On the Secrecy of Timing-Based Active Watermarking Trace-Back Techniques*

Pai Peng Peng Ning Douglas S. Reeves
Cyber Defense Laboratory
Department of Computer Science
North Carolina State University
Raleigh, NC 27695, USA

Abstract

Timing-based active watermarking schemes are developed to trace back attackers through stepping stone connections or anonymizing networks. By slightly changing packet timing, these schemes achieve robust correlation for encrypted network connections under timing perturbation. However, the manipulation on packet timing makes the schemes themselves a potential target of intelligent attackers. In this paper, we analyze the secrecy of the timing-based active watermarking techniques for tracing through stepping stones, and propose an attack scheme based on analyzing the packet delays between adjacent stepping stones. We develop attack techniques to infer important watermark parameters, and to recover and duplicate embedded watermarks. The resulting techniques enable an attacker to defeat the tracing systems in certain cases by removing watermarks from the stepping stone connections, or replicating watermarks in non-stepping stone connections. We also develop techniques to determine in real-time whether a stepping stone connection is being watermarked for trace-back purposes. We have performed substantial experiments using real-world data to evaluate these techniques. The experimental results demonstrate that for the watermark scheme being attacked (1) embedded watermarks can be successfully recovered and duplicated when the watermark parameters are not chosen carefully, and (2) the existence of watermarks in a network flow can always be quickly detected.

1. Introduction

The rapid growth of the Internet not only brings the global connectivity to the general public, but also offers malicious users an opportunity to hide their traces. In particular, a malicious user may attack other computers from anywhere on the Internet through, for example, a sequence of *stepping stones* (i.e., compromised hosts used as intermediate steps to launch the final attack) or an anonymizing network such as Tor [4] and FindNot [7]. As a result, it is challenging to identify the sources of attacks and hold attackers accountable for their malicious actions.

Various network forensic techniques have been proposed in the past decade to address this issue. Several approaches have been proposed to identify the origin of an attack launched through a sequence of stepping stones (e.g., [1,5,18,24,30,35]). Moreover, substantial results have been obtained for IP trace-back aimed at locating the sources of Distributed Denial of Service (DDoS) attacks with spoofed source IP addresses (e.g., [21–23]). Recently researchers have also been investigating how to correlate traffic through anonymizing networks [9, 13, 16, 27, 32].

Among these network forensic techniques, timing-based active watermarking is one of the promising approaches aimed at tracing back to the attacker's origin through a sequence of stepping stones [28] or an anonymizing network [27]. These approaches embed a timing-based watermark into a network flow by delaying selected packets. Trace-back is achieved by embedding/decoding watermarks in network flows and correlating the flows with similar watermarks. Because these techniques only use packet timing for trace-back purposes, they can handle encryption due to secure protocols such as SSH and IPsec. Compared with timing-based passive approaches (e.g., [1, 5, 34, 35]), which correlate network flows simply using packet timing (without alteration), the active watermark schemes in general require fewer packets and thus can identify the sources of attacks more efficiently. It has been shown in [27, 28] that such techniques are robust to timing perturbation injected by attackers and can successfully defeat anonymizing networks such as FindNot [7].

However, research on timing-based active watermarking has overlooked an important issue: the secrecy of the parameters used in watermarking. There are two immediate implications if an attacker knows these parameters. First,

*The work described in this paper has been supported by ARDA under contract NBCHC030142. The contents of this paper do not necessarily reflect the position or the policies of the U.S. Government.

the attacker may attempt to remove the watermark, thus rendering the trace-back schemes ineffective. Second, the attacker may replicate the watermark in other network flows, so that benign users will be held accountable for the attacker's malicious activities. The previous work [27, 28] relies on the assumption that these watermark parameters are kept only to the trace-back system. However, it is desirable to investigate if an attacker can detect and recover the parameters from the network flows when he/she is being traced.

The objective of this paper is to investigate how an attacker being traced by a timing-based active watermarking system can detect and recover the watermark parameters, using the technique proposed in [28] as the target. We assume the attacker uses nothing but the packet timestamps on the stepping stone hosts. We study how such an attacker can achieve two complementary goals: (1) How to recover the watermark parameters or replicate the watermark to a different network flow? (2) How to quickly determine whether the attacker is being traced by a timing-based active watermarking scheme?

In this paper, we propose an attack scheme that compromises timing-based active watermarking trace-back systems by analyzing the packet delays between adjacent stepping stones. We develop a suite of algorithms to infer watermark parameters, recover/duplicate watermarks, and detect the existence of watermarks as early as possible. We also investigate the trade-off between watermark capability (i.e., detection rate & false positive rate) and watermark secrecy, and demonstrate that a watermark cannot evade detection without degradation of its trace-back capability. Our experimental results have confirmed that almost the entire watermark can be recovered or duplicated if the watermark parameters are not selected cautiously. Although our attack focuses on a specific watermarking scheme, it can potentially be extended to compromise other timing-based active watermarking approaches. Our results indicate that the threats of intelligent attackers must be carefully considered for any active trace-back scheme that manipulates packet timing.

The rest of this paper is organized as follows. Section 2 gives a brief introduction to the timing-based active watermarking scheme proposed in [28], which is the main target of our attack. Section 3 presents an overview of the proposed attack. Section 4 discusses our approach for inferring unknown watermark parameters and recovering/duplicating watermark. Section 5 describes our approach for quickly detecting the existence of (unknown) watermarks. Section 6 presents our experiments used to validate the proposed approaches. Section 7 discusses related work, and Section 8 concludes our paper and points out some future research directions.

2. Timing-Based Active Watermarking for Tracing through Stepping Stones [28]

In this section, we briefly describe the timing-based active watermarking scheme proposed in [28], which will be the target of our analysis in later sections.

When using a sequence of stepping stone hosts, an attacker needs to establish a connection between any two adjacent hosts in this sequence to have a chain of connections, so that the attacker's commands can be relayed to the remote host by these connections and the responses can be relayed back to the attacker. The timing-based active watermarking approach in [28] embeds watermarks in upstream connections (flows) and attempts to detect them in downstream connections (flows) in order to trace back to the attacker's origin.

A watermark is embedded through manipulating the *inter-packet delays* (IPDs) of certain pre-selected watermark embedding packets. The IPD between two packets p_a and p_b is $ipd_{(a,b)} = t_b - t_a$, where p_b is transmitted later than p_a, and t_a and t_b are the timestamps of p_a and p_b, respectively. IPDs are quantized for robustness; given a *quantization step* S, the quantization function $q(ipd, S)$ rounds off ipd/S to the nearest integer. To embed one watermark bit w (0 or 1), an ipd is slightly increased (by delaying the second packet the smallest amount) so that the watermarked IPD, denoted as ipd^W, satisfies the condition $q(ipd^W, S) \bmod 2 = w$. This means ipd^W is even multiples of S when 0 is embedded, and odd multiples of S when 1 is embedded.

