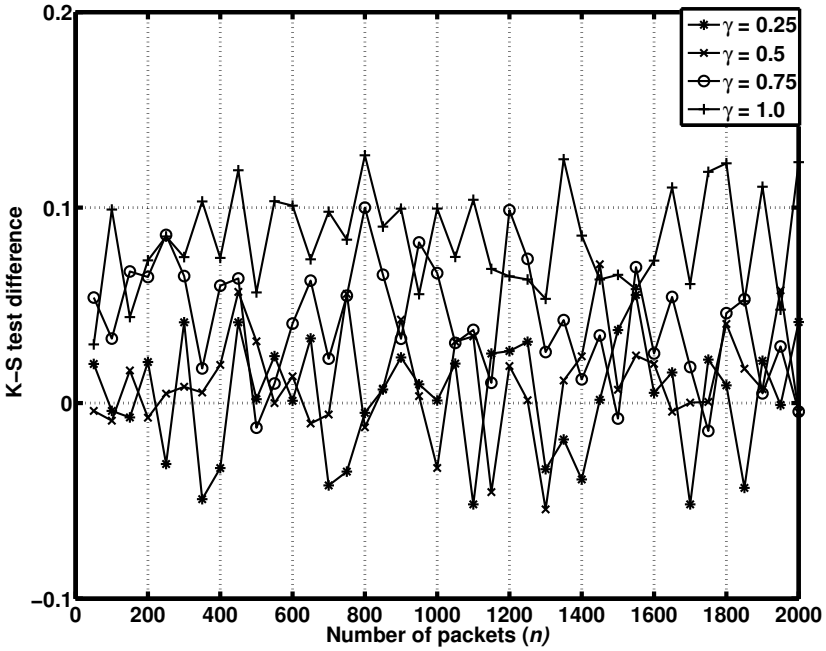


Figure 3.12: Kolmogorov–Smirnov test difference

Table 3.9: Entropy test results to evaluate invisibility of RAINBOW.

γ	EN test		CCE test	
	Regular	Watermarked	Regular	Watermarked
0.25	13.7191	13.7499	2.2475	2.2514
0.50	13.8061	13.7661	2.2476	2.2493
0.75	13.7651	13.7590	2.2471	2.2526
1.00	13.7711	13.7903	2.2484	2.2498
2.00	13.7545	13.6533	2.2496	2.2498

decision thresholds for *EN* and *CCE* tests are 21.20 and 2.17, respectively). This shows that RAINBOW remains invisible in the face of these information-theoretical tools.

3.8 Conclusions

We proposed a novel non-blind network flow watermarking scheme called RAINBOW, for linking flows. RAINBOW combines some of the advantages of passive traffic analysis with watermarking schemes. Like passive traffic analysis, RAINBOW does not interfere with regular users by inserting large

delays that are used in existing watermarking schemes; in fact, we show that RAINBOW is *invisible* to detection by an attacker. Like other watermarking schemes, RAINBOW achieves very low false error rates. In fact, we show, both through analysis and by means of experiment, that the false error rates of RAINBOW are orders of magnitude lower for short observation periods than existing passive and active schemes. RAINBOW can also be made robust to high rates of packet addition and removal by introducing selective correlation, at the cost of somewhat increased observation period lengths. In our future work, we intend to explore coding tools to increase the efficiency of RAINBOW and explore the possibility of a blind or semi-blind watermark scheme that remains invisible.

CHAPTER 4

EFFICIENT BLIND WATERMARKING

Network flow watermarks have been proposed as an *active* alternative to perform traffic analysis more efficiently. Watermarks are more *scalable* (than the traditional passive schemes [1–3, 6–8, 24, 25, 27]), as they require asymptotically less communication and computation; they also can operate on shorter flows and provide lower error rates than passive analysis. Previous watermark designs can be divided into two main categories: packet-based watermarks that operate on individual delays between packets [9, 54], and interval-based watermarks that perform an operation on an entire interval [10, 13, 14]. The former category is not robust to packet losses, reorderings, and insertions; the latter are subject to a multi-flow attack [31] that can recover and remove the watermark. In addition, these watermarks introduce large delays, making them not suitable for practical applications. Ezzeddine and Moulin [54] study timing stegocodes using queue-based encoders [55] and Shannon's encoding functions [56] for channels with causal side information at the transmitter. More specifically, the authors model the communication network as a noiseless channel with discrete-valued inter-packet delays and use a stegocoder that perturbs inter-packet delays in order to embed stego messages. This approach provides theoretical invisibility for the stego traffic as the utilized encoders do not change the probabilistic distribution of inter-packet delays. The authors also derive the achievable rates for the designed stegocodes and compute the maximum achievable rates for certain network parameters. In Chapter 3 we introduced RAINBOW, a packet-based watermark that provides strong invisibility by using a non-blind approach for watermark insertion. RAINBOW [11] is a packet-based watermark that *is* robust to lost or reordered packets; however, RAINBOW takes a non-blind detection approach to achieve high detection accuracy over short flows while using very small delays. Our goal is to build a blind and therefore scalable watermark, at the cost of potentially requiring longer watermarked flows.

Table 4.1: Watermark scheme comparison

Scheme	Invisible?	Robust to losses?	Resilient to MFA?	Scalable?
IPD-based watermark [9]	no	no	yes	yes
Interval-based watermarks [10, 13, 14]	no	yes	no	yes
RAINBOW [11]	yes	yes	yes	no
SWIRL	**yes**	**yes**	**yes**	**yes**

We introduce SWIRL, a Scalable Watermark that is Invisible and Resilient to packet Losses. SWIRL is an interval-based watermark, but it uses a novel approach to resist multi-flow attacks. The watermark pattern is chosen based on the characteristics of the flow being marked; as a result, each flow is marked with a different pattern. SWIRL watermarks introduce small delays to the network flows, and thus are practical to deploy in real-world scenarios. The small amount of distortion also makes SWIRL invisible to state-of-the-art information-theoretic tools for covert channel detection [50]. Table 4.1 summarizes the properties of previous work.

We perform a mathematical analysis of the error rates of SWIRL, showing that it can achieve very low false error rates on short flows. We validate our analysis against simulations and an implementation running on the PlanetLab testbed [49]. We show experimentally that SWIRL is resistant to the multi-flow attacks.

SWIRL provides the first practical approach to large scale traffic analysis; it therefore extends the reach of traffic analysis attacks in both anonymous systems and network attack attribution. We also consider a novel application of watermarks to defend against a congestion attack in the Tor anonymizing network [21]. We show that a watermark, normally a privacy-invasive tool used to link anonymous flows, can actually help protect users' privacy by preventing attackers from creating routing loops. The properties of SWIRL provide a practical defense where previous traffic analysis approaches would not be appropriate.

The rest of this chapter is organized as follows. In Section 4.1 we describe the design of the SWIRL watermarking scheme. We analyze SWIRL by modeling the network flow behavior in Section 4.2 to provide false errors analysis of the scheme. We evaluate SWIRL with simulations as well as implemen-

tation in Section 4.3. We discuss watermark invisibility and resilience to multi-flow attacks in Section 4.4. Finally, we conclude in Section 4.5. Also, in Section 6.2 we provide a novel application of SWIRL to Tor congestion attacks.

4.1 SWIRL watermarking scheme

SWIRL is an interval-based watermark; therefore, it considers the flow as a collection of intervals of length T, with an initial offset o; i.e., the ith interval includes packets during time period $[o + iT, o + (i + 1)T)$. We first describe our approach to flow-dependent marking and then describe the overall SWIRL scheme.

4.1.1 Flow-dependent marking

To perform flow-dependent marking, we select two intervals: a *base* and a *mark* interval. The positions of these intervals will be fixed for all flows, but is otherwise arbitrary, with the restriction that the base interval must come earlier. During watermarking, we will use the base interval to decide which pattern to insert on the mark interval; the detector will correspondingly look for the pattern it computes using its version of the base interval.

The property of the base interval that we use is the interval *centroid*, which is the average distance of the packets from the start of the interval. I.e., if the interval i contains packets arriving at times $t_1, \ldots, t_n$, the centroid is:

$$C = \frac{1}{n} \sum_{j=1}^{n} (t_j - (o + iT)) \tag{4.1}$$

To decide on the pattern to be used on the mark interval, we quantize the centroid to a symbol s in the range $[0, m)$ for some $m \in \mathbb{Z}_+$. Since the range of the centroid is $[0, T)$, a simple approach would be to set $s = \lfloor mC/T \rfloor$. However, this would result in a non-uniform distribution for s, since a centroid is more likely to be in the center than at the interval. The actual distribution of centroids is heavily dependent on the rate of the flow as well as the distribution of packet delays. In order to approximate a uniform

distribution for s, we take the approach of using finer partitioning. Namely, we set:

$$s = \lfloor qmC/T \rfloor \bmod m \tag{4.2}$$

for $q > 1$. The quantization multiplier q helps smooth out the distribution: a larger value of q makes the value of s more sensitive to the value of C, i.e., a smaller change in C is needed for s to change its value. This effect is also shown in the invisibility analysis of Section 4.4.3. To see this effect better, note that $C \in [0\ T]$, based on the definition of C in (4.1). The way (4.2) works is that for any q, it divides the rage of C (i.e., $[0\ T]$) into $\lceil qm \rceil$ subsequent, non-overlapping subintervals, with equal length $\lfloor T/(mq) \rfloor$ (except for the very last subinterval. Let us give a number in $\{1, .., \lceil qm \rceil\}$ to each subintervals based on its order of appearance. The value of $s = k$ is returned by (4.2) if C appears in any of the intervals with numbers αk, where $\alpha \in \mathbb{N}$. We can see that for $q = 1$ there is only a large subinterval that results in $s = k$; however, for larger q, the values of $s = k$ result from a number of smaller subintervals in the range $[0\ T]$. As a result, for larger q the value of each symbol comes from different parts of C's range, which smoothes out the distribution of s.

The value s is then used to transform the mark interval. We first subdivide the mark interval into r subintervals of length T/r each. The subintervals are then further subdivided into m slots each, with the slots numbered $0, \ldots, m-1$, see Figure 4.1. We select a slot in each subinterval i by applying a permutation $pi^{(i)}$ to s; each packet is then delayed such that it falls within a selected slot, possibly moving into the next subinterval. (Any packets at the end of the interval past the last selected slot are not delayed.) This produces a distinct pattern in the mark interval; see Figure 4.2 for an illustration. Note that we must use distinct permutation for each subinterval; otherwise we risk creating a periodic pattern that can easily be observed. The permutations $\pi^{(0)}, \ldots, \pi^{(r-1)}$ are part of the secret watermark key.

4.1.2 Detection

To detect the watermark presence, the detector analyzes packets in the base interval to compute the centroid $\hat{C}$. It then derives $\hat{s}$ from $\hat{C}$ using (4.2). It then counts the fraction of packets in the mark interval that are in the

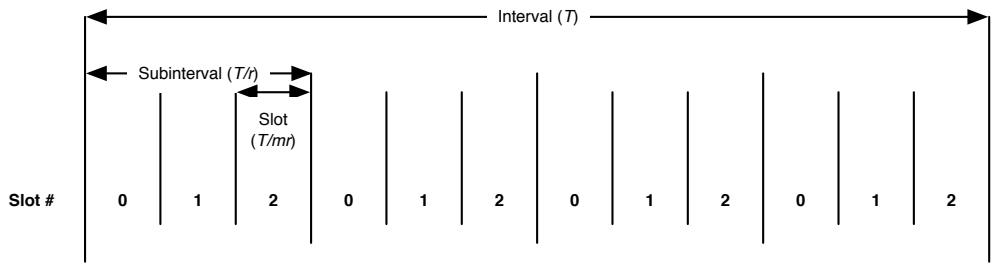
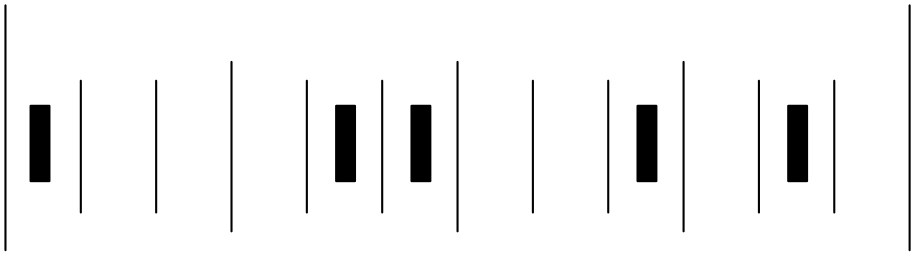
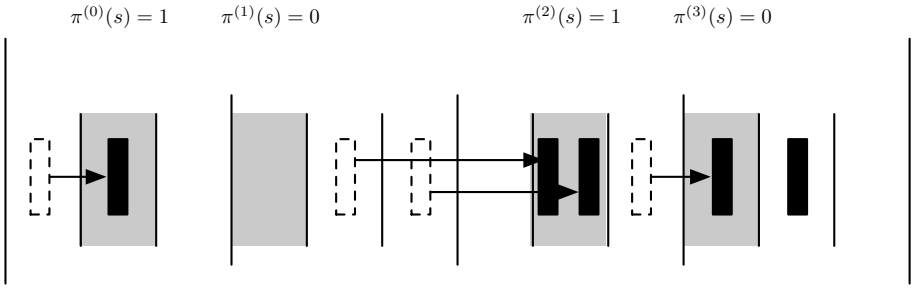


Figure 4.1: Slot numbering ($m = 3, r = 4$)



(a) Original flow



(b) Watermarked flow

Figure 4.2: Delaying packets to insert a watermark ($m = 3, r = 4$).

correct slot ($\pi^{(i)}(\hat{s})$). If this ratio, ρ, is greater than a packet threshold τ, then the watermark is considered to be detected successfully.

Note that the centroid of the interval may have shifted due to network noise. We therefore consider an alternate quantization of it, $\hat{s}'$, to be the next nearest quantization to $\hat{C}$:

$$\hat{s}' = \begin{cases} \lceil mq\hat{C}/T \rceil \bmod m & \text{if } \{mq\hat{C}/T\} \geq 0.5 \\ \lfloor mq\hat{C}/T \rfloor - 1 \bmod m & \text{otherwise} \end{cases} \tag{4.3}$$

Table 4.2: Watermark parameters

System parameters	
r	Number of subintervals
m	Number of slots per subinterval
τ	Packet detection threshold
η	Mark detection threshold
n	Number of base and mark intervals
T	Interval length
q	Quantization multiplier
Secret Parameters	
o	Initial offset
b_j	Location of base intervals, $j = 0, \ldots, n-1$
m_j	Location of mark intervals, $j = 0, \ldots, n-1$
$\pi_j^{(i)} \in S_m$	Permutations for each mark subinterval, $j = 0, \ldots, n-1, i = 0, \ldots, r$

where $\{x\} = x - \lfloor x \rfloor$ denotes the fractional part of x. We then repeat the detection using $\hat{s}'$ to compute ρ'. If $\rho' > \tau$, the watermark is also considered to be detected.

4.1.3 SWIRL design

A single watermark instance is likely to produce too high a rate of false errors. To improve detection, SWIRL uses n base and mark interval pairs. Let d be the number of intervals where the watermark was detected; then the entire watermark is considered to be detected if $d > \eta$ for some threshold η. The full list of parameters for SWIRL is summarized in Table 4.2. These parameters must be shared between the watermarker and detector. The system parameters are chosen to achieve a particular performance based on the properties of the original flows and the noisy channel. The secret parameters are chosen randomly and can be thought of as a secret key shared between the watermarker and the detector. In Section 4.4.1 we analyze the security of the SWIRL watermark by deriving the entropy of its watermark key.

4.2 System analysis

4.2.1 False-positive errors

The false-positive error rate is the probability of detecting a non-watermarked flow as watermarked. To calculate this, first consider the probability that a single packet in some mark interval is in the "correct" slot. If we assume a Poisson distribution for the flows, it is easy to see that:

$$FP_p = \frac{1}{m} \tag{4.4}$$

Of course, actual flows might have different distributions; however, unless the traffic patterns in a flow are correlated with the distances between slots (randomized by $\pi_j^{(i)}$), this will remain a good approximation.