Watermarked IPD ipd^W is computed by the watermark embedding function: $e(ipd, w, S) = [q(ipd + S/2, S) + \delta] \times S$, where $\delta = (w - (q(ipd + S/2, S) \bmod 2) + 2) \bmod 2$. This approach quantizes $(ipd + S/2)$ instead of ipd to ensure $ipd^W \geq ipd$, so that the watermark bit can be embedded by delaying the second packet involved in the IPD by the amount of $(ipd^W - ipd)$. Hereafter, a delay caused by watermark embedding is called as a *watermark delay*.

The watermark decoding function is $d(ipd, S) = q(ipd, S) \bmod 2$. When timing perturbation (due to attacks or network delays) is introduced after watermarking, as long as the change on ipd^W is limited by $(-S/2, S/2]$, this function can decode the watermark bit correctly. Therefore, a watermark bit embedded in a single IPD can resist up to $S/2$ random timing perturbation. To resist perturbations larger than $S/2$, M $(M > 1)$ IPDs are used to embed one bit. Specifically, the average of the M IPDs is computed as $ipd_{avg} = \frac{1}{M}\sum_{i=1}^{M} ipd_i$, and then watermarked IPD average $ipd^W_{avg} = e(ipd_{avg}, w, S)$ is calculated. The watermark bit is embedded by increasing each of these M IPDs by the amount of $(ipd^W_{avg} - ipd_{avg})$. Decoding the watermark bit on the M IPDs simply involves computing $d(ipd^W_{avg}, S)$.

With the same S, embedding 1 bit watermark with multiple IPDs provides higher resistance to random timing perturbation than a single IPD.

This approach uses multiple watermark bits to further increase the detection rate and reduce the false positive rate for trace-back. An L-bit watermark W is embedded by repeating the procedure of embedding a single bit L times. During watermark detection, another L-bit watermark W' is decoded from the suspicious flow and compared with W. If the hamming distance between W and W' is less than or equal to a pre-defined threshold h, this approach reports that a stepping-stone flow is detected. It has been shown in [28] that this approach is highly robust against random timing perturbation.

Table 1 summarizes the parameters used in this approach.

Table 1. Watermark Parameters

S	Quantization step
L	The number of binary bits in the watermark
M	Degree of robustness (i.e., the number of IPDs used to embed 1 bit)
$w_1 \dots w_L$	The watermark, in which each bit w_i is either 0 or 1

3. Basis of the Timing Analysis Attack

As discussed earlier, the secrecy of the watermark parameters has not been seriously investigated. If an attacker is able to derived these parameters, the attacker can either remove the watermark (i.e., reduce the *detection rate*) or duplicate the watermark in benign flows (i.e., increase the *false positive rate*). In both cases, the attacker can successfully defeat the tracing system. In this paper, we take an attacker's position, aiming at understanding how well an attacker can derive the watermark parameters used by timing-based active watermarking through timing analysis.

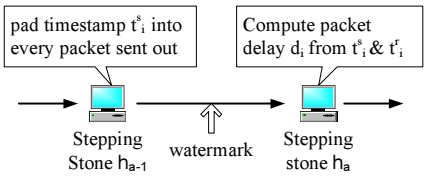


Figure 1. Overview of our attacks to the watermark scheme

As discussed in Section 2, when an attacker uses a series of stepping stone hosts, the attacker has to establish a sequence of connections between adjacent hosts, and application data (e.g., a shell command) is relayed by these connections from the attacker to the final target or vice versa.

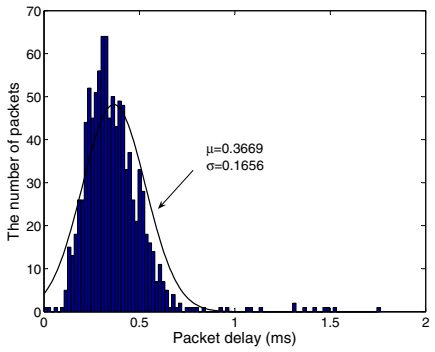


Figure 2. The distribution of packet delays after the clock skew is removed

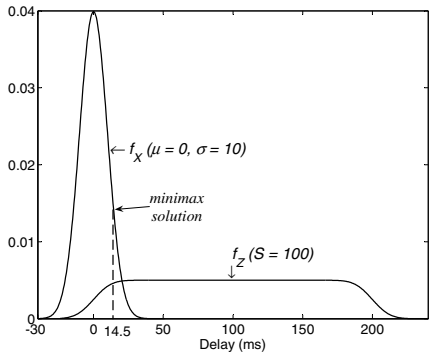


Figure 3. Probability density for normal and the sum of normal and uniform distributions

An attacker can easily obtain the *one-way packet transit delay* (abbreviated as *packet delay*) of a piece of application data as it is forwarded from one host to another. For example, as illustrated in Figure 1, when h_{a-1} forwards a shell command to h_a in a packet, the attacker can include h_{a-1}'s current local time t^s along with the command. When h_a receives this shell command, the attacker can retrieve t^s, check the receipt time t^r at h_a, and calculate the delay as $d = t^r - t^s$. The packet delays are the target of our timing analysis.

When the stepping stone hosts do not have well synchronized time, the packet delay will include the clock offset (i.e., the difference between the clocks at a specific time) and the clock skew (i.e., the first derivative of the offset). Clock skew is a critical issue since it constantly changes the packet delays. We may use the approach proposed in [17] to handle this problem. That is, we use cumulative minima (or maxima) to identify the skew, and then use linear fit to compute and remove the skew. As a result, clock discrepancy only introduces a constant clock offset into all packet delays after the clock skews are removed.

We examined the one-way packet transit delays between computers in the PlanetLab [19], which are distributed

world wide. Figure 2 shows a typical distribution of the packet delays from a computer at MIT to a computer on our campus network after the clock skew between these computers are removed. To simplify our analysis, we approximate such distributions with normal distributions. In our experiments, such approximations pass Kolmogorov-Smirnov [6] goodness-of-fit test with a *significance level* of 0.05, which is the probability that we wrongly reject the normal distribution approximation when it is actually true. It is easy to see that an attacker can observe the packet delays and estimate the parameters (i.e., mean μ and variance σ^2) of the delay distribution. To distinguish such packet delays from the delays introduced by watermark embedding, hereafter we call them *normal network delays*.

When a timing-based active watermarking approach is used for trace-back, certain packets will have to be delayed to embed the watermark. Assuming the packets to be delayed come at random time, we can easily derive that the *watermark delays* (i.e., additional delays of the embedding packets) follow a uniform distribution over $[0, 2S)$, where S is the quantization step. We performed experiments to validate this assumption. The results pass Kolmogorov-Smirnov goodness-of-fit test for uniform distribution with a significance level of 0.05.