Given a mark interval with P packets, the number of packets in "correct" slots will follow a Binomial distribution of P trials with probability of success PF_p, $B(P, FP_p)$. The cumulative distribution function of a Binomial distribution with v trials and success probability h, $B(v, h)$ is given by the regularized incomplete beta function (which is the cumulative distribution function of the Beta distribution [57]), $I(\cdot)$, as:

$$P(X \leq k) = I_{1-h}(v - k, k + 1) \tag{4.5}$$

It follows that the odds of getting at least τ fraction of packets in the correct slots can be computed as:

$$I_{1/m}(\lceil \tau P \rceil, 1 + P - \lceil \tau P \rceil) \tag{4.6}$$

Since we perform the detection for both $\hat{s}$ and $\hat{s}'$, the probability of an interval with P packets being considered detected is:

$$FP_I^P \leq 2I_{1/m}(\lceil \tau P \rceil, 1 + P - \lceil \tau P \rceil) \tag{4.7}$$

Modeling the flow as a Poisson process of rate λ, the number of packets in an interval of length T is distributed according to a Poisson distribution with parameter λT. Therefore, the overall probability of a false positive detection

in an interval is:

$$FP_I = E_P^{\lambda T}[FP_I^P] = \sum_{P=1}^{\infty} \frac{e^{-\lambda T}(\lambda T)^P}{P!} FP_I^P \tag{4.8}$$

where $E_P^{\lambda T}$ computes the expected value with respect to P according to a Poisson distribution with parameter λT. Finally, the total number of detected intervals will once again follow a Binomial distribution $B(n, FP_I)$, thus the overall false-positive rate is:

$$FP \leq I_{FP_I}(\eta, 1 + n - \eta) \tag{4.9}$$

4.2.2 False-negative errors

Now we consider the false-negative errors, i.e., the probability that a watermarked flow is considered not to be watermarked by the detector.

Again, we start by considering a single mark interval. The probability that it is considered not detected is:

$$FN_I \leq FN_s + (1 - FN_s)FN_p r \tag{4.10}$$

where FN_s represents the probability that neither $\hat{s}$ nor $\hat{s}'$ correspond to the original quantization s, and FN_p represents the probability that more than (1-τ) fraction of packets have shifted out of the "correct" slot, s.

Note that for the quantization to be misdetected, the centroid must shift by at least $T/(2mq)$; thus:

$$FN_s \leq P\left(\left|\hat{C} - C\right| > \frac{T}{2mq}\right) \tag{4.11}$$

Note that, given Q packets in the base interval, with delay of δ_j for packet j, we can calculate:

$$\hat{C} - C = \frac{1}{Q}\sum_{j=1}^{Q} \delta_j \tag{4.12}$$

We adopt a Gaussian approximation for the distribution of packet delays, as suggested in previous work [29]. In fact, the rationale behind using this model is the following: the delay applied to each packet between the wa-

termarker and the detector is the accumulation of delays applied to that packet by several network entities (e.g., routers, firewalls). This allows us to model packet delays with a Gaussian distribution based on the Central Limit Theorem [48], which states that the summation a large number of random variables will have an approximately normal distribution. Using synchronization described in Section 4.3.4, we can ensure that the distribution has mean 0. We thus model delay as i.i.d. Gaussian: $\delta_j \sim N(0, \sigma^2)$. Then:

$$\hat{C} - C \sim N(0, \sigma^2/Q) \tag{4.13}$$

$$\begin{aligned} FN_s &\leq P\left(|\hat{C} - C| > \frac{T}{2mq}\right) \\ &= E_Q^{\lambda T}\left[2\left(1 - \Phi_{0,1}\left(\frac{T\sqrt{Q}}{2mq \cdot \sigma}\right)\right)\right] \end{aligned} \tag{4.14}$$

where $\Phi_{0,1}(\cdot)$ is the CDF of $N(0,1)$, and $E_Q^{\lambda T}$ averages with respect to the Poisson distributed variable Q. Note that the Central Limit Theorem is not suitable for approximating the tail of the summation distribution, however in our analysis we have used the middle part of the approximated distribution: in (4.14) $\frac{T}{2mq}$ is not far away from the center of the distribution. More specifically, for the parameters used in this paper (Table 4.3) $\frac{T}{2mq}$ is between 2 to 4 times the standard deviation of network delay, depending on the network conditions. Also, note that decreasing $\frac{T}{2mq}$ (i.e., decreasing T or increasing m and q) makes the approximation more accurate.

To compute FN_p, we first need to consider the probability that each individual packet would have shifted out of the assigned slot. Suppose that the packet p_j was distance x from the center of the slot ($-T/(2rm) \leq x \leq T/(2rm)$). Given the Gaussian distribution of δ_j, the probability of the shift is:

$$P(p_j \text{ shifted}|x) = 1 - \Phi_{0,1}\left(\frac{(T/2rm) - x}{\sigma}\right) + \Phi_{0,1}\left(-\frac{(T/2rm) + x}{\sigma}\right) \tag{4.15}$$

Given that x will have a uniform distribution within the slot, we can integrate to find:

$$FN_{p_j} = \frac{rm}{T}\int_{x=-T/(2rm)}^{T/(2rm)} P(p_j \text{ shifted}|x)dx \tag{4.16}$$

The number of packets that are misdetected, out of P packets in the mark interval, is given by the Binomial distribution $B(P, FN_{p_j})$. Correspondingly:

$$FN_p = E_P^{\lambda T}\left[I_{FP_{p_j}}(1 + P - \lceil \tau P \rceil, \lceil \tau P \rceil)\right] \tag{4.17}$$

Using equations (4.14) and (4.17), we can compute FN_I in (4.10) and correspondingly:

$$FN = I_{FN_I}(n - \eta + 1, \eta) \tag{4.18}$$

4.3 Evaluation

We evaluate SWIRL watermarking scheme for the application of stepping stone detection. Our evaluation is also valid for Tor congestion attack prevention application, discussed in Section 6.2. For the application of linking flows in anonymous networks, a new set of parameters would need to be derived following the methodology described in this section.

4.3.1 Parameter choices

Table 4.3 shows the tradeoffs that result from choosing different parameters of the watermarking scheme, along with the chosen values for our implementation. The choice of q represents a tradeoff; on one hand, larger q increases the false-negative rate by increasing FN_s of (4.14). On the other hand, smaller q may result in an uneven distribution of s, resulting in a multi-flow attack (MFA). We will defer a full examination of this tradeoff until our MFA analysis in Section 4.4.3; for the subsequent simulations and experiments, we set $q = 2.5$.

Likewise, r represents a tradeoff between false negatives and the amount of delay. The maximum inserted delay is bounded by $T/r(2 - 2/m)$. We experiment with different choices of r in the design.

In our experiments, we pick $n = 32$, i.e., 32 base and mark intervals are selected. This means that the watermark sequence must be at least $64T$ long; however, this ensures a low overall rate of errors.

The parameter T should be chosen based on the rate of the flow, since the false positive rate is proportional to $T\lambda$. In our experiments, we use flows

Table 4.3: Tradeoffs in selecting watermark system parameters

Param.	Tradeoffs		Selected value
	Increasing improves:	Decreasing improves:	
r	Delay, invisibility	False-negative errors	20
m	False positives	False negatives	5
τ	False positives	False negatives	0.5
η	False positives	False negatives	12
n	Detection performance (FP,FN)	Detection time	32
T	False-positive errors	Detection time	2 sec
q	MFA invisibility	False-negative errors	2.5

that have a rate of 4–7 packets per second (pps), thus we set T to be 2s. For flows with rates lower than 3pps, we suggest doubling the T parameter to compensate for the smaller number of packets in each interval.

Both τ and η can be used to control the rates of false positive and false negative errors. For a fixed η, increasing the τ threshold improves the false positives while it worsens the false negatives. Likewise, having the η threshold fixed increasing the τ threshold improves the false positives and worsens the false negatives.

Note that given a choice of η, it is possible to find the value of τ that results in an equal rate of false-positive and false-negative errors; this occurs at $\tau = 0.5$ for our parameter setting. The corresponding error rate is called the *cross-over error rate* (COER); in this case, it is approximately 10^{-7}. We can use this to optimize the joint choice of τ and η by computing the COER that can be achieved at any given choice of η. Figure 4.3 shows this for flows with average λ of 4.4pps. As can be seen, $\eta = 12$ minimizes the COER while the corresponding value of τ where the COER is achieved is approximately 0.5. Note that some applications will benefit from a different optimization target; e.g., lowest false-negative rate given a false-positive rate of at least 10^{-6}. In this case, the analytical[1] false error rates can be used to find the optimal values of η and τ.

We also compute the η threshold that achieves the minimal COER for different flow rates. Figure 4.4a shows the results; the corresponding COER is shown in Figure 4.4b.[2] This shows that, for optimal detection, η should

[1]By "analytical" we mean the estimates obtained from the analysis presented in Section 4.2.

[2]The non-monotonic behavior in the graph corresponds to changing the value of T for flows below 3pps.

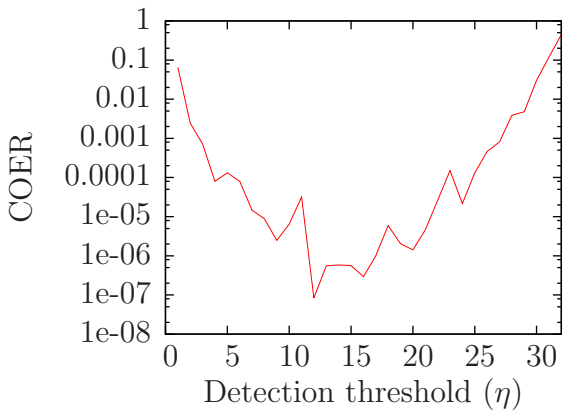
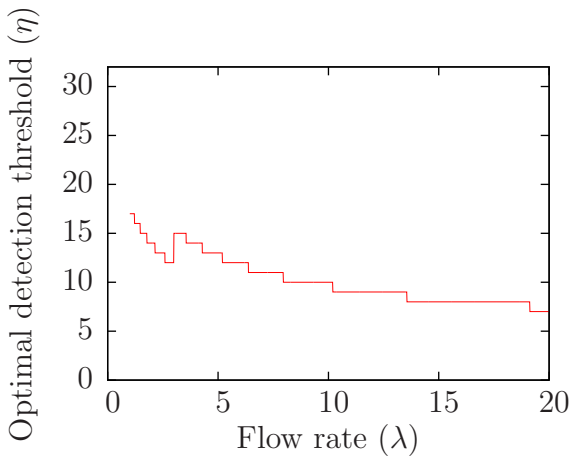


Figure 4.3: Cross-Over Error Rate (COER) for different detection thresholds η ($\lambda = 4.4pps$).

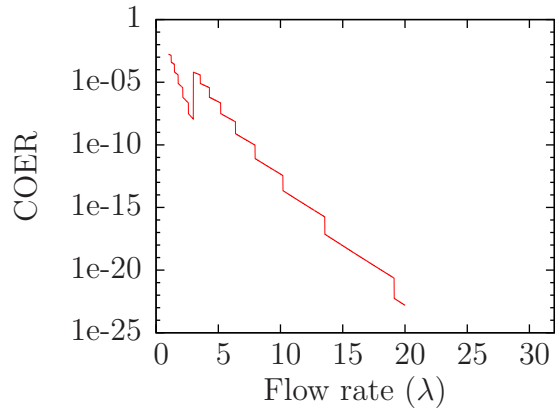
be chosen based on the flow rate. The analytical computations, however, are based on approximate models of traffic and delays, and compute upper bounds of error rates. For simplicity of implementation, one might choose to use a single detection threshold regardless of the flow rate. Figure 4.5 shows the false positive and false negative error rates for a detector using a fixed $\eta = 12$, with the corresponding optimal η COER rates shown for comparison. As can be seen, for flows with smaller rates the fixed detection threshold improves the false negative errors rates at the price of increasing the false positive errors; this is opposite for the higher rate flows, but offers reasonable error performance overall.

4.3.2 Simulations

We simulated the SWIRL watermarking system in Matlab. A watermark key is generated using the random number generators. We use $n = 32$, and use the system design parameters described in the previous section (see Table 4.3). We use traces collected by the CAIDA project from its `equinix-chicago` monitor—an OC192 link of a Tier 1 ISP—in January 2009 [58]. We selected SSH (port 22) flows out of the traces, since SSH is frequently used with interactive stepping stones; we used flows that were at least $2nT = 128\,\text{s}$ long, for a total of 304 flows. In every run of the simulation, an SSH flow is randomly selected from the database and is watermarked using the designated watermarking key. Since the analysis in Section 4.2 pre-



(a) Optimal detection threshold



(b) COER

Figure 4.4: COER and optimal detection threshold for different effective rates.

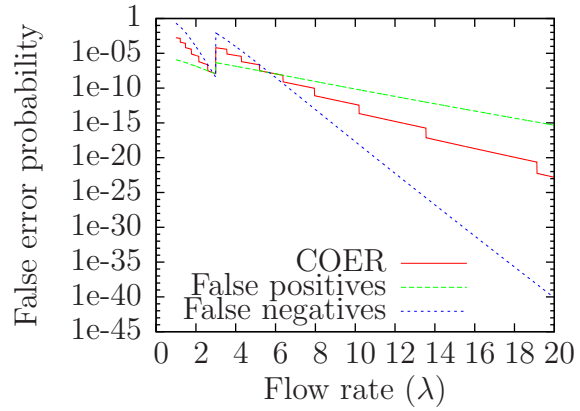


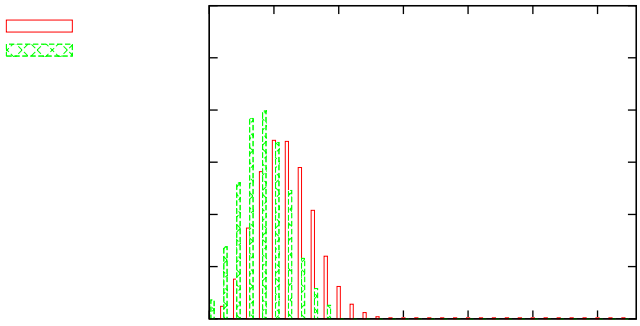
Figure 4.5: Probability of false positive and false negative errors for the optimal threshold (COER) and for a constant detection threshold of $\eta = 12$.

dicts that error rates are dependent on the rate of the flow, we chose flows that have similar rates for our simulations ($9\,\text{pps} < \lambda < 10\,\text{pps}$).

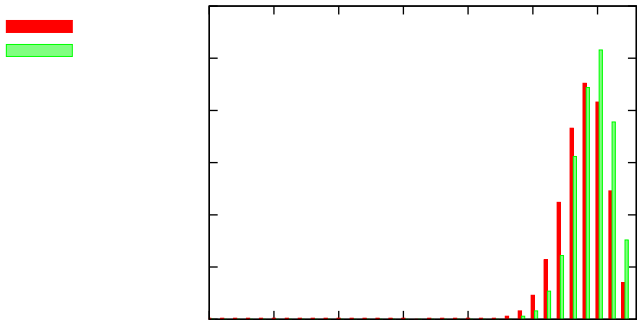
To simulate the effect of network delays, we captured traces of round-trip delays between pairs of randomly chosen PlanetLab nodes [49]; each trace captures the jitter properties of the relevant Internet path.[3] The traces have standard deviations ranging from $\sigma = 6.2\,\text{ms}$ to $\sigma = 12\,\text{ms}$. For every run of the simulations a network delay sequence is selected at random and applied to the watermarked flow. Finally, the watermarked flow affected by network delay is evaluated by the simulated watermark detector to check for the shared watermark. We run this experiment 1000 times, each time with the same watermark key but random selection of network flows and network delays. Figure 4.6b shows the histogram of the number of watermark intervals (out of $n = 32$) that the detector successfully detects by evaluating a watermarked flow, namely *true detected intervals*. We compare this to the expected errors as predicted by the analysis in Section 4.2. The simulations show better than predicted error behavior due to the use of upper bounds in the analysis.

To consider false-positive errors, we perform the same simulations to evaluate the number of watermark intervals detected when the detector inspects non-watermarked flows. Similar to the previous experiment, we randomly select network flows from the database and apply a network delay trace to the selected flows using the same scenario. We then pass the flows through the watermark detector to check for the watermarked intervals. This experiment is also run for 1000 times. Figure 4.6a illustrates the experimental and analytical histograms of *false detected intervals*, namely, the number of watermark intervals detected by the SWIRL detector from non-watermarked flows. Again, the simulations result in fewer errors than predicted by the upper bounds in the analysis. Comparing the two figures, it is easy to see that there is a strong separation between the two distributions; thus we should be able to achieve a low false error rate by choosing the detection threshold appropriately. Using a threshold of $\eta = 12$, we observed no false-positive or false-negative errors in our simulations.

[3] We approximate the one-way jitter by the round-trip properties.