When a watermark is embedded, the packet delay of each embedding packet should be the combination of the normal network delay and the watermark delay. Let normal random variable X represent the normal network delay, and uniform random variable Y represent the watermark delay. The delay of the embedding packet is $Z = X + Y$. We can easily derive the probability density function of Z as:

$$\begin{aligned} f_Z(x) &= \int_{-\infty}^{+\infty} f_X(y) f_Y(x-y) dy \\ &= \frac{1}{2\sigma S\sqrt{2\pi}} \int_{x-2S}^{x} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy, \end{aligned} \quad (1)$$

where f_X is the probability density of a normal distribution with mean μ and variance σ^2. S is the quantization step. The mean μ is not very important since it does not affect the shapes of f_X and f_Z, but only moves them along x-axis simultaneously. Figure 3 shows an example of f_X and f_Y.

Apparently, when a network flow is watermarked by a trace-back system, some packet delays will follow the distribution of Z, which is different from that of X. (A packet p_i delayed by watermark may cause some following packets to be postponed and sent out immediately after p_i to keep the correct packet order. These collateral delays can be identified by checking whether there is a large delay that affects its following packets. For simplicity, we do not consider such collateral delays.) In this paper, we investigate techniques that can take advantage of this observation to detect the existence of timing-based active watermarks, recover the watermark parameters, and remove and duplicate the observed watermarks. In order to fully understand attackers' threats on the active watermarking scheme, we focus on investigating the following problems:

1. How can an attacker infer the watermark parameters and how much can be recovered?
2. How can an attacker duplicate a watermark to mislead the trace-back system? How well?
3. How can an attacker that connects through stepping stones determine whether he/she is being traced by a timing-based active watermarking system as quickly as possible?

We choose to first tackle the watermark recovery/duplication problem since it puts a major threat on the watermark scheme. In the following, we assume the attacker obtain the packet delays from two adjacent hosts in the stepping stone connections, and try to compromise the watermark scheme at the second host.

4. Inferring Watermark Parameters

As discussed earlier, an attacker can observe the packet delays and obtain the distribution (i.e., μ and σ). We also assume the attacker has obtained a sequence of packet delays $d_1, d_2, ..., d_n$ between two stepping stone hosts where a watermark is embedded. However, the attacker does not know the watermark parameters, including the quantization step S, the degree of robustness M, the length L of the watermark, and the exact watermark bits. The goal of this section is to investigate whether, how, and how well the attacker can recover these parameters.

When there is no watermark, the observed packet delays are entirely caused by normal network delays. However, when an watermark exists, some delays will be the combination of both normal network delays and watermark delays. That is, the observed packet delays are drawn from a mixture of two random variables X and Z. Thus, the distribution of d_i's is

$$f(x, \theta) = (1-\theta) f_X(x) + \theta f_Z(x), \quad (2)$$

where θ is the proportion of d_i's that are from watermark delays. When no watermark is embedded, $\theta = 0$.

In the following, we first estimate the quantization step S and the proportion parameter θ, then identify the packets delayed due to watermark, and finally recover the remaining watermark parameters or duplicate the watermark (without knowing all the parameters).

4.1. Estimating the Quantization Step S and the Proportion Parameter θ

We propose to use Expectation Maximization (EM) algorithm [3] to estimate the quantization step S and the pro-

portion parameter θ from a sequence of observed packet delays. The EM algorithm is an iterative optimization method to find the maximum likelihood estimation of parameters in probability densities when there is unobservable or missing data. It is also widely used to estimate the parameters and proportions where different probability densities are mixed together.

Let $\Psi = (S, \theta)^T$ be the vector of unknown parameters we want to estimate. Given the vector of n observed packet delays $\mathbf{d} = (d_1, \ldots, d_n)^T$, the likelihood function for Ψ is $\mathcal{L}(\Psi) = \prod_{j=1}^{n} f(d_j|\Psi)$. The EM algorithm estimates Ψ by finding the value that can maximize the likelihood $\mathcal{L}(\Psi)$. This can be done by solving the equation $\partial\mathcal{L}(\Psi)/\partial\Psi = 0$, or equivalently, $\partial \log \mathcal{L}(\Psi)/\partial\Psi = 0$, where $\log \mathcal{L}(\Psi) = \sum_{j=1}^{n} \log\big((1-\theta)f_X(d_j) + \theta f_Z(d_j)\big)$. However, it is in general difficult to directly solve such an equation.

In order to utilize the EM algorithm to estimate parameters in a mixture of two probability densities, we introduce additional parameters $\mathbf{z} = (z_1, \ldots, z_n)^T$, where z_j is 0 (or 1) indicating that d_j is from the distribution f_X (or f_Z). (Note that the values in $\mathbf{z}$ cannot be observed.) By including $\mathbf{z}$, the log likelihood for Ψ is transformed into

$$\begin{aligned} \log \mathcal{L}_c(\Psi) &= \sum_{j=1}^{n}(1-z_j)\log f_X(d_j) + \sum_{j=1}^{n} z_j \log f_Z(d_j) \\ &+ \sum_{j=1}^{n}(1-z_j)\log(1-\theta) + \sum_{j=1}^{n} z_j \log\theta. \quad (3) \end{aligned}$$

We call $\log \mathcal{L}_c(\Psi)$ the complete-data log likelihood.

The general procedure of the EM algorithm begins with an arbitrary initial value $\Psi = \Psi^{(0)}$. In round $i+1$, where $i = 0, 1, 2, ...$, the algorithm first performs the E-step to calculate the expectation of the log likelihood $Q(\Psi|\Psi^{(i)}) = E_{\Psi^{(i)}}\big\{\log \mathcal{L}_c(\Psi) \mid \mathbf{d}\big\}$. Then the algorithm performs the M-step to maximize $Q(\Psi|\Psi^{(i)})$ with respect to Ψ, that is, find $\Psi^{(i+1)}$ such that $Q(\Psi^{(i+1)}|\Psi^{(i)}) \geq Q(\Psi^{i}|\Psi^{(i)})$ for all possible values of Ψ. It terminates when the difference between $\mathcal{L}(\Psi^{(i+1)}) - \mathcal{L}(\Psi^{(i)})$ is small enough.