(a) False detected intervals



(b) True detected intervals

Figure 4.6: Histogram of watermark intervals detected by the simulated SWIRL detector for $\frac{T}{r} = 100$ (1000 random runs), as well as expected histogram values from the analysis.

4.3.3 Implementation

We implemented the SWIRL watermarking scheme over the PlanetLab infrastructure to evaluate its performance. We used the same data set of SSH flows from the CAIDA traces, but we explored a wide range of flow rates. In each experiment, the watermarker reads the timings of packets in a flow read from the trace and then applies the watermark to them to generate a sequence of packets. These packets are then sent over the wide area to another PlanetLab node running the detector. Both the watermarker and detector are written in C++. As in simulations, we use the system parameters from Table 4.3. We also generate the watermarking key randomly as described before. To obtain false-positive rates, we fed flow timings from traces directly

Table 4.4: Watermark detection results for the PlanetLab experiments.

Group	Flow rate λ range (packet/sec)	r	T (sec)	True detected intervals		False detected intervals	
				Mean	Range	Mean	Range
A	6–10	10	2	31.6	30–32	3.5	0–6
		20	2	30.56	28–32	2.6	0–5
		30	2	31.4	29–32	2.8	2–4
B	3–6	10	2	30.89	29–32	2.87	1–5
		20	2	31.25	27–32	2.87	2–4
		30	2	29.4	24-32	2.87	1–4
C	0–3	10	4	25.25	15–31	1.4	0–2
		20	4	22.75	14–30	1.4	0–2
		30	4	20.66	14-27	1.1	0–3

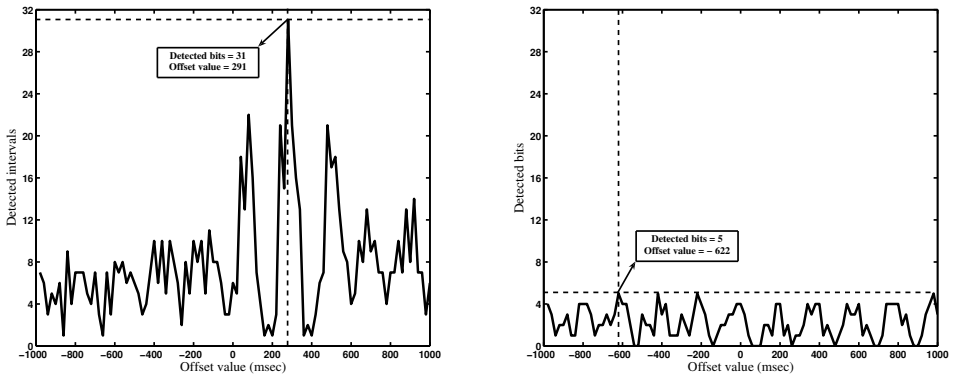
into the detector.

Table 4.4 summarizes the PlanetLab experiment results. Since the analysis suggests different detection performance for different flow rates, the flows are selected such that their rates lie in one of the three ranges shown in Table 4.4; we used about 100 flows per group. Since the detection performance drastically improves with flow rate we skip data rates higher than 10 pps in our experiments. Also, in order to illustrate the effect of r parameter on the system performance each group of flows are watermarked with three different values of r. As the results show, in all cases a choice of $\eta = 12$ results in zero errors.

We notice that detection performance improves for higher rate flows, e.g., group A results in the best detection results. For a given group of flows, increasing r degrades detection performance, but improves watermark delay and invisibility as discussed in Section 4.4. Note that, as mentioned before, for the lower-rate flows, SWIRL detector uses a higher value for the T parameter, in order to compensate for the smaller number of packets in each interval. One can show that increasing T significantly improves the detection performance at the expense of longer watermark detection times.

4.3.4 Detector synchronization

As shown in Table 4.2, the offset o is shared between the watermarker and the detector. In fact, this is not necessary, as the scheme is self-synchronizing:



(a) Watermarked flow (b) Non-watermarked flow

Figure 4.7: Synchronization at watermark detection.

the detector can perform detection using multiple offset value and return the best result. For example, Figure 4.7 shows a detector trying offset values in the range $[0, T]$ using steps of $T/100$. This approach allows the detector to use a randomized offset; this can serve as an additional countermeasure for the multi-flow attack, as discussed in [31]. It also ensures that two flows that exhibit similar behavior (e.g., repeated downloads of the same web page) will nevertheless be marked with different patterns.

4.4 Watermark invisibility

In this section, we start by showing that the very high entropy of the SWIRL watermark key makes is infeasible for an attacker to guess the watermark key. This is important to show, since an attacker who has access to the watermark key (e.g., through guessing the key) can easily verify if a flow has been watermarked. Then, we show that without having access to the watermark key an attacker is unable to detect the SWIRL watermark from a single watermark flow, as well as from multiple watermarked flows.

4.4.1 Watermark key entropy

To maintain invisibility, the watermark key must remain secret. We therefore estimate the size of the key space for the secret parameters used in SWIRL,

```
1 intervals = range(2*n) # 0,...,2n-1
2 for i in range(n):
3     b[i] = intervals[0]
4     intervals.remove(b[i])
5     # pick m[i] uniformly at random
6     # out of remaining intervals
7     m[i] = random.choice(intervals)
8     intervals.remove(m[i])
```

Figure 4.8: Algorithm to generate interval assignments (shown in Python).

as listed in Table 4.2.

First, we consider the space of permutations $\pi_j^{(i)}$. Each permutation is a random member of S_m, and each permutation is chosen independently. Therefore, the total space of permutations is $(m!)^{rn}$. Next, we must consider the space of parameters b_j and m_j. Note that it is possible to create equivalent keys by renumbering the intervals, therefore, we must count the number of non-equivalent interval assignments; we do so by defining a canonical ordering scheme such that $b_i < b_j$ for any $i < j$.

We can consider a recursive algorithm for generating a random assignment of $2n$ intervals into base–mark pairs, shown in Figure 4.8. It is easy to see that this algorithm generates every assignment with canonical ordering exactly once. The only random choice is on line 7 of the algorithm; at iteration $i(= 0, \ldots, n-1)$, there are $2(n-i)-1$ choices available. Therefore, the space of choices is:

$$(2n-1)(2n-3)\ldots(3)(1) = \frac{(2n)!}{2^n(n!)} \tag{4.19}$$

For a conservative analysis, we can assume that $o = 0$ and that the first $2n$ intervals are chosen for watermarking; this results in the minimal required watermark duration of $2nT$ ($= 128\,\mathrm{s}$ using the parameters in Table 4.3). We can thus estimate the entropy of the key choice as:

$$\log_2 \frac{(m!)^{rn}((2n)!)}{2(n!)} \quad = \quad rn\log_2(m!) \;+\; \log_2(2n!) \;-\; \log_2(n!) \;-\; n \tag{4.20}$$

Using the parameters from Table 4.3, the key entropy is over 4000 bits. We conjecture that the key equivocation is very high too and thus it is completely

Table 4.5: Average watermark delay (per packet) for different values of T/r along with the detection performance ($\sigma = 10$ msec, results averaged over 500 runs).

r	T/r (msec)	Average delay (msec)	Maximum delay (msec)	Mean true intervals (out of n=32)	Mean false intervals (out of n=32)
10	200	53.77	200	29.56	2.67
20	100	17.91	100	26.3	2.70
30	66.7	11.84	66.67	23.40	2.43
40	50	9.05	50	20.26	2.45

infeasible for an attacker to guess the secret key.

4.4.2 Single flow invisibility

In this section we demonstrate the infeasibility of distinguishing between SWIRL watermarked flows and benign flows by an attacker who does not have access to the watermark key.

Delay. As mentioned in Chapter 1, a flow watermark is required to be invisible. The magnitude of delays makes a scheme easier or harder to identify. The maximum delay inserted by SWIRL is:

$$D_{max} = \frac{T}{r}\left(2 - \frac{2}{m}\right) \tag{4.21}$$

Table 4.5 shows the average watermark delay over the packets for different values of the redundancy parameter r, with T fixed at 2s. As evident from (4.17) and (4.14), lower r will reduce the number of false negatives at the cost of higher delay.

Information-theoretic tests. We also test the invisibility of the SWIRL using the information-theoretic tools designed by Gianvecchio et al. [50] for the detection of covert timing channels. We use two entropy tests of EN and CCE and apply them over a database of SSH flows, collected from real traces at the North Carolina State University (the average rate of the flows is 4.4pps). The tests are evaluated for two classes of flows: a) regular non-watermarked flows, and, b) the same flows watermarked with SWIRL (each

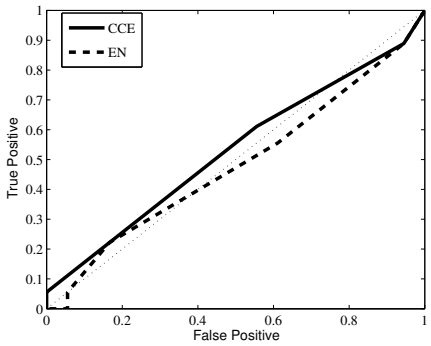


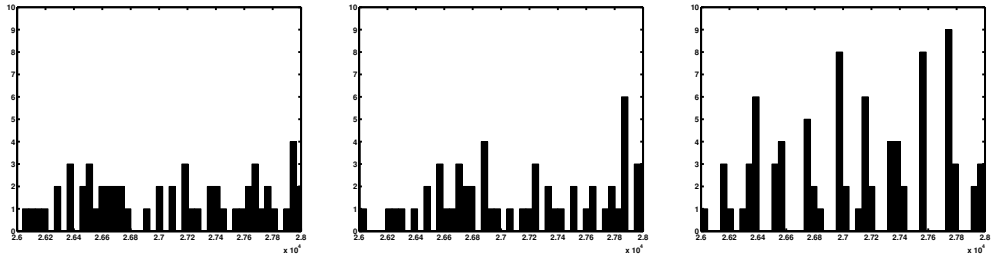
Figure 4.9: The ROC curves for the EN and CCE tests.

flow is 2000 packets long), with 10 tests per class. We then try different decision thresholds to decide whether a test metric corresponds to a watermarked flow. Figure 4.9 draws the ROC curves for the EN and CCE test metric, where the *true positive* is the odds of detecting a watermarked flow and the *false positive* is the odds of declaring a non-watermarked flow to be watermarked. As can be seen, the test metrics are not able to provide a confident separation between non-watermarked and SWIRL watermarked flows.

4.4.3 Multiple flow invisibility

Kiyavash et al. [31] show how multi-flow attacks (MFA) can be applied to compromise invisibility of interval-based flow watermarking schemes [10, 13, 14]. The main idea of the MFA attack is to collect a number of network flows watermarked by an interval-based watermarking scheme, using the same watermark key, and aggregate these flows to extract watermarking parameters and the watermark key. More specifically, the MFA attacker evaluates the *aggregate histogram* of a number of watermarked flows in order to find the watermark patterns, e.g., empty intervals, that are similar for all of the watermarked flows. The MFA attack has been shown to be highly effective in compromising previous interval-based watermarking schemes [10, 13, 14]. We analyze the resilience of SWIRL to multi-flow attacks.

The flow-dependent approach taken by SWIRL is designed to resist multi-flow attacks. In particular, since different flows have different watermarks, an aggregated histogram should not exhibit any repeated patterns. However,



(a) Non-watermarked histogram

(b) Watermarked histogram ($q = 5$)

(c) Watermarked histogram ($q = 1.25$)

Figure 4.10: Cumulative histogram of 10 flows, non-watermarked and watermarked with different values of q.

if the distribution of quantized values s is not uniform, an MFA attack may be able to identify the watermark. For example, if all watermarked flows use the same value of s for some interval, this will have a pronounced effect on the histogram.

The parameter q helps smooth out the distribution. In Figure 4.10, we plot the histogram of 10 non-watermarked flows, and 10 watermarked flows, using different choices for q. With $q = 5$, the variance of the watermarked flows is similar to the unwatermarked case. However, with $q = 1.25$, the histogram exhibits a clear pattern, since the number of quantification steps is too small and thus the distribution of s is heavily skewed.

In this scenario, we watermarked all flows using the same offset. By choosing randomized offset, we can destroy the synchronization between flows: any shift of at least $T/(mr)$ will result in completely unaligned flows. As discussed by Kiyavash et al. [31], an adversary could still examine different potential alignments; however, when parameters from Table 4.3 are used, it would be necessary to examine 640^k alignments to find the correct alignment of k flows; this is both computationally expensive and is also deleterious to the false-positive detection rate for an MFA attack.

Note that, although increasing the q parameter improves resilience to the multi-flow attack, it also increases the false-negative rate. We demonstrate this in Table 4.6 by plotting the number of intervals counted as detected among a sample of both watermarked and non-watermarked flows. We note that the true detection rate falls with increasing q, whereas the false positives remain unaffected (for a given threshold η); this is consistent with the analysis in Section 4.2, where q factors into the false-negative but not the false-positive

Table 4.6: Detected intervals for varying values of q ($\lambda = 4.1pps$, 1000 runs).

q	Watermarked		Non-watermarked	
	Mean	Range	Mean	Range
5	29.16	21–32	1.71	0–7
2.5	29.44	23–32	1.69	0–6
1.6	29.61	22–32	1.82	0–8
1.25	29.78	23–32	1.77	0–7

calculations.

Based on these results and the effect seen in Figure 4.10, we pick $q = 2.5$ to balance detection performance and susceptibility to the MFA attack.

4.4.4 Active attacks

An adversary may use a more active approach to detecting and removing watermarks; e.g., by sending packets with embedded timestamps [29] to detect extra delays, or by introducing extra delays at the stepping stone. It is easy to see that, in the limit, the attacker can defeat any traffic analysis scheme by generating an independent packet schedule for a relayed flow, using dummy packets and introducing potentially large delays [8]. Previous work on stepping stone detection has considered limiting an attacker by a maximum tolerable delay [7]; however, we expect that a normal user would be less tolerant of added delays than a determined attacker, and a blind watermarking scheme that introduces delays that are much shorter than those it tolerates remains elusive and is an apt area for future research. We note that SWIRL will work well at detecting stepping stones and other relays over which the attacker does not have full control, as is the case in the application described in Section 6.2.

4.5 Conclusions

We proposed SWIRL, a novel flow-dependent watermarking scheme for network flows. SWIRL uses an interval-based structure in order to provide robustness to network perturbations, while evading multi-flow attacks by making the watermark dependent on the containing flow. SWIRL performs blind

watermarking, reducing the communication overhead and computation overhead compared to passive traffic analysis or non-blind watermarking schemes. We show through analysis, simulation, and experiments that SWIRL is able to link related flows using flow lengths as short as 2 minutes, while providing error rates on the order of 10^{-6} or less. SWIRL introduces short delays on average and it is undetectable using existing covert channel detection tools. We also show in Section 6.2 that SWIRL can be used to address a congestion attack on the Tor network.

CHAPTER 5

CODING-BASED FLOW FINGERPRINTING

The main objective of network flow watermarking is to identify flows that have been tagged (by a watermark) at a certain point in the network. In simpler words, for any observed network flow a watermark detector seeks the answer to the following question: *Has this flow been tagged by any of the watermarking agents?* However, in some scenarios it does not suffice to just check whether a flow has been tagged (i.e., watermarked), but additional information needs to be checked about the observed flow. We call the message containing such additional information a *flow fingerprint* and the act of inserting fingerprints inside a network flow as *flow fingerprinting*. Flow fingerprints can provides information about the origin of the observed flow, its relation to other observed flows, the identity of its tagger, and other information depending on the specific application. Example questions targeted by flow fingerprinting are: *Which specific fingerprinter (out of all fingerprinters) tagged this flow? Which specific flow is related to the observed flow? In which part of the network was this flow tagged?* and so on. Flow watermarking has been well studied in the past decade and several watermarking schemes have been proposed and analysed by researchers [10, 11, 13, 14, 42]. However, the importance of flow fingerprinting has been overlooked. In fact, in some applications, the use of flow watermarks is insufficient to achieve the objectives of traffic analysis, as more information needs to be inferred from an intercepted flow rather than the fact that the flow has been tagged. For instance, consider a case where attackers aim to break anonymity by finding the sender/receiver relations among the users of an anonymity network. Flow watermarking has long been considered as a tool to conduct this attack by having the attacker insert a watermark on flows entering the anonymous network and to check for the same watermark on flows leaving the anonymity network. Unless the attack is targeted on a specific pair of sender and receiver, the attackers need to be able to distinguish between the tags that

have been placed on the flows of different users (in order to be able to link a receiver to the right sender); the tags tailored for different flows are flow fingerprints.