In our case, the E-step simply computes the current conditional expectation of $\mathbf{z}$ using the observed packet delays $\mathbf{d}$. The i-th round value of $\mathbf{z}^{(i)}$ is $z_j^{(i)} = \theta^{(i)} f_X(d_j)/f(d_j|\Psi^{(i)}), 1 \leq j \leq n$. With the value of $\mathbf{z}^{(i)}$, we can easily compute $Q(\Psi|\Psi^{(i)})$. However, in the M-step, it is difficult to globally maximize $Q(\Psi|\Psi^{(i)})$ due to the probability density f_Z. To deal with this problem, we use the generalized EM algorithm (GEM) [15], in which the M-step is modified to find $\Psi^{(i+1)}$ such that $Q(\Psi^{(i+1)}|\Psi^{(i)}) \geq Q(\Psi^{(i)}|\Psi^{(i)})$. More specifically, we use the GEM algorithm based on one Newton-Raphson step [20] in the M-step to compute $\Psi^{(i+1)}$ by $\Psi^{(i+1)} = \Psi^{(i)} + \delta^{(i)}$, where $\delta^{(i)} = -[\partial^2 Q(\Psi|\Psi^{(i)})/\partial\Psi\partial\Psi^T]^{-1}_{\Psi=\Psi^{(i)}}[\partial Q(\Psi|\Psi^{(i)})/\partial\Psi]_{\Psi=\Psi^{(i)}}$. (Note that this is the first iteration of the Newton-Raphson method when computing a root for equation $\partial Q(\Psi|\Psi^{(i)})/\partial\Psi = 0$.) In our experiment, the initial value $\theta^{(0)}$ is set to 0.5 so that the algorithm can converge quickly for both bigger and smaller θ. For this $\theta^{(0)}$, the average packet delay is $2\mu + S$. Therefore we simply use the average value of the packet delays $\mathbf{d}$ as the initial value $S^{(0)}$ since μ is small compared with S in practice.

4.2. Identifying Watermark Delayed Packets

We can determine the probability densities f_Z and f with the estimated S and θ. In this subsection, we decide for each packet whether it has been delayed by the watermark encoder or not. We adopt the *Bayes decision rule* [6] in our decision process to minimize the cost of wrong decisions. Let λ_{ij} be the loss incurred for deciding i when the true state is actually j. In our case, the values of i and j are 1 (packets with only normal network delays) and 2 (packets with watermark delays), respectively. By using the Bayes decision rule, the expected loss (called *risk*) can then be minimized for our probability density functions and the losses λ_{ij} $(i, j = 1, 2)$. According to the Bayesian decision theory [6], we can decide a packet p_i as a watermark delayed packet if its packet delay d_i satisfies

$$\frac{f_Z(d_i)}{f_X(d_i)} \geq \frac{\lambda_{21} - \lambda_{11}}{\lambda_{12} - \lambda_{22}} \cdot \frac{1-\theta}{\theta}. \quad (4)$$

Here θ and $1-\theta$ can be seen as the priori probabilities for watermark delays and normal network delays, respectively. From inequality 4 we can numerically compute a threshold value $\bar{d}$. Packet p_i is identified as a watermark delayed packet when $d_i \geq \bar{d}$.

We may also develop a threshold that works for the same S but different θ, and thus avoid computing a new threshold for each θ when the same S is used for multiple flows. Specifically, we use the *minimax* [6] solution for inequality 4 to achieve good performance over all values of θ. Briefly speaking, the minimax solution searches for the decision threshold with which the maximum risk is minimized. Figure 3 gives an example of the minimax solution when *zero-one* loss is used (i.e., $\lambda_{11}, \lambda_{22} = 0$ and $\lambda_{12}, \lambda_{21} = 1$). The threshold is $\bar{d} = 14.5$ms, and the minimax risk is 0.074.

4.3. Watermark Recovery and Duplication

We have developed ways to infer the quantization step S and the watermark embedding packets. Now we further investigate how to recover and/or duplicate the entire watermark. We first consider several cases that watermarks can be embedded. In Section 4.3.6, we integrate these cases and describe the general approach.

- **Case I:** The watermark encoder uses one IPD to embed each watermark bit ($M = 1$). Moreover, it reuses the second packet in a previous IPD as the first packet in the next IPD in order to reduce the number of packets required to embed watermark. Thus, only $L+1$ packets are needed to embed an L-bit watermark.
- **Case II:** The watermark encoder still reuses packets to embed watermark, as in Case I. But multiple IPDs ($M > 1$) are used for each watermark bit to increase robustness. Totally $M \times (L + 1)$ packets are needed for an L-bit watermark. Obviously, Case I is a special case of Case II.
- **Case III:** The watermark encoder uses one IPD to embed each watermark bit ($M = 1$), but it does not reuse the same packet in more than 1 IPD. Moreover, the packets used to embed an earlier watermark bit do not interleave with those used to embed a later watermark bit. $2L$ packets are needed to embed an L-bit watermark.
- **Case IV:** The watermark encoder does not reuse the same packet in more than 1 IPD, and the packets used to embed an earlier watermark bit do not interleave with those used to embed a later watermark bit. However, it uses multiple IPDs to embed each watermark bit ($M > 1$) as in Case II. $2M \times L$ packets are needed to embed an L-bit watermark.
- **Case V:** Those not covered by the above four cases.

We will show that in the first two cases, we can recover most of the watermark embedded for trace-back purposes, and in Cases III and IV, we can duplicate the watermark in unrelated network flows with a high probability. We may still use the techniques developed for Case IV in Case V; however, the performance will drop quickly. The first four cases are used in the implementation (of the techniques in [28]) that we obtained from the Footfall project[1]. Indeed, in Case V, embedding watermark bits in different packets is not independent of each other due to the interleaved embedding packets, and will make the implementation more complicated.

4.3.1. Case I

We begin with the simplest case, where $L + 1$ embedding packets are needed to embed an L-bit watermark. Suppose the embedding packets are $p_{e_0}, \ldots, p_{e_L}$. Then $ipd_1 = t_{e_1} - t_{e_0}$ is used to embed bit w_1, $ipd_2 = t_{e_2} - t_{e_1}$ is used to embed bit w_2, and so on. Though by reusing the embedding packets we can reduce the number of packets needed, it also gives attackers an opportunity to break the watermark scheme.

Assume the quantization step S is estimated correctly using the approach in Section 4.1, and the embedding packets $p_{e_1}, \ldots, p_{e_L}$ have been identified using the approach discussed in Section 4.2. Thus, we can compute watermarked IPDs $ipd_2, \ldots, ipd_L$. Since each watermarked IPD must be a multiple S, they can be re-written as $a_2 \cdot S, \ldots, a_L \cdot S$, where $a_i (2 \le i \le L)$ are all integers. As discussed in Section 2, an even (or odd) a_i represents that w_i is 0 (or 1). Thus, we can recover the watermark embedded simply by dividing each ipd_i by S and checking the parity of the quotient. However, this approach has a limitation: An error in the estimated S may affect the recovery of all watermark bits.

In our work, we take a more robust approach that does not use S directly. Consider the greatest common divisor of $ipd_2, ipd_3, ..., ipd_L$. It is easy to see that $\text{GCD}(ipd_2, \ldots, ipd_L) = y \cdot S$, where $y = \text{GCD}(a_2, \ldots, a_L)$. Suppose watermark bits $w_2, \ldots, w_L$ are not all 0's. Thus, $a_2, \ldots, a_L$ cannot all be even, and y must be an odd number. Now we divide the IPDs by $\text{GCD}(ipd_2, \ldots, ipd_L)$ to get

$$a'_i = \frac{ipd_i}{\text{GCD(IPDs)}} = \frac{a_i \cdot S}{y \cdot S} = \frac{a_i}{y}, \text{ for } 2 \le i \le L. \quad (5)$$

Since y is an odd number, the parity of a'_i must be the same as that of a_i. However, when $w_2, \ldots, w_L$ are all 0's, this approach will generate an incorrect result. Nevertheless, this approach has already reduced the number of possible watermarks from 2^L to 4. As a result, the attacker can easily mislead the tracing system once for every four trials.