In this chapter, we study fingerprinting of network flows by modifying flow patterns; to the best of our knowledge we are the first to delve into this area.[1] Our fingerprinting approach is similar to flow watermarking in that fingerprinting is performed by making slight modifications to the patterns of network flows. This pattern modification embeds a message inside the network flow, the *flow fingerprint*, that is later extracted from the flow after passing through a possibly noisy network. Flow fingerprinting embeds *various* tags, i.e., fingerprints, on different flows, so that each fingerprinted flow contains a different message. This is in contrast to flow watermarking, where all watermarked flows contain the same message of "this flow has been tagged." From an information theoretical point of view, a flow watermarking system embeds a *single bit* of information on each network flow, whereas a fingerprinting system embeds *multiple bits* of information on each flow. As a result, extracting fingerprints from an observed flow is a more challenging problem as compared to watermark detection. It should be noted that several flow watermarking systems work by tagging flows with a watermark that contains a sequence of binary numbers, called "watermark bits" [10, 11, 42, 59]. Even though these schemes provide mechanisms for *detect*ing flows that contain a particular sequence of watermark bits they are not able to *extract* these bits from a target flow reliably, so we do not consider them as flow fingerprinting schemes. For instance, Ling et al. [59] propose a Tor-specific flow watermark that works by embedding a secret watermark sequence on a Tor flow through making modifications to cell [60] counters. More specifically, a compromised/malicious Tor relay manipulates the number of Tor cells being sent together on a Tor circuit so that each single cell denotes a "0" bit and each triple cell denotes a "1" bit. As network congestion and network delay can separate and merge cells in a circuit, the authors in [59] use a "recovery mechanism" to detect the distorted bits for a given (i.e., fixed) sequence of watermark bits. This mechanism, however, is not able to reliably extract watermark bits from a candidate flow, hence it can not

[1] By fingerprinting we mean fingerprinting in network pattern, which is suitable for encrypted traffic. Payload fingerprinting of non-encrypted flows have been studied before in the literature.

distinguish between different watermark sequences. For instance, consider a Tor flow watermarked by [59] with a watermark sequence of "010". If the triple cells containing the "1" bit are split due to the network, a watermark detector will return a positive correlation against both "010" and "00000" watermark sequences (for "010" the detector would assume that the flow has been perturbed, and for "00000" it would assume a perturbation-free connection). [59] does not study the extent of such cross-watermarks false positive errors (it only analyzes false positives against non-watermarked flows).

A fingerprinting system consists of two main entities: one (or more) *fingerprinter*(s) and one (or more) *fingerprint extractor*(s). A fingerprinter slightly modifies the pattern of an observed network flow, such that it carries a *fingerprint* message, e.g., a sequence of information bits. A fingerprint extractor tries to extract the embedded fingerprint message from the flow after it has passed through a noisy network. A fingerprint should be *invisible*, meaning that an entity not part of the fingerprinting system should not be able to distinguish between a fingerprinted flow and a regular flow. Also, fingerprints should be *robust* to noisy networks, meaning that a fingerprint should not vanish even after its carrying flow has passed through a low-latency communication network that delays packets. To ensure robustness, the patterns used for fingerprinting should keep the fingerprint even after passing through low-latency communication networks and after (re-)encryption of packet payloads. The possible patterns for such intent are packet timings, flow rate, number of packets in specified intervals, and so on. In this chapter, we design fingerprinting systems that are based on modifying timing patterns of flows.

In this chapter, we design a class of flow fingerprinting schemes, called Fancy, that use coding algorithms in their design. Fancy uses a topology similar to the RAINBOW flow watermarking system (Chapter 3), and leverages communication codes to be able to insert multiple bits of information on each flow reliably. Several flow watermarks have used simple coding schemes, such as spread-spectrum [14] and repetition codes [10, 13], to improve detection performance of flow watermarks; however, flow fingerprinting has been largely underexplored. We investigate the use of several powerful coding schemes in the design of our Fancy flow fingerprint. In particular, we design and evaluate Fancy using codes from three main classes of linear error-correcting codes, i.e., block codes, convolutional codes, and turbo codes [61]. We simulate Fancy and evaluate its fingerprinting performance using different

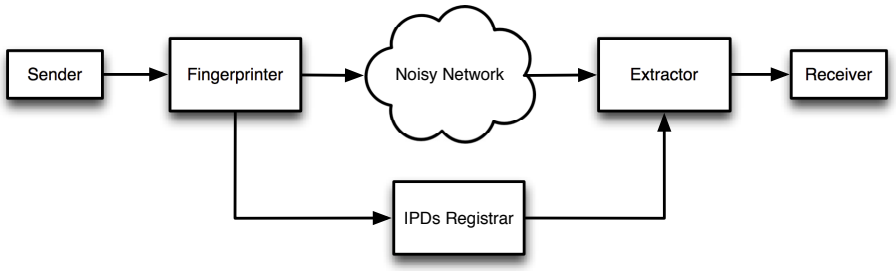


Figure 5.1: The main model of Fancy fingerprinting system.

coding algorithms.

The rest of this chapter is organized as follows: in Section 5.1 we describe the design of our proposed flow fingerprinting scheme, Fancy. We provide simulation results of Fancy in Section 5.2. Finally, the chapter is concluded in Section 5.3.

5.1 Fancy fingerprinting scheme

In this section, we detail the design of our flow fingerprinting system, Fancy. The high-level design of Fancy fingerprinting system is similar to that of RAINBOW flow watermarking system (Chapter 3), but it differs in the way embedded messages are generated and shaped, as discussed in the following.

A flow fingerprinting system should consist 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 pattern of intercepted flows, trying to extract fingerprint messages inserted by the fingerprinters. Similar to the RAINBOW watermark, our Fancy fingerprinting scheme is *non-blind*: the fingerprinter communicates with fingerprint extractors some information about the flows being fingerprinted. To perform this communication, Fancy uses a third element in its design, *IPDs registrar*, that is accessible by fingerprinters and fingerprint extractors. Fancy fingerprinter store some information on the IPDs registrar, whihc are used by Fancy extractors for fingerprint extraction. Figure 5.1 shows the high-level block diagram of Fancy.

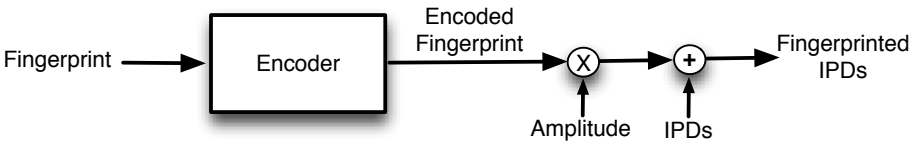


Figure 5.2: Fingerprinting scheme of Fancy.

5.1.1 Embedding fingerprints

Fancy fingerprinter's design is shown in Figure 5.2. 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 sequence, $\boldsymbol{f} = \{f_i | f_i = \pm 1, i = 1, .., \ell\}$, 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., $\boldsymbol{\tau} = \{t_i | i = 1, .., \ell^c, \tau_i = t_{i+1} - t_i\}$, 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 5.1.3 we discuss the reason for both embedding the fingerprint into the flow, and recording it in the IPDs registrar).

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

$$\tau_i^c = \tau_i + a \cdot f_i^c \tag{5.1}$$

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 +1 and -1 values the number of all possible, distinct fingerprint sequences is 2^ℓ. A fingerprinter encodes an ℓ-bits fingerprint sequence, $\boldsymbol{f}$,

into and 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 encoder being used, i.e.,

$$r = \ell^c/\ell. \tag{5.2}$$

5.1.2 Extracting fingerprints

Suppose that a Fancy extractor recieves 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 denoted being encoded/fingerprinted and the superscript r denoted 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:

$$f_i^{r,k} = (\tau_i^{c,r} - \tau_i^k)/a \qquad i = 1, .., \ell^c \tag{5.3}$$

   where $\tau_i^{c,r}$ are 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 of the encoder block used by Fancy fingerprinters.

3. The extractor declares that the received flow contain the fingerprint sequence $\boldsymbol{f}^k$ if it is the same as the decoder's output, i.e., if we have that:

$$f_i^{d,k} = f_i^k, \qquad \forall i \in \{1, .., \ell\} \tag{5.4}$$

Otherwise, the extractor uses the next flow record from the IPDs registrar and repeats the steps above.

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

- Case 1: 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

$$\tau_i^{c,r} = \tau_i^c + \delta_i \tag{5.5}$$

where δ_i is the network jitter applied on the i-th IPD and τ_i^c is the fingerprinted IPD. In this case, using (5.1), (5.5), and (5.3) we get that:

$$\begin{aligned} f_i^{r,k} &= (\tau_i^{c,r} - \tau_i^k)/a & (5.6)\\ &= (\tau_i + \delta_i + a f_i^c - \tau_i)/a & (5.7)\\ &= f_i^c + \delta_i/a & (5.8) \end{aligned}$$

In other words, using the right flow record from the registrar the step 1 will generate a perturbed version of the coded fingerprint. As a result, passing this perturbed coded fingerprint through the decoder (step 2) can return the embedded fingerprint, depending on the noise conditions and decoder performance (in Section 5.2 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: 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 algorithm will result in a non-relevant fingerprint sequence. The odds that this non-relevant 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 very tiny false extraction rate, so we neglect this case in the performance evaluation of Fancy.

Filtering flow records In order to speed up the extraction process, the extractor can leave 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 that the packet count of the received flow. A second filter that we use is a IPDs-distance metric, to leave out the records with non-similar IPDs. In particular, we evaluate the distance between a received flow with IPDs $\boldsymbol{\tau}^r$ and a flow record $R_k = (\boldsymbol{\tau}^k, \boldsymbol{f}^k)$ as:

$$d(\boldsymbol{\tau}^r, \boldsymbol{\tau}^k) = \frac{1}{\ell^c} \sum_{i=1}^{\ell^c} (\tau_i^r - \tau_i^k - f_i^c) \tag{5.9}$$

If R_k is the right record corresponding to the received flow this distance metric gives the average network jitter on the path. The extractor considers only those records from the registrar whose distance from a received flow are smaller than a threshold, η. By putting η equal to four times the standard deviation of network jitter the odds that the right record will be left out is around 10^{-4} (assuming a Laplace distribution for network jitter as in [11]).

5.1.3 Alternative designs

As described in this section, in order to fingerprint a flow n with the fingerprint message 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 f into the network flow n. Alternatively, one could suggest to only record the fingerprint in the registrar, or only embed it into the flow. In the following we back our design decision (i.e., putting the fingerprint in the registrar *and* on the flow), by discussing the performance degradation of the two alternatives.

Passive fingerprinting An alternative approach to Fancy is to only record fingerpint sequences in the IPDs registrar, along with their corresponding IPDs sequences, however do not embed the fingerprints into the flows. In fact

this approach is passive traffic analysis, which has extensively been studied in the literature [7–9]. Unfortunately, this approach may result in high-rate of false extractions, especially when the evaluated flows are cross-correlated. The common examples for correlated network flows are web traffic (to the same websites), and file transfers. In fact, our simulations of real web traffic in Chapter 3 shows that even an optimum passive traffic analysis scheme can produce very large false correlations (see Table 3.7).

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 extract a fingerprint sequence that is different from what is really embedded into that flow. By recording the fingerprints in the registrar as well, which is done in Fancy, the extractor can easily identify such error extraction by simply comparing the extracted fingerprint from the one recorded in the IPDs registrar.

5.2 Simulation results

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 [61] considering the conditions of our communication channel. Based on our measurements over Planetlab [39] 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 [62], 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 result in higher latencies in extracting fingerprints from the flows. For these parameters, we look for appropriate coding algorithms to be used by Fancy's encoder. Dolinar et al. [63] 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 [63]) 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 Plabetlab [49], and by using Matlab's built-in coding functions and the CML coding library [64].

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

Extraction Rate (P_E): This metric is the rate of fingerprinted flows that are successfully extracted by Fancy extractor.

Miss Rate (P_M): This is the rate of fingerprinted flows that Fancy extractor is not able to extract correctly, and considers them as non-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).

5.2.1 Reed-Solomon (RS) codes

Reed-Solomon (RS) codes [61] are a class of linear block codes that are maximum distance separable (MDS), i.e., they meet the equality criteria of the *singleton bound* [61]. 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 the message bit stream. This makes them a good candidate for our application since bursty noise may happen to inter-packet delays due to temporal network congestions. In particular, the RS codes have been used in satellite communications for many years because of their strength facing bursty errors. 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

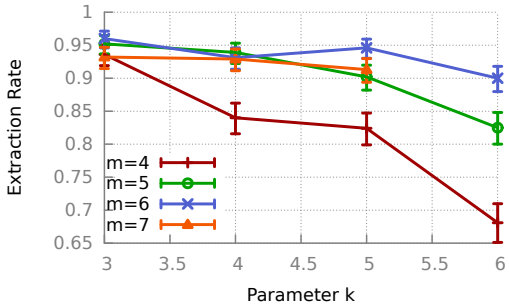


Figure 5.3: Extraction rate of Fancy-RS for different RS encoders. ($a = 10ms$, and $\ell = mk$)

ℓ^c-bits coded fingerprint $\boldsymbol{f}^c$ from an ℓ-bits fingerprint sequence $\boldsymbol{f}$ by passing it$\boldsymbol{f}$ through an (n, k)-RS encoder. We have that $\ell^c = n/k\ell$.

We simulate Fancy-RS in Matlab. We use traces of network jitter gathered over Plabetlab [49] 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 5.1.2 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 5.3 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 5.4 shows the coding redundancy of Fancy-RS for different parameters of the RS code. Finally, the number of distinct fingerprints, N, that can be embedded and extracted reliably by Fancy-RS is given by

$$N = 2^{mk} \tag{5.10}$$

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 5.5,

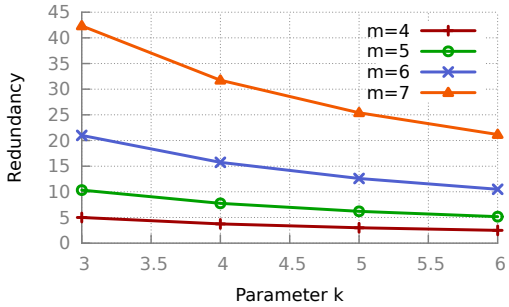


Figure 5.4: Coding redundancy of Fancy-RS for different RS encoders. ($a = 10ms$, $p = 2$, and $\ell = mk$)

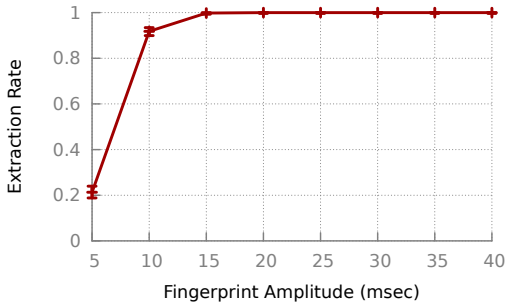


Figure 5.5: Extraction rate of Fancy-RS for different fingerprint amplitudes. ($m = 5$, $k = 5$, i.e., the redundancy is 6.2)

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 a makes the fingerprint less invisible, as discussed in Chapter 3.

5.2.2 Convolutional codes

Convolutional codes are another class of linear error-correcting codes that have use in several different applications [61]. In this chapter we design a flow fingerprint, Fancy-Conv, that uses convolutional codes. 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

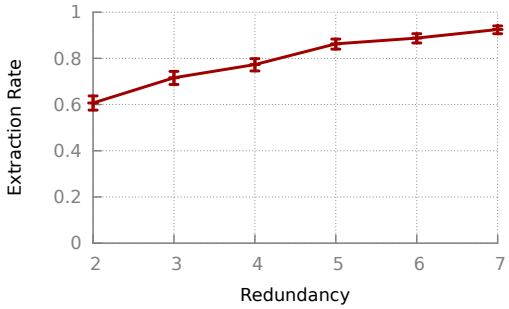


Figure 5.6: Extraction rate of Fancy-Conv for different encoder redundancies. ($a = 10ms$)

Viterbi algorithm [61].

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 that generates the final encoded fingerprint $\boldsymbol{f}^c$ consisting of ℓ^c bits.

We implemented Fancy-Conv in Matlab using the CML [64] 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 5.6 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 the redundancy, i.e., $\ell^c = r\ell$.

We also measure the effect of the fingerprint amplitude on the detection performance. As can be seen from Figure 5.7, 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 as discussed before. 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 $N \approx 10^7$.

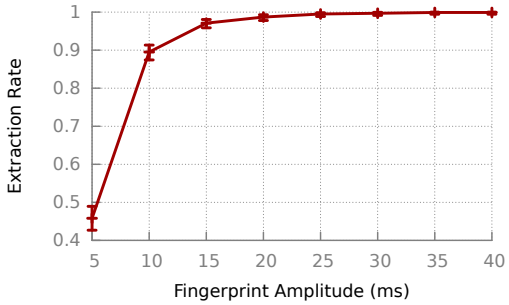


Figure 5.7: Extraction performance of Fancy-Conv for different fingerprint amplitudes. ($\ell = 18$, and the convolutional code's redundancy is 8)

5.2.3 Turbo codes

Turbo codes are a class of high-performance error correction codes and are the first practical capacity-approaching codes [65]. 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 [65]. 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 [64] 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

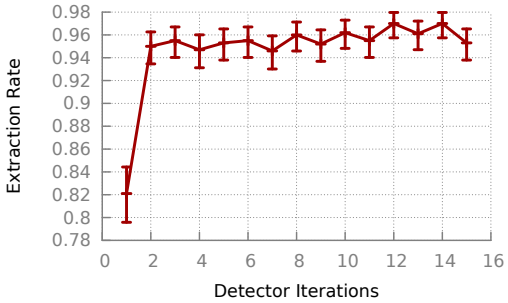


Figure 5.8: Extraction rate of Fancy-BTC for different values of maximum decoder iteration. ($a = 10ms$, $\ell = 25$, and $r = 5.95$)

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 5.8 shows the extraction rate for different values of 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.

In the second experiment, we keep $v_x = v_y = 6$ and $a = 10ms$, but vary the fingerprint length ℓ. Figure 5.9 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 5.10 shows

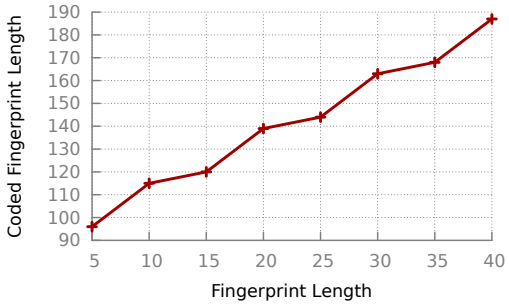


Figure 5.9: The length of the coded fingerprint for different fingerprint lengths for Fancy-BTC.

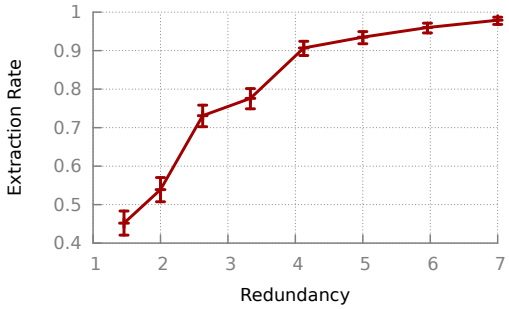


Figure 5.10: Extraction rate of Fancy-BTC for different redundancies of the BTC code. ($a = 10ms$, $\ell = 25$)

the extraction rate for codes with different redundancies. As can be seen, the 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.

Finally, we illustrate the effect of the fingerprint amplitude on the extraction performance. As can be seen from Figure 5.11, increasing a rapidly improves the extraction, such that $a = 20ms$ results in an extraction rate of in 1000 runs of the experiment.

5.2.4 Comparison

We also compare the performance of the three versions of Fancy, i.e., Fancy-RS, Fancy-Conv, and Fancy-BTC. Figure 5.12 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

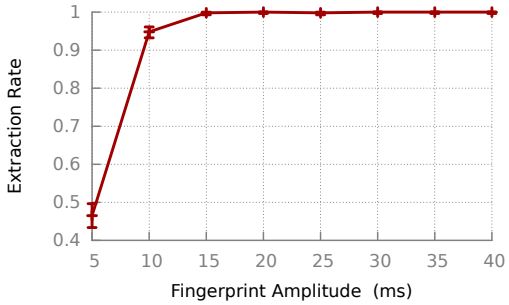


Figure 5.11: Extraction rate of Fancy-BTC for different values of fingerprint amplitude. ($\ell = 25$, and $r = 4$)

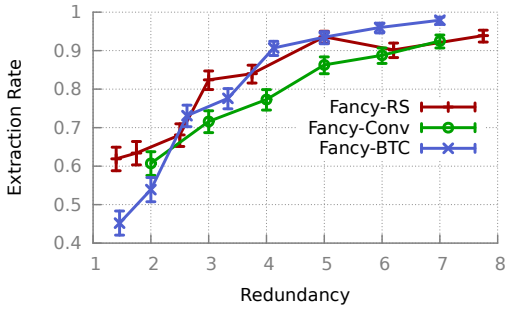


Figure 5.12: Comparing the extraction performance of Fancy-RS, Fancy-Conv, and Fancy-BTC for different code redundancies.

the schemes perform poorly for such small values of redundancy. As the redundancy increases, all scheme 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 5.13 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 more perturbations applied to the fingerprinted flows.

5.3 Conclusions

We designed a reliable flow fingerprinting system, called Fancy, which can be used to send additional information about the flows being tagged. Fancy uses

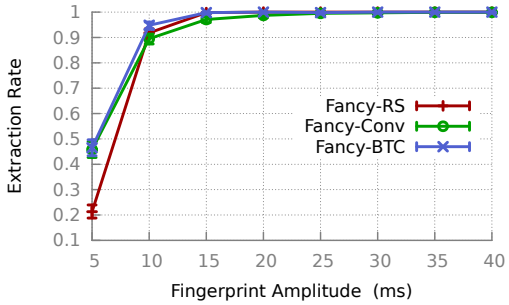


Figure 5.13: Comparing the extraction performance of Fancy-RS, Fancy-Conv, and Fancy-BTC for different fingerprint amplitudes (with a similar redundancy of around 6).

a design similar to Rainbow flow watermark, and uses coding algorithms to ensure reliable communication of fingerprint bits. We simulate Fancy using several popular coding algorithms. Our simulations show that Fancy can reliably send tens of fingerprint bits over flows with only about 100 packets.

CHAPTER 6

OTHER APPLICATIONS OF FLOW WATERMARKS

In the previous chapters we discussed how network flow watermarking, which is an active approach for traffic analysis, improves the performance of traffic analysis as compared to the traditional passive traffic analysis. In particular, we described the use of network flow watermarking for two specific applications: detection of stepping stone attacks, and compromising anonymity networks. These applications share a common objective that lets flow watermarking be used as a solution for them: to link network flows that have traversed low-latency communication networks. In other words, what makes the use of network flow watermarking effective is the fact that flows that are being linked have passed through low-latency networks that do not destroy the watermark pattern. Based on this, we can make the following general statement regarding the application of network flow watermarking: *flow watermarking can be used in any networking application in which linking flows is desired, but is not feasible by analyzing the communication contents due to encryption or other means of evasion, and in which the communication network affecting the flows of interest is low-latency.*

In this chapter, we seek alternative applications for flow watermarks, other than the two applications that are well-known in the literature and that are described in the previous chapters. In particular, we introduce two novel applications for flow watermarking: one targets the detection of centralized botnets and the other targets solving an open-challenge against the availability of the Tor [4] anonymous network.

6.1 BotMosaic: botnet detection using flow watermarks

A *botnet* is a network of compromised machines, *bots*, that is controlled by one or more *botmasters* to perform coordinated malicious activity. Botnets are among the most serious threats in cyberspace due to their large size [66]. This enables the bots to carry out various attacks, such as distributed denial of service, spam, and identity theft, on a massive scale.

Botnets are controlled by means of a command-and-control (C&C) channel. A common approach is to use an Internet Relay Chat (IRC) channel for C&C: all the bots and a botmaster join a channel and the botmaster uses the channel to broadcast commands, with responses being sent back via broadcast or private messages to the botmaster. The IRC protocol is designed to support large groups of users and a network of servers to provide scalability and resilience to failures, thus it forms a good fit for providing a C&C infrastructure. Because of their simple design and deployment, IRC botnets have been widely used by cybercriminals since 2001 [67]. Some botnets use a more advanced structure, with bots communicating directly with each other in a peer-to-peer fashion, but recent studies show that *many existing botnets use the IRC model* because of its simple-yet-effective structure [67, 68]. In this research we focus on the IRC botnets.

Much research has been devoted to the detection of IRC botnets [69–74]. However, most effective detection techniques are complex and have potential to generate false positives. This means that organizations with a large security budget are able to find potential bot infections and disable, investigate, and disinfect affected machines. Organizations with less developed IT practices, as well as home users, however, remain vulnerable to bot infections and provide a fertile ground for botnets, allowing them to remain strong.

We propose a technique that follows a service model. It leverages the efforts of one organization to capture and instantiate bot instances to provide low-cost detection of bots in other networks. We develop BotMosaic—a watermark that, when inserted into the communication between the captured bots and an IRC server, creates a pattern that is observable at other sites hosting botnets. The pattern can be recognized simply by observing the timings of the packets in a given flow, thus the detection can be carried out at a large scale by border routers. By inserting an artificial pattern, we

can ensure that false-positive rates are very low, enabling automated actions to disconnect infected bots. Since only packet timings are used, BotMosaic works even when the botnet uses encrypted connections to the IRC server.

The watermark will be visible on all connections between the bots and the IRC server. It will likewise appear in the connection from the botmaster to the IRC server. Botmasters typically use stepping stones [3] to hide their true location. The watermark can be used to detect such stepping stones and aid in botmaster traceback.

A novel and unique feature of our watermark is that it is *collaborative*: the watermark is inserted simultaneously into the flows of all captured bots. This is in contrast to past watermarks that affect a single flow at a time [9–15]. The collaborative feature amplifies the effect of the watermark and is necessary to create a timing pattern that is recognizable among the noise generated by traffic from other bots. In fact, *none* of the previous flow watermarks [9–15] can be used for the botnet detection application targeted in this thesis: a bot connection watermarked using a non-collaborative scheme gets destroyed once it is mixed with flows from other bots in the C&C channel, whereas the collaborative watermarking of BotMosaic is able to persist in the mixed C&C traffic of a botnet.

In summary, BotMosaic has the following unique features as compared to previous approaches: 1) BotMosaic is implemented by one organization, and can be used as a *low-cost* service by other organizations, i.e., *client*s. A client organization only needs to deploy the low-cost watermark detectors of BotMosaic on their border routers. This is in contrast to other approaches that suggest each organization to deploy its own, resource-intensive botnet detection mechanism. 2) A client organization can use BotMosaic to detect various instances of bots simultaneously, without the need to modify its BotMosaic detectors for different botnets. The BotMosaic watermarkers use different watermark signals for different instances of botnets. 3) Each client organization can detect not only the bot infected machines, but also the botmasters and stepping stones hosts residing *inside* their networks.

We analyze our scheme using simulations and experiments on PlanetLab [39]. We find that we can achieve a high rate of detection with few false positives using a watermark applied to captured/imitated bots that comprise a small fraction of the botnet, with a detection time of about a minute.

The rest of the chapter is organized as follows: Section 6.1.1 describes previous work on IRC botnet detection. Section 6.1.2 describes the overall detection framework used by BotMosaic. Section 6.1.3 describes the detailed structure of the BotMosaic collaborative watermark. Simulations and implementation results are presented in Section 6.1.4. Section 6.1.5 offers a brief discussion of some additional issues, and Section 6.1.6 concludes the discussion.

6.1.1 Related work and motivation

The primary goal of the BotMosaic is to detect bot-infected machines inside a network of interest, e.g., an ISP. The literature on this can be divided into *host-based* and *network-based* approaches. Host-based approaches analyze the information on hosts of the network; this is not easy to deploy on all hosts, especially in organizations where computers are not centrally managed. BotMosaic falls in the network-based category.

Network-based detection mechanisms aim to detect bot infected machines by analyzing the network traffic information. These mechanisms mainly are classified into two categories: *traffic signature* schemes and *traffic classification* schemes. The traffic signature approaches use the captured bots to develop signatures for each botnet instance; they have widely been used for IRC botnet detection [69, 71, 75]. As an example, Blinkley and Singh [69] combine IRC statistics and some TCP metrics to generate signatures that can be used to detect the infected machines. Traffic classification approaches are based on gathering network traces and clustering them in order to detect botnets based on their behavioral difference with the normal traffic [70,72,73]. As an example, Villamar et al. use Bayesian methods to isolate centralized botnets, based on the similarity of their DNS traffic with those of some known DNS botnet traces [73].

In this chapter we consider a third approach for performing network-based bot detection. BotMosaic uses *network flow watermarking* to mark the botnet traffic, resulting in low-cost mechanisms for the detection of bots and botmasters. Network flow watermarking is a technique that actively perturbs the traffic patterns of a network flow to insert a watermark inside them that can later be detected. Flow watermarking has been used to detect stepping

stones, as well as to compromise anonymous communication [9–14]. Existing techniques, however, cannot be applied to the problem of bot/botmaster traceback for two reasons. First, they are designed to work on long-lived flows; typically, hundreds of packets are necessary to detect the presence of a watermark. Botnet communication, however, tends to be short-lived, with only a few packets sent from each bot. Furthermore, a watermark that is applied to a single bot-to-botmaster/botnet communication will be overwhelmed by traffic from other bots that will be aggregated along the same stepping stone connection. Although some of the existing watermarks are designed to resist a reasonable level of chaff, they do so by increasing the length of the watermark and thus cannot be used for botnet traceback in practice.

More recently, Ramsbrock et al. [15] designed a watermark specifically targeted to the task of botmaster traceback. Their watermark works by adding extra whitespace at the end of IRC messages sent by the bots. They also adjusted the timings of packets in order to improve detection ability. Though an important first step, the whitespace watermarking approach has several serious limitations. Whitespace watermarking only works well in the presence of low rates of chaff—less than 0.5 packets/second—whereas even in a small-size botnet, an aggregate response from all the bots would create a significantly higher chaff rate. Whitespace watermarking is also fragile to repacketization or retransmission of packets, as such events can cause it to lose timing synchronization. Finally, whitespace watermarking relies on modifying the *contents* of the messages sent by the bots, which can be be difficult if encrypted connections are used.

6.1.2 BotMosaic detection framework

In this section we describe the features of IRC botnets exploited by BotMosaic and its deployment scenarios.

IRC botnets

Internet Relay Chat (IRC) is a network protocol designed for Internet text messaging or synchronous conferencing [76, 77]. In order to use an IRC network, clients *join* an IRC *channel* created by an IRC server, providing a

nickname (the server may also require client authentication). The clients then send broadcast messages to the channel, or private messages to specific nicknames inside that channel.

During the last decade, IRC channels have been used as a common way to construct the C&C channel of botnets; the rationale behind this traditional decision by cybercriminals is the low weight of IRC client software, the simplicity of the IRC protocol, and the existence of many public IRC servers over the Internet that can be used by the botnets [67, 68]. Some examples of IRC-based botnets are SdBot, Virut, SpyBot, and RBot [68]. The infected bot hosts act as IRC clients and join a specific channel used by the botnet. Some botnets use fixed channels, while other change them dynamically in order to avoid detection and shutdown. The bot then communicates with the botmaster and other bots using the IRC channel.

The botmaster sends commands to the bots by sending a broadcast message to the channel, or by sending private messages to individual bots. For example, a botmaster might send messages such as "send me recorded passwords" or "start DDoS on target X." The bot will send responses back either as private messages (for sensitive data, such as credit card information) or public broadcasts (for, e.g., status updates). To avoid detection, bots may encrypt the contents of the messages and/or use an encrypted connection to the IRC server.

BotMosaic Architecture

In BotMosaic, a *service provider* inserts watermarks using captured bots that are then detected by a number of *client organization*s (shortly, *client*s). The service provider performs the majority of the work, whereas the clients run low-cost detection. The service provider may either charge clients to use the watermark, by selling a subscription to the watermark secret keys, or provide it as a public service.

Captured Bots: BotMosaic relies on a number of bot instances that are controlled by the service provider. These bots will be used to insert the watermark pattern into the IRC channel. To capture these bots, the service provider may deploy a honeynet [78] or manually infect a number of (possibly virtual) machines with the bot.