The problem is more complicated in practice. First, due to false negatives, we may miss certain embedding packets, thus two or more IPDs may be combined together. For example, when p_{e_4} is missed, we will have an IPD of $ipd_4 + ipd_5$. However, the missing of p_{e_4} will not affect IPDs other than ipd_4 and ipd_5. Second, due to false positives, we may include certain non-embedding packets that happen to be delayed abnormally long. Such packets may further divide an IPD into two or more sub-IPDs, which may not be multiples of S. In this case, S has to be used to identify those non-embedding packets. Third, network delays will make the IPDs not exact multiples of S, which can be addressed by quantizing S. These problems are common for all four cases and will decrease the accuracy of watermark recovery and duplication.

4.3.2. Case II

Case II is similar to Case I, but a bit more complicated, since we have to identify what packets are used to embed the same watermark bit. Using the approach in Section 4.2, we can estimate $M \times L$ embedding packets except for the first M ones.

Now we identify which packets are used to embed the same watermark bits. Since the extra watermark delays for

[1]The Footfall Project, `http://footfall.csc.ncsu.edu`.

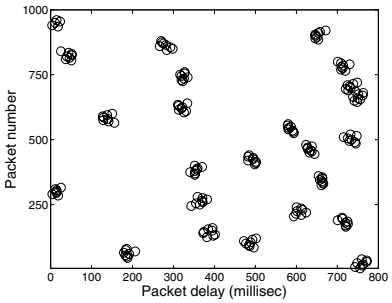


Figure 4. Clusters of embedding packets

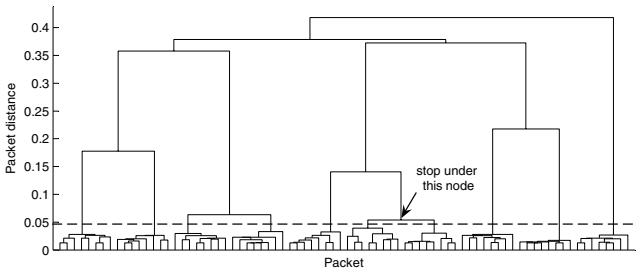


Figure 5. Hierarchical cluster tree

the same watermark bit are identical, we may cluster the embedding packets based on their delays. To seek patterns, we performed an experiment with watermark parameters $S = 400$ms, $M = 8$, $L = 24$, and normal network delays with $\mu = 0$, $\sigma = 10$ms. Figure 4 shows the packet number (y-axis) and the packet delay (x-axis) of the observed packets. We can see there are clear patterns. It is because to ensure the correctness of the watermark scheme, the delayed packets (i.e., the second packets in IPDs) for watermark bit i cannot come later than the delayed packets for bit $i + 1$, i.e., delayed packets of one bit cannot *overlap* with those of another bit.

We adopt the agglomerative algorithm (AGNES) [12] to cluster the watermark delayed packets for different watermark bits. The clustering features are packet sequence number and packet delay. This algorithm starts by placing each packet in its own cluster. Each time, it merges two clusters with the most *similarity* together. Here we define the *similarity* as the shortest Euclidean distance between any packets in two clusters. Before computing the distances, we normalize the packet numbers and packet delays based on their maximum and minimum values. Based on packet distances, we create a *hierarchical cluster tree*. The cluster tree for the first 10 clusters in Figure 4 is shown in Figure 5. The leaf nodes in the tree are packets (singleton clusters) and non-leaf nodes are clusters. The height of a non-leaf node is the distance of the two nodes merged at that node.

We need to decide when to stop the merging process and output the clusters. If chosen correctly, the number of clusters should be L, and the number of packets in each cluster should be roughly the same (M in the ideal situation). We use the variance of normal network delays σ estimated in Section 3 to decide when to stop merging. We stop merging at node n_i if the delay difference of any two packets in the clusters to be merged is out of the range $\pm 3\sigma$. Since the delay differences of the packets for a watermark bit are only caused by normal network delays, this guarantees that 99.73% of the chance clusters can be formed correctly under the normal distribution assumption about normal network delays. We then approximate M by computing the *average cluster size* as the median value of all cluster sizes. To further improve the performance and compensate for inaccurate estimation of σ, we also test the nodes above and below the chosen stop node n_i and generate various cluster formations. Only those cluster formations that still have the same average cluster size are kept. This is to make sure we do not over-split or over-merge the clusters. We then compute the variance of cluster sizes for these cluster formations to determine the final stop node that can produce the most similar cluster sizes.

Having the clusters, we calculate the average IPDs for adjacent clusters. Due to false positives and false negatives in the previous steps, our clustering algorithm cannot guarantee all clusters have the same size. Therefore, unlike the procedure in Section 2, the average IPD is computed as the difference of average timestamps of adjacent clusters. Now the situation is similar to case I. Following the approach in Case I, the watermark can be recovered through computing the GCD of the $L - 1$ average IPDs.

4.3.3. Case III

In Case III, each watermark bit is embedded on 1 IPD and embedding packets are not reused. Because the watermark encoder only delays the second packets in IPDs, the first packets will not be delayed and cannot be identified by our algorithm in Section 4.2. Thus we cannot compute the IPDs to recover the watermark. However, the watermark scheme can still be compromised if the same watermark is added to normal flows. Attackers can confuse the watermark decoder by increasing the false positive rate. Therefore, we investigate how to duplicate the watermark using the packet delay information.

In this case, although the exact location is unknown, the first packet in the IPD of bit i must fall between the two adjacent second packets for bits $i - 1$ and i. We also know the watermarked IPD is a multiple of S. Using these two conditions, we find a number of possible first packets of IPDs. (For brevity, we simply refer to the first packet of an IPD as the first packet.) Due to network delays, we only require the IPD for a possible first packet is *close* to a multiple of S. For normal network delays with mean μ and variance σ^2,

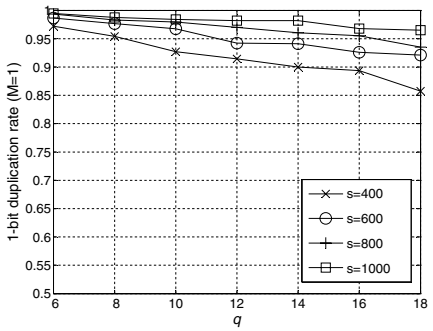


Figure 6. 1-bit duplication rate (M = 1)

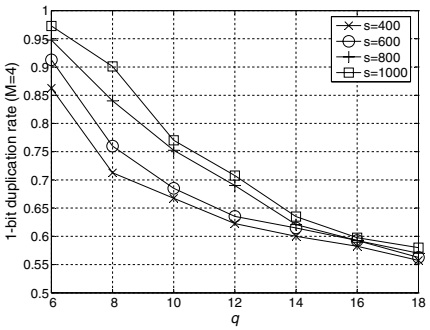


Figure 7. 1-bit duplication rate (M = 4)

the watermarked IPD is normally distributed around multiples of S with mean 0 and variance $2\sigma^2$. So we define *close* as within $\pm 3\sqrt{2}\sigma$ of a multiple of S. The probability we miss the real first embedding packet is only about 0.0027.

For better duplication, we want the number of possible first packets, which is affected by several factors, is as small as possible. Assuming IPDs are independent and uniformly distributed in $[0, S)$ if mod S, the probability an IPD is *close* to multiples of S is $P_c = \frac{6\sqrt{2}\sigma}{S}$. Let q be the number of packets between two adjacent identified embedding packets. The probability that exactly i IPDs are close is $P_c(i) = \binom{q}{i}(P_c)^i(1-P_c)^{q-i}$. So the expected packet number of possible first packets is: $E_p(q) = \sum_{i=0}^{q} i \times P_c(i)$.

Now we discuss how those possible first packets can facilitate watermark duplication. Let $p_{f_1}, \ldots, p_{f_k}$ be the possible first packets, and p_e be the second packet for a watermark bit in the stepping stone flow. Let $p'_{f_1}, \ldots, p'_{f_k}$ and p'_e be the corresponding packets in the normal flow, on which we duplicate the watermark bit by delaying p'_e. For every p_{f_i}, we decode an possible watermark bit w_{f_i} with p_e. Since the real watermark bit is unknown, our strategy is to try all delays from 0 to $2S-1$ for p'_e. For each delay, we decode k watermark bits with $p'_{f_1}, \ldots, p'_{f_k}$, and compare them with the watermark bits w_{f_i} from the stepping stone flow to find the optimum delay d_{opt} that maximizes the number of matched bits. Therefore delaying p'_e by d_{opt} can increase the possibility of successful watermark duplication.

We first use simulation to examine the performance of our duplication algorithm. Figure 6 shows the 1-bit duplication rate (i.e., the probability to detect the same bit in the normal flow after duplication) for different S and q, in which all the rates are greater than 85%. Note that by randomly delaying p'_e, the duplication rate will be only 50%. We can further derive the increased false positive rate for the entire watermark after the duplication. For an L-bit watermark and a hamming distance threshold h, our algorithm increases the false positive rate to: $\sum_{i=0}^{h}\binom{L}{i}(P_{dup})^{L-i}(1-P_{dup})^i$, where P_{dup} is the 1-bit duplication rate as shown in Figure 6. For example, when $L = 24$ and $h = 5$, the original false positive rate is only $\sum_{i=0}^{h}\binom{L}{i}(\frac{1}{2})^L = 0.33\%$. If the 1-bit duplication rate is $P_{dup} = 0.9$, the false positive rate dramatically increases to 97.23%.

4.3.4. Case IV

In Case IV, each watermark bit is embedded with M IPDs and requires $2M$ distinct embedding packets. Similar to Case III, the embedding packets for different bits do not interleave, i.e., the $2M$ packets for bit i must all come earlier than the $2M$ packets for bit $i+1$.

The clustering algorithm discussed in Case II is used to identify the embedding packets for different watermark bits. Because the first embedding packets still cannot be identified, we follow the algorithm in Case III to find the possible first embedding packets. However, now for each bit, we are looking for a combination of M possible first packets, instead of only 1 packet as in Case III. The average IPD differences ipd_{avg} between the M possible first packets and the M identified second packets are required to be *close* to a multiple of S, where *close* is still defined as within $\pm 3\sqrt{2}\sigma$. Suppose there are totally q packets between adjacent identified embedding packets, through a similar analysis as in Case III, the expected number of possible combinations of first embedding packets is $E_p(M, q) = \sum_{i=0}^{q'} i \times P_c(M, i)$, where $q' = \binom{q}{M}$, and $P_c(M, i) = \binom{q'}{i}(P_c)^i(1-P_c)^{q'-i}$. Clearly the equation $E_p(q)$ in Case III is a special case of $E_p(M, q)$ when $M = 1$.

The watermark duplication algorithm for Case IV is also very similar to Case III. In order to duplicate one bit, we try all delays from 0 to $2S-1$ for the M identify second packets, decode watermark bits from the combination of M possible first packets, then find the optimum delay d_{opt}. We also perform initial evaluation through simulation; the result for $M = 4$ is shown in Figure 7. Compared with Figure 6, the duplicate rates are lower and decrease more quickly with increasing q, because of the larger number of possible first embedding packets. Using the same equation as in Case III, we can also compute the increased false positive rate after duplication.

4.3.5. Case V

We do not have special algorithm for Case V. We may still apply algorithms similar to those for Case IV. However, to find possible first packets, we cannot restrict only to the packets between adjacent second packets. A lot more packets for each watermark bits need to be considered. This will increase q and $E_p(M, q)$. As a result, the duplicate rate of our algorithms will decline to around 50% more quickly.

However, even when the watermark may not be duplicated to normal flows, we may remove the watermark easily with the knowledge of S. In particular, we may delay each identified embedding packets by S. Then the average IPDs are also increased by S. The new watermark bits will be the negation of the original ones.

4.3.6. The General Procedure for Watermark Recovery/Duplication

Since attackers do not know which case the watermark encoder uses, they should first run the clustering algorithm to identify the embedding packets used for the same watermark bits. When single IPDs are used (Case I & III), most of the clusters will only contain one packet. Next the attackers compute the average IPDs for adjacent clusters. When packets are reused (Case I & II), the average IPDs should be close to multiples of S. Then they can recover and/or duplicate the watermark following the specific steps for different cases.

5. Detecting Watermark Existence

In Section 4, we have introduced our approaches to compromise the watermark scheme. These approaches require to observe a substantial number of packets in order to produce satisfactory results. In this section, we discuss the watermark detection problem, i.e., how can an attacker detect the existence of watermarks in their flows as early as possible, so that any countermeasures may be applied can have a higher chance to succeed.

When a watermark is embedded, extra delays have to be introduced for certain packets. So packet delay is the natural choice to detect the watermark. However, although most of the network delays are small, we may occasionally find large delays comparable to watermark delays. In the following, we investigate how many (possible) watermark delays must appear before we can decide a flow is watermarked.