Watermarker: The watermarker mediates the traffic between the cap-

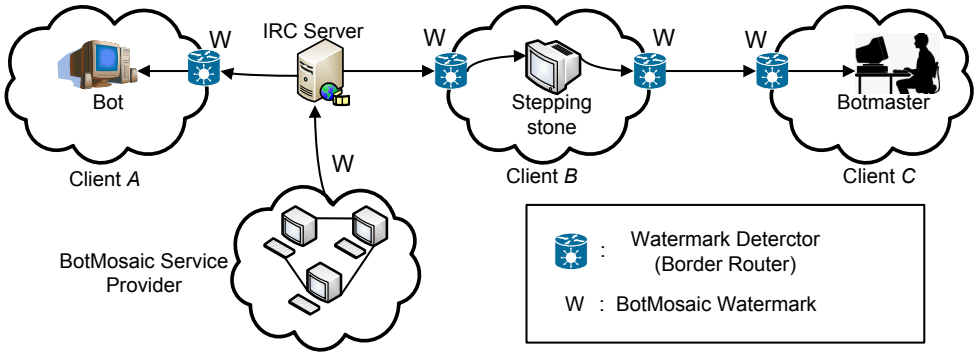


Figure 6.1: Topology of the BotMosaic traceback system.

tured bots and the outside world to insert the watermark. In particular, it delays network packets to create a particular timing pattern. The watermarker will delay packets from *all* captured bots simultaneously to ensure that the watermark signal is strong enough to be detected. Note that the watermark is content-agnostic; BotMosaic will work even if the traffic between the bots and the IRC server is encrypted.

Detector: A detector, run by the client, watches network traffic for the watermark pattern. Typically it would be deployed at or near border routers, to examine all the traffic entering and leaving the client's organization. It only needs to examine packet timings and headers (the latter to group packets into flows) to detect the mark; importantly, it does not need to perform deep packet inspection. The detection algorithms can thus be run efficiently on high-speed network links.

Value Proposition

The BotMosaic clients will benefit from the scheme in several important ways, creating incentives for deployment of the scheme by ISPs and enterprises.

Detecting Bots When the watermarker inserts a mark onto broadcast messages from the captured bots, this watermark will be observed on the traffic from the IRC server to other bots. A client running a BotMosaic detector can, therefore, detect bots hosted in its own network by monitoring incoming traffic for watermarks, such as in network A in Fig. 6.1. The detector is much lower cost than other methods of botnet detection, and the

watermark parameters can be configured to ensure an acceptably low rate of false positives.

Detecting Stepping Stones A botmaster will typically use a number of *stepping stones* to connect to the IRC channel, in order to disguise his or her identity [3]. The stepping stones will carry the watermark as well; a client can discover a stepping stone hosted within its network by observing the watermark on an *outgoing* flow, as in network B in Fig. 6.1.

A stepping stone is usually a compromised computer, thus detecting stepping stones is valuable to the client. This information can also be used to help locate the botmaster: if the first stepping stone used by the botmaster is in an organization running BotMosaic detection, the IP address of the botmaster will be revealed. Note that this remains true even if all of the other stepping stones are on networks not covered by BotMosaic.

Our approach is a variant on other techniques that use watermarks for stepping stone detection [10, 11, 13]. However, previous work required an organization to insert watermark on all inbound traffic. BotMosaic, in contrast, does not require clients to modify or delay traffic flows and thus will not interfere with the level of service provided to legitimate users. It can be deployed on a mirrored port, whereas watermark insertion must be performed inline, creating a potential point of failure.

Detecting the Botmaster Finally, a client hosting the botmaster will be able to observe the watermark on its inbound connection, as in network C in Fig. 6.1. The botmaster will receive the watermark both on broadcast messages and on private responses from bots (as long as the private responses are sent by all bots in response to a command). Thus, one way to distinguish the botmaster from ordinary bots is that, in some instances, the botmaster will be able to observe the watermark even though other bots did not.

6.1.3 BotMosaic watermarking scheme

In this section, we describe the watermarking scheme that we devise to be used for the BotMosaic botnet traceback system. The watermark is novel in being *collaborative*: the BotMosaic service provider uses *multiple* captured bots for watermark insertion, which makes the scheme specialized for the

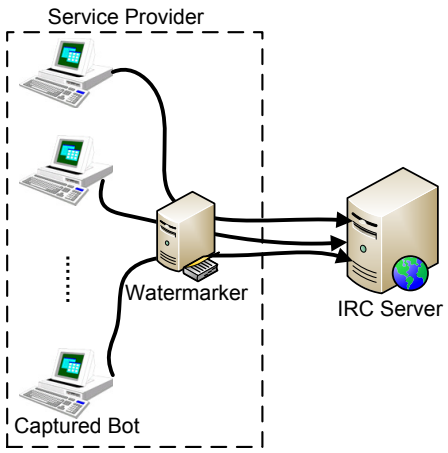


Figure 6.2: BotMosaic service provider structure.

problem of botnet watermarking. Multiple captured bots allow us to spread the watermark power over a larger amount of traffic, compensating for the small amount of traffic each individual bot sends to the botmaster/botnet. BotMosaic adopts an interval-based design used in other watermarks [10, 13, 14] to provide robustness to packet losses, delays, and reordering as well as chaff introduced by the traffic of the other bots in the botnet.

Watermark insertion

Fig. 6.2 shows the structure of the BotMosaic service provider. The BotMosaic service provider starts R captured bots, joining and communicating with the botnet through the IRC C&C channel. Such connections are intercepted by a watermarker, as described later. Using virtual machines [79], R can be made reasonably large while using modest amounts of resources. Let B be the number of *active* real bots connecting through the same C&C channel. Increasing the ratio R/B improves watermark detection efficiency, as will be shown in the following sections.

Fig. 6.3 illustrates the collaborative watermark insertion on the communication of captured bots. Each of the flows contain a share of the watermark so that the mixture of the packets of these flows, combined in the botnet C&C channel, generates the watermark pattern, whereas any single captured bot flow is insufficient to detect the watermark.

The watermarker inserts a watermark sequence of length l into the cap-

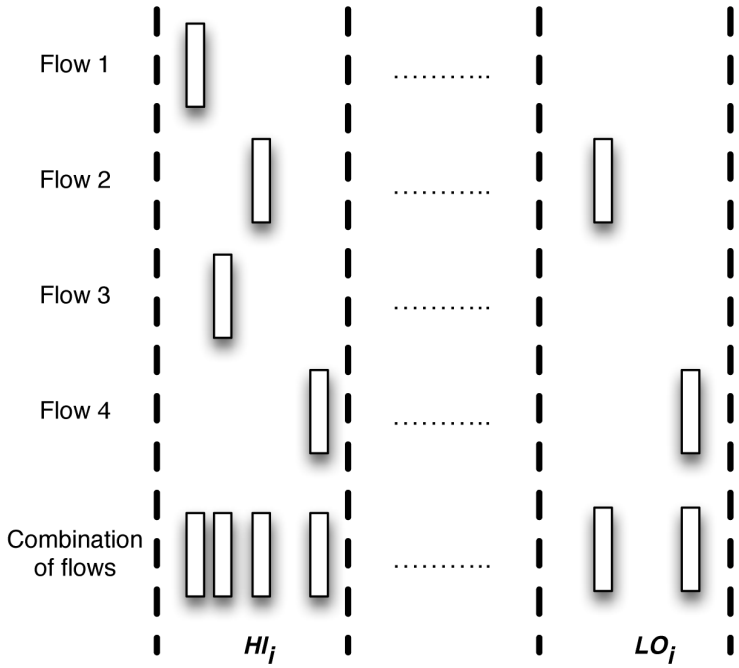


Figure 6.3: Collaborative watermark insertion using multiple flows.

tured bots flows. The watermarker divides the time axis into $2l$ non-overlapping intervals with equal length T. The intervals are labeled as $HI_1, \ldots, HI_l$ and $LO_1, \ldots, LO_l$ using a random assignment. The interval mappings form the secret watermark key that is necessary to detect the watermark; the service provider can therefore sell watermark key subscriptions to clients.

The basic idea of the watermark is to send more packets in HI interval compared to its corresponding LO interval in the mixture of all captured flows. For each HI-LO pair, the watermarker assigns the timing of R captured flows so that in the mixture of all R captured flows the number of packets appearing in the HI_i interval of that is larger than the number of packets in the corresponding LO_i interval by some threshold η (see Fig. 6.3). In other words, we should have:

$$\sum_{f=1}^{R} N_f(HI_i) - \sum_{f=1}^{R} N_f(LO_i) \geq \eta \qquad i = 1, \ldots, l \tag{6.1}$$

where $N_f(\cdot)$ gives the number of packets showing up in the given interval of f^{th} captured flow

For the manipulation of the captured flows the watermarker first determines the total number of packets in each interval of the aggregated water-

marked flow and then assigns the number of packets to each captured flow, namely *watermark shares*. Based on IRC standards [77] the rate of the flows from IRC client to the server should not exceed 0.5 packets per second (exceeding this threshold causes the client to be penalized by extra delays on its packets). So, in an interval with length T seconds we expect to have at most $\frac{T \cdot R}{2}$ packets accumulated over all R captured flows. For each i, we randomly select the cumulative packet number for HI_i interval to be:

$$N(HI_i)\epsilon \left[\frac{T \cdot R}{4}, \frac{T \cdot R}{2}\right] \tag{6.2}$$

We then assign the total number of packets in the LO_i interval to be:

$$N(LO_i) = N(HI_i) - \eta - \psi \tag{6.3}$$

where η is the *detection threshold* and ψ is the *confidence threshold*. Doing so for all of the HI-LO pairs we find the cumulative number of packets $N(j)$ for any interval j. We then randomly distribute the $N(j)$ packets within the j^{th} interval of all R captured flows so that the overall rate of each flow does not exceed the 0.5 packets per second constraint mentioned above. For each captured flow, the watermarker buffers a number of packets before starting the watermark insertion, to make sure it has an adequate number of packets for its manipulations.

Detection Scheme

The watermark detectors deployed on the border routers of the BotMosaic clients monitor network traffic to detect the watermark patters inserted by the BotMosaic service provider (see Section 6.1.2). The detectors are provided with watermark key(s) by the BotMosaic service provider:

$$Key = (T, \{\forall i = 1, \ldots, l : HI_i, LO_i\}) \tag{6.4}$$

As mentioned before, the flows watermarked by the BotMosaic service provider get mixed with other flows, resulting in a single mixed (and usually, encrypted) flow. We assume that the target mixed-encrypted flow contains packets from R captured bots and B real bots. A watermark detector breaks up a candidate flow into intervals, and then computes $N(HI_i)$ and $N(LO_i)$.

The detector then calculates the following:

$$\Delta(i) = N(HI_i) - N(LO_i), \qquad \text{for } i = 1, \ldots, l \tag{6.5}$$

The detector uses two thresholds to decide whether a watermark is present. First, if $\Delta(i) > \eta$ (η is the detection threshold in (6.3)), the detector calls the i^{th} pair in the sequence *detected*. Note that the confidence threshold ψ used during the watermark insertion ensures that natural variations in numbers of packets do not destroy the watermark.

Finally, the detector declares the candidate flow to be watermarked if the total number of detected pairs n_c (out of l) is greater than or equal to some threshold θ, *Hamming threshold*. It is easy to see that by increasing η and θ, we can decrease the number of false positives at the cost of creating more false negatives. We will discuss parameter choices in Section 6.1.4. Due to the delays applied to the mixed watermarked flows passing through the network, the detectors need to perform *synchronization*. This is done using sliding windows, as will be discussed in section 6.1.5.

6.1.4 Simulations and experiments

In the simulations and experiments of this section we only consider the detection of BotMosaic watermarks being inserted into the traffic towards the botmaster. The detection of watermarks on traffic to bots is similar; however, botmaster detection is more difficult as the botmaster traffic is relayed through a number of stepping stones, resulting in more delays affecting the watermark pattern.

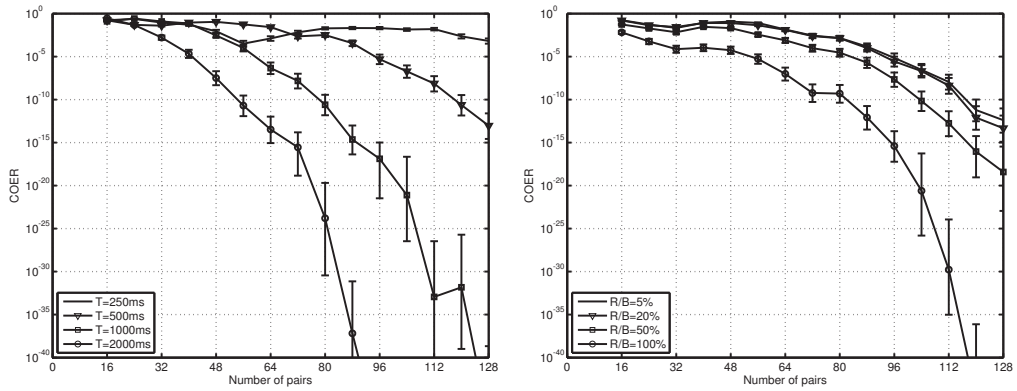
Simulations

We simulated BotMosaic in Matlab to evaluate its performance. We used the traces of botnets for *SpyBot* and *SdBot* botnets that were also used in BotMiner research [80]. The SdBot trace belongs to a botnet with a botmaster and four real bots. Since we needed a larger botnet to evaluate our scheme, we extended the trace to have 100 real bots. Based on analyzing the trace, bots listen on the IRC channel for the commands, and upon receiving a command they respond to it appropriately. To extend the botnet trace

from the existing 4-bot trace to a 100-bot trace, we sent a response to the channel on behalf of the newly added bots after a random delay whenever a command was issued and the existing bots responded to it.

To simulate the watermarking scheme, we added watermarked flows generated by the rogue bots to the trace for different settings of watermark parameters. For each run of the simulations, we generated a new extended trace, selecting a different part from the real trace randomly and extending the trace for 100 bots as discussed above. We insert the watermark into the trace and calculate the number of pairs that are detected by the detection scheme (true positive pairs); we also run detection on the unwatermarked version of the trace to and count the number of detected pairs (false positive pairs). This lets us estimate the error rates for a given threshold θ; namely, how many watermarked flows would *not* be detected (false-negative rate) and how many non-watermarked flows would be misdetected (false-positive rate). We then adjust θ so that false-positive and false-negative error rates are equal; the resulting rate is called the *crossover error rate* (COER) and we call the corresponding threshold $\hat{\theta}$.

We also add delay and jitter to the botnet traffic, based on measurements we have performed on PlanetLab [39]. In Section 6.1.5 we will discuss how to synchronize our detector with the watermarker. However, non-uniform delay for different bots, as well as network jitter, will decrease the accuracy of our detection and thus we include it in our simulation. Each experiment is run 100 times (each time with the same watermark parameters but different watermark key and different bot traces) to get the mean and variance of true-positive and false-positive parameters. Using these statistics we estimate the COER for each experiment by approximating the false error rate distributions with normal distributions. The Kolmogorov-Smirnov (K-S) test indicates that the true-positive and false-positive parameters are fitted to normal distributions with average significance levels of 0.0121 and 0.045, respectively. The average K-S distances from a normal distribution are 0.0808 and 0.0680, respectively. In our experiments, we set the number of (active) real bots to be $B = 100$, and vary the number of rogue bots, R. Fig. 6.4a illustrates the estimated COER versus the watermark length, for different values of the parameter T. The R/B value is fixed to 10%. In all of the simulations, the detection threshold η and the confidence threshold ψ are set to 1.



(a) $R/B = 10\%$ (b) $T = 500ms$

Figure 6.4: COER error of the detection scheme over SdBot botnet traces along with 95% confidence intervals.

We also estimated the $COER$ for different values of R/B ratio. Fig. 6.4b shows the $COER$ for different watermark lengths having different ratios of R/B. As expected, increasing R/B improves the $COER$ at the expense of requiring more resources to run a larger number of instances.

Table 6.1 shows the results of the experiment for two different settings of the watermark parameters (each experiment is run 500 times). For the interval length of $T = 500\,\text{ms}$ and using 64 pairs and for $R/B = 10\%$, a watermark can be inserted into a 64 second connection with the botmaster, and the resulting COER is on the order of 10^{-8}, which is very promising. Increasing the T parameter improves the COER, at the expense of needing more time for the botmaster to be online. We find that $\hat{\theta}$ is approximately $l/2$.

We also performed similar experiments over SpyBot traces from [80], leading to similar results. Table 6.1 also shows the detection results for two sample sets of watermarking parameters for the SpyBot traces. As can be seen the watermark can be detected in as few as 64 seconds with a COER of about 10^{-8}. Similar to SdBot simulations, we can trade elapsed time for COER, using different values of the watermarking parameters.