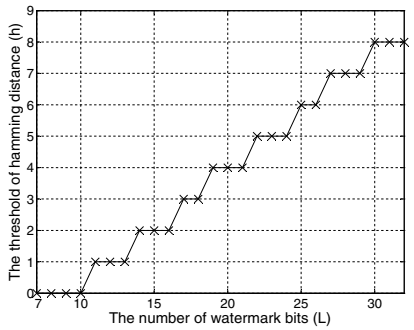
5.1. The Watermark Capability and the Minimum Number of Watermark Delayed Packets

Two things define the usefulness of a watermark. First, the *detection rate* $\mathcal{D}_t$ defines how well a stepping stone flow can be identified. A detection rate is only meaningful when considered under certain amount of timing perturbation. Second, the *false positive rate* $\mathcal{F}_p$ specifies how likely an arbitrary flow may be wrongly identified as the stepping stone flow. In this paper, we name the detection rate and the false positive rate of a watermark as its *capability*, which is determined by the watermark parameters. Given a required capability, we can derive the set of parameters satisfying the capability. We are especially interested in the value of $M \times L$, since this is the number of packets that have to be delayed.

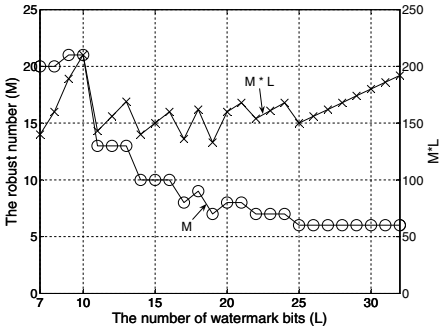
Since each watermark bit has a 50% chance to be decoded from an arbitrary flow, the false positive rate is $\mathcal{F}_p = \sum_{i=0}^{h} \binom{L}{i} (\frac{1}{2})^L$, where h is the threshold of hamming distance. Given a false positive rate, we can derive the relation between L and h. In Figure 8(a), we give out the maximum values of h that fulfill $\mathcal{F}_p \leq 1\%$ for different values of L.

Now we consider the detection rate. Assuming the timing perturbation, which comes from attackers and normal network delays, is distributed with mean 0 and variance σ_p^2, the probability that a single watermark bit can be correctly decoded is $p_d \approx 2\Phi(\frac{S\sqrt{M}}{2\sqrt{2}\sigma_p}) - 1$, where $\Phi()$ is the cumulative distribution function of the normal distribution. For an L-bit watermark, the detection rate is the probability that no more than h bits are incorrectly decoded, i.e., $\mathcal{D}_t = \sum_{i=0}^{h} \binom{L}{i} p_d^{L-i} (1 - p_d)^i$. Note that attackers do not know the value of S at this stage. However, since S cannot be too large (otherwise attackers will notice the abnormal amount of delays in the stepping stone connections), attackers can assume a maximum S, e.g. 1 second. Having the maximum S, the timing perturbation, and the required detection rate, we can derive the relation between M and L. The maximum value of h obtained for the required false positive rate is also used here. For each L, we compute the minimum value of M that satisfies the required detection rate. For example, under the requirement that $\mathcal{D}_t \geq 95\%$, the timing perturbation is uniformly distributed in $[0, 2]$ seconds (i.e., $\sigma_p = 2000/\sqrt{12}$ ms), and $S = 1000$ms, Figure 8(b) shows the minimum values of M under different values of L. The value $M \times L$, which is also shown in the figure, is the minimum number of packets that have to be embedded with extra watermark delays. There always exists a minimum value of $M \times L$ for any desired watermark capability. Therefore we can compute the lower bound on $M \times L$ for any desired watermark capability under reasonable assumptions about timing perturbation and the maximum value of S. Figure 8(c) lists the minimum $M \times L$ for different false positive rates and detection rates when $S = 1000$ms and uniform timing perturbation in [0, 2s].

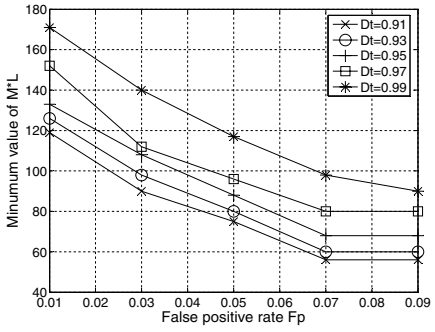
Having the minimum number of embedding packets to be delayed, if the total packet number in the watermarked flow is also known, then the ratio θ of the watermark de-



(a) Computing threshold h



(b) Computing M and $M \times L$



(c) The minimum $M \times L$ for different capabilities

Figure 8. The watermark capability for different watermark parameters

layed packets among all packets is determined. In the next subsection, we demonstrate how this ratio can be used to detect the watermark. Since the stepping stone flows are controlled by attackers, they can always decide how many packets their flows can have. On the other hand, a watermark encoder cannot predict the total number of packets in a flow. In order to guarantee the watermark can be fully embedded, the safe way will be trying to squeeze the watermark into the beginning of the flow. However, this makes the ratio θ very high at least for the beginning part of the flow, thus greatly facilitates watermark detection. An alternative way is to choose packets independently of the possible number of packets, e.g., randomly select the embedding packets from N packets and hope there will be at least N packets in the flow. This way enables the watermark encoder to reduce the ratio θ. However, if the flow is short and the watermark cannot be fully embedded, the capability of the watermark will decrease and the attackers still have better chances of not being caught.

5.2. Detecting Watermark through Sequential Probability Ratio Test

In above, we discover that for a watermark to be useful (i.e., to have a desired capability), there is a lower bound on $M \times L$, which the number of packets that have to be delayed. However, this number cannot be directly used for watermark detection because the embedding packets can be selected from all packets in the stepping stone flow. Instead, we utilize the ratio θ of the watermark delayed packets, which is shown in equation 2. In the following, we first focus on the selection method that chooses embedding packets randomly from N packets. In the end, we show that our algorithm works for any selection methods.

We convert the watermark detection problem to hypothesis testing the value of θ. When θ is big enough, sufficient packets show the characteristic of watermark delays and we can decide a watermark is embedded. We find out that our problem fits naturally into the *sequential hypothesis test* [10] concept. Each time an observation (i.e., a packet delay) is obtained, we make one of the 3 decisions about the hypothesis on θ: (1) to accept the hypothesis, (2) to reject the hypothesis, or (3) to continue the experiment for an additional observation. Since it is also required that a watermark is detected as early as possible, we adopt the Sequential Probability Ratio Test (SPRT) algorithm [26], which minimizes the expected number of observations needed to make the decision.

Intuitively, we can choose a threshold θ' and test two alternatives $\theta < \theta'$ and $\theta \geq \theta'$. However, such an inequality defines *composite hypotheses*, for which it is difficult to derive the SPRT solution for our problem. Following [26], we choose two thresholds $\theta_0 < \theta' < \theta_1$, and test for hypotheses $\theta \leq \theta_0$ and $\theta \geq \theta_1$. These two hypotheses are further converted into *simple hypotheses* $\theta = \theta_0$ and $\theta = \theta_1$, because the SPRT solution for the simple hypotheses can provide a satisfactory result for the original hypotheses. Having the SPRT algorithm on the simple hypotheses, we decide that a watermark is detected when hypothesis $\theta = \theta_1$ is accepted.