Implementation

We tested BotMosaic on PlanetLab by creating synthetic bots that use an IRC channel to communicate with the botmaster. The captured bots are

Table 6.1: Two sample runs of the detection scheme over SdBot and SpyBot traces, for $R/B = 10\%$ and a watermark sequence of length 64 (averaged over 100 random runs).

Botnet type	T	True pairs	False pairs	$\hat{\theta}$	COER	Elapsed time (s)
SdBot	250	43.9	22.4	33	$2.8 * 10^{-3}$	32
	2000	51.3	12.9	32	$3.52 * 10^{-13}$	256
SpyBot	500	48.0	16.2	32	$2.32 * 10^{-8}$	64
	2000	50.3	13.5	32	$7.55 * 10^{-11}$	256

implemented over physically separate hosts in PlanetLab. Watermark proxies are installed in front the captured bot hosts, and are controlled by a controller to insert the watermark. We route all the bot traffic through the watermark proxy. Watermark proxies are responsible for watermarking the bot traffic on all the captured bot machines, so that the accumulated traffic makes the final watermarked traffic. By using a proxy, we avoid having to reverse engineer and modify the bot code to insert watermarks.

We implemented the BotMosaic watermarking scheme over the PlanetLab infrastructure using randomly selected nodes as different entities in the experiment. Fig. 6.5 shows the structure of our experiment. A botmaster is controlling botnet through the IRC C&C. To hinder detection, the botmaster relays his traffic to the IRC server through 5 stepping stone nodes located in geographically different locations, and also encrypts the connections between stepping stones.

There are 100 real bots ($B = 100$) connected to the IRC C&C channel, listening for the commands from the botmaster and sending appropriate responses to the channel. The real bots are chosen randomly, and are located in geographically diverse locations. We set up $R = 10$ captured bots to send watermarked flows to the C&C channel ($R/B = 10\%$). The captured bots are also chosen randomly and are located in different places. A controller node commands the captured bots to join the C&C channel. Once all the captured bots have joined the channel, the controller commands all of them to start sending packets on the C&C channel containing corresponding shares of the watermark.

Again, we only provide the results for the watermark inserted on the botnet traffic to the botmaster, i.e., through PRIVMSG to the botmaster; the detection performance is the same for the watermark inserted into the traffic

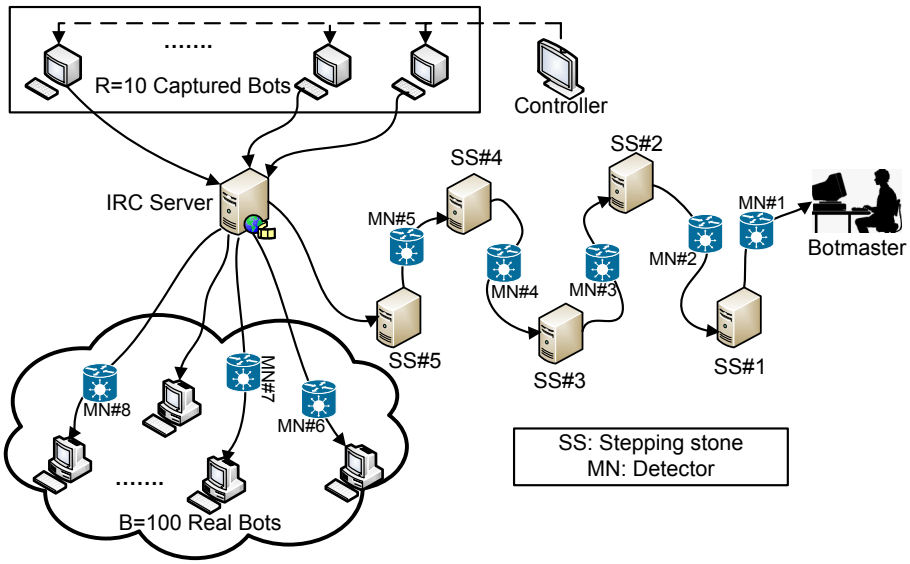


Figure 6.5: Testbed structure of BotMosaic implementation over PlanetLab.

directed to the bots.

We set up several watermark detectors across the network to look for the inserted watermark in network flows. The detector deployment is described in Section 6.1.2. We set up 5 detectors on the way to the botmaster to check the true detection rate, and also 3 detectors on the paths not leading to the botmaster to evaluate the false detection rate.

Fig. 6.6 shows detection results for different detectors. We set the T parameter of the watermarking system to be $T = 500\,\text{ms}$ and use watermarks with sequence length l equal to 32, 64, and 128. According to the simulations in the previous section, we set the Hamming threshold θ to be $l/2$ in each case. Results are normalized by the sequence length.

As the results show, detectors on the path from IRC server to the botmaster ($MN\#1$ to $MN\#5$) are able to detect the watermark from the mixed-encrypted flows, as soon as only 32 seconds. On the other hand, detectors placed on the paths not containing the botmaster watermark do not detect the watermark on the innocent flows.

6.1.5 Discussion

We briefly discuss several other issues regarding the BotMosaic scheme.

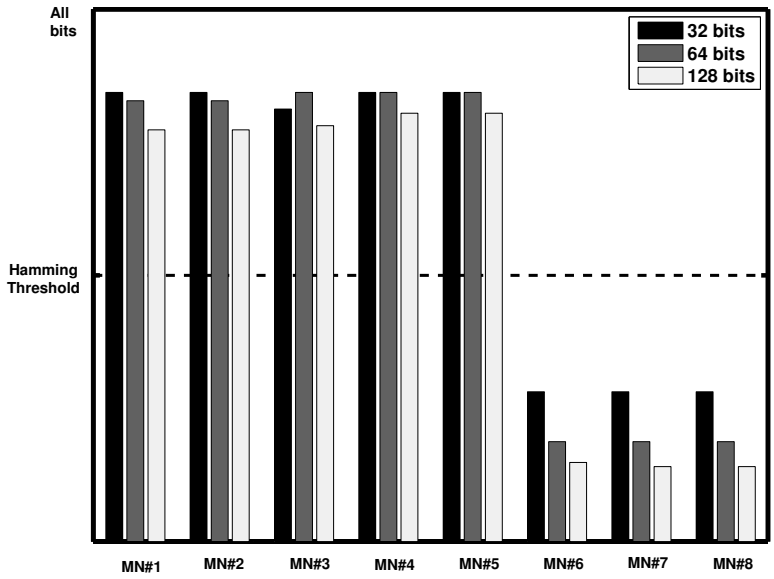


Figure 6.6: Watermark pairs detected by different detectors across the testbed (normalized by total number of pairs).

Detector synchronization

Watermark detectors need to synchronize the received watermarked flow with the watermark sequence, i.e., minimize the offset between intervals of watermarked flow and those of the watermark, in order to successfully detect the watermark. To find the right offset of the watermarked flow, we run the watermark detection scheme over the received flow applying different offset values from 0 to T in $T/100$ steps, and select the offset maximizing the number of detected pairs as the right offset for that flow. Our experiments show that running the synchronization mechanism over an non-watermarked flow, the number of detected pairs remains below the hamming threshold θ for different offset values. The use of this synchronization mechanism makes detection scheme tolerate different network delays (though variable network jitter still impacts the detection accuracy).

Resources

In order to study the processing and memory costs of BotMosaic detection scheme, we ran the watermark detector over a 21 GB network trace gathered from the routers of an anonymous US university. The utilized trace contains

Table 6.2: Resources required for BotMosaic.

l	Processing Time (μsec/flow)	Total Memory per flow (KB)	Total Memory (MB)
32	28.0	0.16	3.46
64	49.3	0.27	6.06
128	86.9	0.48	10.49

21 744 concurrent flows, with a total of 2.1 GB of timing information. Since the detection scheme should be implemented over border routers, this volume of traffic is representative of a highly loaded border router.

The experiment is done using a Unix system with a 1.6 GHz Intel CPU and 2 GB of memory. Table 6.2 illustrates the result of the experiment over the university traces. Even for a watermark of length 128, which would provide very low error rates, the total memory needed for watermark detection is about 11 MB, and the processing time for each flow is as only 87 μs. The time and processing resources are even smaller for shorter watermarks. Thus we expect that it may be possible to deploy BotMosaic even in high-performance routers used by large ISPs, to provide a better vantage point for bot detection and botmaster traceback.

Watermark evasion

As with any flow watermarking scheme, an attacker who wishes to foil watermark detection can do so by inserting large delays and other modifications to the flow structure. Therefore, an adversary with full control over an IRC server can render a botnet immune to BotMosaic. There are other evasion avenues available to the botnet designer, including the use of a peer-to-peer structure or covert communication [81] for C&C. Our goal, however, is to capture a class of existing bots that, despite using simple and well-understood C&C techniques, still comprises a large fraction of current botnets seen in the wild [67,68]. Forcing botnet operators to use more advanced C&C mechanisms imposes new costs and affects the profitability of the entire criminal enterprise.

Other issues

Traditional flow watermarking schemes consider issues like chaff, repacketization, and packet addition/removal on the performance of watermark detection. BotMosaic is designed to be robust to interfering traffic from non-captured bots and the corresponding mechanisms likewise address the above issues.

Interval-based watermark schemes are subject to a multi-flow attack discovered by Kiyavash et al. [31]. This might allow a non-subscriber to recover the secret watermark key; if this is a concern, we can use the mechanisms proposed by Houmansadr et al. [41], for example, by changing the watermark key (the set of HI and LO intervals) over time.

Finally, it might be the case that the botmaster puts limitations on the number of packets each bot can send. In this case, watermarking will still be feasible by using more captured bots to collaborate in the generation of BotMosaic watermarks.

6.1.6 Conclusions

We have presented a new botnet traceback scheme, BotMosaic, that detects bot infected machines and helps to track down the botmasters controlling the centralized botnets. BotMosaic uses a service-based approach where detector clients perform fast and low-cost watermark detection, which is much cheaper and easier to deploy than existing signature- and classification-based detectors. We presented a new collaborative flow watermarking structure, making it suitable for the botnet detection problem. We showed through experiments that our watermark can be quite effective when 5%-10% of the bots are captured/imitated by a service provider, and that our detection is simple enough to be able to handle large volumes of traffic. Any individual organization deploying the low-cost BotMosaic detectors can realize bot defense benefits, providing an incremental path to widespread deployment of the BotMosaic architecture and potential detection of botmasters.

6.2 Mitigating the Tor congestion attack

Watermarks have been traditionally seen as privacy-invasive tools, since they can be used to link relayed flows and thus compromise anonymity systems such as Tor [21]. We show that our watermark design of SWIRL, presented in Chapter 4, enables a new, privacy-enhancing use of watermarks in order to prevent a certain type of attack against Tor.

Evans et al. [82] demonstrated an attack on Tor that uses active probing to detect which Tor routers are used to forward a particular tunnel, thus breaking anonymity. Unlike watermarks or passive traffic analysis, their attack works even when the routers being used are *not* under the control or observation by the adversary. The basis of the attack comes from an earlier congestion attack, explored by Murdoch and Danezis [83]. However, a key feature of the new attack is the use of bandwidth amplification to create sufficient congestion to make this attack practical on today's Tor network.

The bandwidth amplification exploits the fact that paths in Tor can be constructed to have an arbitrary length. This, coupled with the fact that each hop on a path knows only the previous and the next hop, makes it easy to construct a path that loops through a set of routers many times. This, in turn, ensures that a single packet sent by a user will result in k packet transmissions at each of the routers in the loop, for near-arbitrary values of k.

A potential defense described by Evans et al. is to modify the Tor protocol to restrict the number of circuit extensions it allows, and thus the maximum path length. However, they point out that this is not sufficient to completely prevent such congestion attacks, as loops can still be created by going outside the Tor network and then returning. In particular, a client can create a Tor tunnel, which forwards its traffic over Tor to TCP connection from an exit node to some destination on the Internet. This TCP connection can then be used to connect *back* to Tor as a client, and repeat the circuit again. Iterating this process yields the same functionality as the long-path attack. Although a naive approach may be foiled by exit and entrance policies in Tor, the attacker can instead use proxies, other anonymizers, or hidden Tor entry and exit points. Evans et al. leave defense to such external routing loops as an open problem.

We propose to use SWIRL as a solution. The basic strategy is to configure

Tor exit nodes to insert a SWIRL watermark on all outgoing TCP traffic. Note that this labels the traffic as coming from Tor, but given that the list of exit nodes is published in the Tor directory, this does not significantly degrade privacy. Each entry guard, correspondingly, tries to detect the SWIRL watermark on an incoming TCP connection and rejects the stream if the watermark is found. This way, the congestion attack is restricted to internal paths only, which can be solved using the solution described above.

Note that this application can tolerate a significant rate of false positives (say, 10^{-3} or even higher). This is because a false positive will simply cause a legitimate user to retry a connection; given that the current Tor network does not provide very reliable service, an occasional extra failed connection is unlikely to significantly affect usability. This means that SWIRL parameters can be tuned to be able to mark shorter flows as compared with other settings. Additionally, full invisibility is not needed, as the open proxies are unlikely to be adversarial (if they were, they could simply generate the traffic themselves). Thus, the q parameter can be reduced to decrease false-negative errors.

It is important to realize that, although in principle any traffic analysis technique could be used, the properties of SWIRL make it particularly suitable for this task. Passive traffic analysis techniques (and non-blind watermarks) would require each exit node to communicate timing patterns of each exiting flow to each entry node. In addition to being very expensive, such an approach would completely defeat the protection provided by the Tor network, as each entry node would be able to detect which exit node each of its flows was using! A watermark, on the other hand, marks only the exiting flow and cannot be linked to the entry node. (Each exit node could, in fact, use the same watermark.) Additionally, other watermarking schemes introduce large delays, affecting the usability of the Tor network. The delays used by SWIRL, on the other hand, are significantly smaller than the typical latency of a Tor tunnel [84] and are unlikely to be noticed.

To study this attack, we simulated SWIRL being applied to Tor traffic flows. We used a set of flow timings observed by a Tor middle node[1] in our tests. We used a total of 14 flows that were long enough for our watermark. We then ran tests using both watermarked and non-watermarked versions of

[1]This data set was provided to us by Steven Murdoch.

Table 6.3: Watermark detection results for Tor flows.

Flow rate (pps)	True intervals		False intervals	
	mean	range	mean	range
3.25–3.57	27.51	17–32	5.89	2–11
11.58-14.33	28.76	21–32	6.88	3–14

the flows to compute the number of true and false intervals detected. The results are shown in Table 6.3. The rates of the flows had a natural separation into two classes and we present the results for each class separately. Note that our tests are most representative for a direct connection from an exit to an entry node; any proxies or other relays may introduce extra delays that affect parameter choices. (However, the large and highly variable delays in the actual Tor network do not matter here, since the watermark is being transmitted over a channel external Tor.) Prior to implementation, it would be necessary to do a survey of proxy mechanisms available for congestion attacks and tune the parameter choices appropriately; we leave this to future work.

CHAPTER 7

CONCLUSIONS

In this thesis, we investigated the problem of network flow watermarks. In short, this thesis shows that it is possible to design effective flow watermarks that aid traffic analysis for different application scenarios.

We proposed two novel flow watermarks: the non-blind scheme of RAINBOW and the blind scheme of SWIRL. In designing these watermarks we tried to achieve the following two objectives simultaneously: i) watermark robustness to network perturbations, and ii) watermark invisibility from non-watermarking entities. Previous research fails to achieve both of these properties at the same time and tends to sacrifice one for the other. For instance, several watermarks provide robustness to network perturbations by applying large-value delays to network flows, compromising the invisibility of the inserted watermark. The watermarks designed in this thesis ensure watermark invisibility through performing statistical tests that try to distinguish between watermarked flows and regular non-watermarked flows. Also, the designed watermarked are able to perform watermark detection after observing short lengths of network flows, providing real-time identification of linked flow.

Flow watermarking literature has only considered two applications: the use of flow watermarks by security defenders to detect stepping stone attacks, and its use by cyber criminals to compromise anonymity networks. In this thesis we show that it is possible to use flow watermarks in other applications where linking network flows is challenging due to traffic obfuscation. In particular, we propose two new applications for network flow watermarks.

Finally, this thesis also investigated flow fingerprinting, an underexplored, yet important, kind of flow tagging. In contrast to watermarking, flow fingerprinting aims at inserting *several* bits of information over network flows reliably. As a result, effective extraction of flow fingerprints is a more challenging problem than watermark detection. This thesis shows the possibility

of designing reliable flow fingerprints by utilizing coding techniques. I believe that more advances can be made in the realm of flow fingerprinting, which requires additional research in the future.

REFERENCES

[1] S. Staniford-Chen and L. T. Heberlein, "Holding intruders accountable on the Internet," in *IEEE Symposium on Security and Privacy*, C. Meadows and J. McHugh, Eds. IEEE Computer Society Press, May 1995, pp. 39–49.

[2] K. Yoda and H. Etoh, "Finding a connection chain for tracing intruders," in *European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, F. Cuppens, Y. Deswarte, D. Gollmann, and M. Waidner, Eds., vol. 1895. Springer, Oct. 2000, pp. 191–205.

[3] Y. Zhang and V. Paxson, "Detecting stepping stones," in *USENIX Security Symposium*, S. Bellovin and G. Rose, Eds. Berkeley, CA, USA: USENIX Association, Aug. 2000, pp. 171–184.

[4] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an analysis of onion routing security," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, July 2000, pp. 96–114.

[5] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, M. Blaze, Ed. Berkeley, CA, USA: USENIX Association, 2004.

[6] X. Wang, D. Reeves, and S. F. Wu, "Inter-packet delay based correlation for tracing encrypted connections through stepping stones," in *European Symposium on Research in Computer Security*, ser. Lecture Notes in Computer Science, D. Gollmann, G. Karjoth, and M. Waidner, Eds., vol. 2502. Springer, Oct. 2002, pp. 244–263.

[7] D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford, "Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay," in *International Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, A. Wespi, G. Vigna, and L. Deri, Eds., vol. 2516. Springer, Oct. 2002, pp. 16–18.

[8] A. Blum, D. X. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *International Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Jonsson, A. Valdes, and M. Almgren, Eds., vol. 3224. Springer, Sep. 2004, pp. 258–277.

[9] X. Wang and D. S. Reeves, "Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays," in *ACM Conference on Computer and Communications Security*, V. Atluri, Ed. New York, NY, USA: ACM, 2003, pp. 20–29.

[10] Y. Pyun, Y. Park, X. Wang, D. S. Reeves, and P. Ning, "Tracing traffic through intermediate hosts that repacketize flows," in *IEEE Conference on Computer Communications (INFOCOM)*, G. Kesidis, E. Modiano, and R. Srikant, Eds., May 2007, pp. 634–642.

[11] A. Houmansadr, N. Kiyavash, and N. Borisov, "RAINBOW: A Robust And Invisible Non-Blind Watermark for Network Flows," in *Network and Distributed System Security Symposium*, Feb. 2009.

[12] X. Wang, S. Chen, and S. Jajodia, "Tracking anonymous peer-to-peer VoIP calls on the Internet," in *ACM Conference on Computer and Communications Security*, C. Meadows, Ed. New York, NY, USA: ACM, Nov. 2005, pp. 81–91.

[13] X. Wang, S. Chen, and S. Jajodia, "Network flow watermarking attack on low-latency anonymous communication systems," in *IEEE Symposium on Security and Privacy*, 2007, pp. 116–130.

[14] W. Yu, X. Fu, S. Graham, D.Xuan, and W. Zhao, "DSSS-based flow marking technique for invisible traceback," in *IEEE Symposium on Security and Privacy*, 2007, pp. 18–32.

[15] D. Ramsbrock, X. Wang, and X. Jiang, "A first step towards live botmaster traceback," in *RAID*, ser. Lecture Notes in Computer Science, R. Lippmann, E. Kirda, and A. Trachtenberg, Eds., vol. 5230. Springer, 2008, pp. 59–77.

[16] A. Houmansadr and N. Borisov, "Botmosaic: Collaborative network watermark for botnet detection," Computing Research Repository arXiv, Tech. Rep. arXiv:1203.1568v1, Mar. 2012.

[17] T. Ylonen and C. Lonvick, "The secure shell (SSH) protocol architecture," RFC 4251, Jan. 2006.

[18] J. Boyan, "The anonymizer: Protecting user privacy on the web," *Computer-Mediated Communication Magazine*, vol. 4, no. 9, September 1997.

[19] P. Syverson, G. Tsudik, M. Reed, and C. Landwehr, "Towards an analysis of onion routing security," in *Proceedings of Designing Privacy Enhancing Technologies: Workshop on Design Issues in Anonymity and Unobservability*, H. Federrath, Ed. Springer-Verlag, LNCS 2009, July 2000, pp. 96–114.

[20] A. Back, I. Goldberg, and A. Shostack, "Freedom systems 2.1 security issues and analysis," Zero Knowledge Systems, Inc., White Paper, May 2001.

[21] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," in *USENIX Security Symposium*, M. Blaze, Ed. Berkeley, CA, USA: USENIX Association, 2004.

[22] T. He and L. Tong, "Detecting encrypted stepping-stone connections," *IEEE Transactions on Signal Processing*, vol. 55, no. 5, pp. 1612–1623, May 2007.

[23] W. Dai, "PipeNet," http://freehaven.net/anonbib/cache/pipenet10.html, Tech. Rep., 1998.

[24] A. Back, U. Möller, and A. Stiglic, "Traffic analysis attacks and trade-offs in anonymity providing systems," in *Information Hiding*, ser. Lecture Notes in Computer Science, vol. 2137. Springer, 2001, pp. 245–247.

[25] J.-F. Raymond, "Traffic analysis: protocols, attacks, design issues, and open problems," in *International Workshop on Designing Privacy Enhancing Technologies*. Springer, 2001, pp. 10–29.

[26] D. Chaum, "Untraceable electronic mail, return addresses, and digital pseudonyms," *Communications of the ACM*, vol. 4, no. 2, February 1981.

[27] G. Danezis, "The traffic analysis of continuous-time mixes," in *Workshop on Privacy Enhancing Technologies*, ser. Lecture Notes in Computer Science, D. Martin and A. Serjantov, Eds., vol. 3424. Springer, May 2004, pp. 35–50.

[28] B. Coskun and N. Memon, "Online Sketching of Network Flows for Real-Time Stepping-Stone Detection," in *2009 Annual Computer Security Applications Conference*. Ieee, Dec. 2009. [Online]. Available: http://www.acsa-admin.org/2009/openconf/modules/request.php?module=oc_program\&action=view.php\&id=122http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5380704 pp. 473–483.

[29] P. Peng, P. Ning, and D. S. Reeves, "On the secrecy of timing-based active watermarking trace-back techniques," in *IEEE Symposium on Security and Privacy*, V. Paxson and B. Pfitzmann, Eds. IEEE Computer Society Press, May 2006, pp. 334–349.

[30] W. Jia, F. P. Tso, Z. Ling, X. Fu, D. Xuan, and W. Yu, "Blind detection of spread spectrum flow watermarks," in *INFOCOM*. IEEE, 2009. [Online]. Available: http://dx.doi.org/10.1109/INFCOM.2009.5062144 pp. 2195–2203.

[31] N. Kiyavash, A. Houmansadr, and N. Borisov, "Multi-flow attacks against network flow watermarking schemes," in *USENIX Security Symposium*, P. van Oorschot, Ed. Berkeley, CA, USA: USENIX Association, 2008.

[32] J. Huang, X. Pan, X. Fu, and J. Wang, "Long PN Code Based DSSS Watermarking," in *31st Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2011, pp. 2426 –2434.

[33] W. Fischer and K. Meier-Hellstern, "The Markov-modulated Poisson process (MMPP) cookbook," *Performance Evaluation*, vol. 18, no. 2, pp. 149–171, 1993.

[34] R. G. Gallager, *Discrete Stochastic Processes*. Kluwer Academic Publishers, 1996.

[35] E. N. Brown, R. Barbieri, V. Ventura, R. E. Kass, and L. M. Frank, "The time-rescaling theorem and its application to neural spike train data analysis," *Neural Computation*, vol. 14, no. 2, pp. 325–346, 2002.

[36] V. Paxson and S. Floyd, "Wide-area traffic: The failure of Poisson modeling," *IEEE/ACM Transactions on Networking*, vol. 3, no. 3, pp. 226–244, June 1995.

[37] "Ask Dr. Math: Consecutive failures in Bernoulli trials," July 1999, The Math Forum @ Drexel. [Online]. Available: http://mathforum.org/library/drmath/view/56637.html

[38] R. Merris, *Combinatorics*, 2nd ed., ser. Wiley series in discrete mathematics and optimization. Hoboken, NJ: John Wiley and Sons, 2003.

[39] 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," 2004, pp. 253–266.

[40] I. Marsh and F. Li, "Wide area measurements of VoIP quality," in *Workshop on Quality of Future Internet Services*, 2003.

[41] A. Houmansadr, N. Kiyavash, and N. Borisov, "Multi-flow attack resistant watermarks for network flows," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009, pp. 1497 –1500.

[42] A. Houmansadr and N. Borisov, "SWIRL: A scalable watermark to detect correlated network flows," in *Proceedings of the Network and Distributed System Security Symposium, NDSS'11*, 2011.

[43] A. K. Goteti and P. Moulin, "QIM watermarking games," in *ICIP*, 2004, pp. 717–720.

[44] W. Bender, D. Gruhl, N. Morimoto, and A.Lu, "Techniques for data hiding." *IBM Systems Journal*, vol. 35, no. 3/4, pp. 313–336, 1996.

[45] I. J. Cox, J. Killian, T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for images, audio, and video," in *IEEE International Conference on Image Processing (ICIP)*, vol. III, 1996, pp. 243–246.

[46] J. Kilian, F. Leighton, L. Matheson, T. Shamoon, R. Tarjan, and F. Zane, "Resistance of digital watermarks to collusive attacks," in *IEEE International Symposium on Information Theory*, 1998, p. 271.

[47] B. N. Levine, M. K. Reiter, C. Wang, and M. Wright, "Timing attacks in low-latency mix systems," in *Financial Cryptography*, ser. Lecture Notes in Computer Science, A. Juels, Ed., vol. 3110. Springer, Feb. 2004, pp. 251–265.

[48] H. V. Poor, *An Introduction to Signal Detection and Estimation*. Springer-Verlag, 1998.

[49] 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 *Symposium on Networked Systems Design and Implementation*, R. Morris and S. Savage, Eds. USENIX, Mar. 2004, pp. 253–266.

[50] S. Gianvecchio and H. Wang, "Detecting covert timing channels: an entropy-based approach," in *ACM Conference on Computer and Communications Security*, P. Ning, S. D. C. di Vimercati, and P. F. Syverson, Eds. ACM, 2007. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315284 pp. 307–316.

[51] S. Kotz, T. Kozubowski, and K. Podgórski, *The Laplace Distribution and Generalizations: A Revisit with Applications to Communications, Economics, Engineering, and Finance*, ser. Progress in Mathematics Series. Birkhäuser, 2001. [Online]. Available: http://books.google.com/books?id=cb8B07hwULUC

[52] H. Chernoff, "A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations," *Annals of Mathematical Statistics*, vol. 23, pp. 493–507, 1952.

[53] C. Walsworth, E. Aben, kc claffy, and D. Andersen, "The CAIDA anonymized 2009 Internet traces—January," http://www.caida.org/data/passive/passive_2009_dataset.xml, Mar. 2009.

[54] I. Ezzeddine and P. Moulin, "Achievable rates for queue-based timing stegocodes," in *2009 IEEE Information Theory Workshop*. IEEE, 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5351197 pp. 379–383.

[55] V. Anantharam and S. Verdu, "Bits through queues," *IEEETIT: IEEE Transactions on Information Theory*, vol. 42, 1996.

[56] C. E. Shannon, "Channels with side information at the transmitter," *IBM Journal of Research and Development*, vol. 2, no. 4, pp. 289–293, 1958.

[57] A. K. Gupta and S. Nadarajah, *Handbook of Beta Distribution and Its Applications*. CRC Press, 2004.

[58] C. Walsworth, E. Aben, K. C. Claffy, and D. Andersen, "The CAIDA anonymized 2009 Internet traces—January," http://www.caida.org/data/passive/passive_2009_dataset.xml, Mar. 2009.

[59] Z. Ling, J. Luo, W. Yu, X. Fu, D. Xuan, and W. Jia, "A new cell counter based attack against Tor." New York, New York, USA: ACM Press, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1653662.1653732 p. 578.

[60] R. Dingledine and N. Mathewson, "Tor Protocol Specification," https://gitweb.torproject.org/torspec.git?a=blob_plain;hb=HEAD;f=tor-spec.txt.

[61] J. H. van Lint, *Introduction to Coding Theory (third edition)*. Berlin (D): Springer Verlag, 1998.

[62] S. Benedetto and E. Biglieri, *Principles of Digital Transmission: With Wireless Applications*, ser. Information Technology: Transmission, Processing, and Storage. Kluwer Academic/Plenum Press, 1999. [Online]. Available: http://books.google.com/books?id=thfj9S8cbdoC

[63] S. Dolinar, D. Divsalar, and F. Pollara, "Code performance as a function of block size," TMO Progress, Tech. Rep., 1998. [Online]. Available: http://tmo.jpl.nasa.gov/progress_report/42-133/133K.pdf

[64] "The Coded Modulation Library (CML)," http://www.iterativesolutions.com/Matlab.htm.

[65] D. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Sep. 2003.

[66] A. Ramachandran and N. Feamster, "Understanding the network-level behavior of spammers," in *SIGCOMM*, L. Rizzo, T. E. Anderson, and N. McKeown, Eds. ACM, 2006, pp. 291–302.

[67] L. Kharouni, "SDBOT IRC botnet continues to make waves," Trend Micro Threat Research, White Paper, December 2009.

[68] J. Zhuge, T. Holz, X. Han, J. Guo, and W. Zou, "Characterizing the IRC-based botnet phenomenon," Department for Mathematics and Computer Science, University of Mannheim, Tech. Rep. TR-2007-010, 2007.

[69] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*. Berkeley, CA, USA: USENIX Association, 2006, pp. 7–7.

[70] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using dnsbl counter-intelligence," in *SRUTI'06: Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*. Berkeley, CA, USA: USENIX Association, 2006, pp. 8–8.

[71] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[72] M. P. Collins, T. J. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, "Using uncleanliness to predict future botnet addresses," in *IMC '07: Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*. New York, NY, USA: ACM, 2007, pp. 93–104.

[73] R. Villamarín-Salomón and J. C. Brustoloni, "Bayesian bot detection based on dns traffic similarity," in *SAC '09: Proceedings of the 2009 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2009, pp. 2035–2041.

[74] W. Zilong, W. Jinsong, H. Wenyi, and X. Chengyi, "The detection of IRC botnet based on abnormal behavior," in *Multimedia and Information Technology (MMIT), 2010 Second International Conference on*, vol. 2, april 2010, pp. 146 –149.

[75] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by IRC nickname evaluation," in *First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[76] J. Oikarinen and D. Reed, "RFC 1459: Internet Relay Chat Protocol," May 1993, status: EXPERIMENTAL. [Online]. Available: ftp://ftp.internic.net/rfc/rfc1459.txt,ftp://ftp.math.utah.edu/pub/rfc/rfc1459.txt

[77] C. Kalt, "Internet Relay Chat: Server Protocol," RFC 2813 (Informational), Apr. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2813.txt

[78] L. Spitzner, "The Honeynet Project: trapping the hackers," *Security & Privacy Magazine, IEEE*, vol. 1, no. 2, pp. 15–23, 2003.

[79] N. Provos, "A virtual honeypot framework," in *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2004.

[80] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[81] S. Nagaraja, A. Houmansadr, P. Agarwal, V. Kumar, P. Piyawongwisal, and N. Borisov, "Stegobot: A covert social network botnet," in *13th Information Hiding Conference (IH)*, ser. Lecture Notes in Computer Science, A. Ker, S. Kraver, and T. Filler, Eds. Heidelberg, Berlin: Springer, May 2011.

[82] N. Evans, R. Dingledine, and C. Grothoff, "A practical congestion attack on Tor using long paths," in *USENIX Security Symposium*, F. Monrose, Ed. USENIX, Aug. 2009, pp. 33–50.

[83] S. Murdoch and G. Danezis, "Low-cost traffic analysis of Tor," in *IEEE Symposium on Security and Privacy*, V. Paxson and M. Waidner, Eds. IEEE Computer Society, May 2005.

[84] R. Wendolsky, D. Herrmann, and H. Federrath, "Performance comparison of low-latency anonymisation services from a user perspective," in *Privacy Enhancing Technologies Symposium*, ser. Lecture Notes in Computer Science, N. Borisov and P. Golle, Eds., vol. 4776. Springer, June 2007, pp. 233–253.