The selection of θ_0 and θ_1 is a critical issue, and prior knowledge has to be used. We choose θ_0 based on how well the network delays fit into normal distribution. In our experiment, only 1-2% of the packets do not fit well. The selection of θ_1 is based on attackers' assumptions on the required watermark capability, thus the minimum number of packets had to be delayed. Then attackers can decide the

Table 2. SPRT simulation result

θ	0	0.05	0.1	0.15	0.2	0.25	0.3
output(avg)	0	0	0.8	1	1	1	1
packet #	30	56	438	68	26	17	13

total number of packets in their flows and compute θ_1.

The SPRT algorithm works as follows. Each time a packet p_j is received, we derive its delay d_j and compute

$$r_j = \log \frac{f(d_1, \theta_1) \cdots f(d_j, \theta_1)}{f(d_1, \theta_0) \cdots f(d_j, \theta_0)}, \quad (6)$$

where f is specified in equation 2. Then r_j is compared with two parameters A and B ($B < A$). If $r_j \leq B$, we terminate the testing and accept $\theta = \theta_0$. If $r_j \geq A$, we terminate and accept $\theta = \theta_1$. If $B < r_j < A$, we wait for the next packet. The values of A and B are determined by the pre-selected false negative rate α and false positive rate β for the SPRT algorithm: $A = \log((1-\beta)/\alpha)$ and $B = \log(\beta/(1-\alpha))$.

We have run simulations to validate the usefulness of the SPRT algorithm. For flows with normal network delays with $\mu = 0$ and $\sigma = 10$ms, we embed different watermarks with θ changing from 0 to 0.5. S is set at 400ms. The thresholds are chosen as $\theta_0 = 0.05$ and $\theta_1 = 0.15$. Therefore to attackers, the watermarks with $\theta < \theta_1$ are considered as lacking the required capability and not big threats. The SPRT false negative rate and false positive rate are selected as $\alpha = \beta = 0.05$. The result of the SPRT is 1 or 0, which stands for a watermark is detected or not. Because S is unknown to attackers, our SPRT uses the maximum value of $S = 1000$ms. Table 2 shows the average SPRT output for 100 traces (i.e., the detection rate), and the number of packets processed before termination. We can see that all SPRT results are correct when $\theta \geq \theta_1$ and $\theta \leq \theta_0$. Between θ_0 and θ_1, 80% of the time SPRT can detect the watermark. Note detecting watermarks from non-watermarked traces (false positive) only causes attackers inconvenience and is not a big problem. The packet number required before termination is quite small for $\theta \geq \theta_1$ and $\theta \leq \theta_0$. It increases significantly between (θ_0, θ_1).

A watermark encoder may want to conceal the watermark by choosing a larger N, or by postponing the embedding for a certain number of packets at the beginning of the flow (e.g., 30 packets as shown in Table 2). However, doing so will greatly increase the probability that the watermark cannot be fully embedded and impair watermark capability. To handle this, we slightly change the SPRT algorithm so that it does not terminate when hypothesis $\theta = \theta_0$ is accepted. Instead, it clears all previous result and starts again for the next packet. It only terminates when a watermark is detected or the flow is closed. Therefore, no matter how the embedding packets are chosen, SPRT detects the watermark by looking for a part of the flow where $\theta \geq \theta_1$. Both the original and the modified SPRT algorithms are tested in our experiment.

Table 3. GEM estimation of S

S	400	600	800
mean of S'	389.1	587.8	782.9
std dev of S'	18.44	17.67	24.86

Table 4. GEM estimation of θ

θ	0.1578	0.2366	0.4723
mean of θ'	0.1573	0.2367	0.4694
std dev of θ'	0.0021	0.0023	0.0041

6. Experiments

To evaluate the performance of our approaches to attack the watermark scheme, we have launched a series of experiments. From Sections 6.1-6.5, we test the ability to recover and duplicate watermarks. In Section 6.6 the performance of watermark detection is evaluated.

We use network flows collected from hosts in PlanetLab. Totally 41 traces from our computer to one host in MIT, Hong Kong and Taiwan are used. For each remote host, we select one trace, compute the network delays as in Section 3 and estimate mean μ and variance σ^2. All other traces from the same host will directly use μ and σ without estimating their own parameters.

In our experiment, different values of $S = 400$, 600, 800ms, $M = 1, 4, 6, 8$ and $L = 16, 24, 32$ are tested. The watermark embedding packets are selected using a parameter $K = 2, 4, 6$, i.e., we randomly select one watermark embedding packet from every K packets. For each combination of these 4 parameters, we embed 20 randomly generated watermarks and compute the average result.

6.1. Estimating S and θ

After a watermark is embedded, we first evaluate our GEM algorithm, which estimates the quantization step S and the mixture ratio θ. The result of S is shown in Table 3, in which S' is the estimated value. Since the value of θ changes with other parameters, here we only pick three different values of θ and show the estimation result in Table 4. We can see the estimation of S and θ are very close to the real values. Having the estimated S and θ, we then apply the Bayes decision rule described in Section 4.2 to identify the watermark delayed packets. Zero-one loss is used for λ_{ij} in Equation 4.

Table 5. Watermark recovery rate for Case I

L	16	24	32
$S = 400$	0.9723	0.9679	0.9670
$S = 600$	0.9712	0.9729	0.9705
$S = 800$	0.9712	0.9733	0.9740

Table 6. Recovery rate for Case II (S-L)

L	16	24	32
$S = 400$	0.9317	0.9326	0.9401
$S = 600$	0.9323	0.9421	0.9355
$S = 800$	0.9341	0.9317	0.9460

Table 7. Recovery rate for Case II (S-M)

M	4	6	8
$S = 400$	0.9310	0.9413	0.9319
$S = 600$	0.9421	0.9369	0.9309
$S = 800$	0.9395	0.9311	0.9413

6.2. Watermark Recovery for Case I

In this experiment, we test the performance of our watermark recovery algorithm for Case I, in which $M = 1$ and embedding packets are reused for adjacent watermark bits. The experiment input contains the estimated S and the identified embedding packets. Table 5 shows the watermark recovery rates. This rate is computed as the number of correctly recovered watermark bits divided by $L-1$, since theoretically the first watermark bit cannot be recovered. Our algorithm shows very good performance on recovering watermark. For the different values of S and L tested, the recovery rates are very close and we cannot recognize any clear trend. Since watermark recovery depends on the result of identifying embedding packets, it also shows that our identification algorithm works very well.

6.3. Watermark Recovery for Case II

This experiment evaluates the recovery rate for Case II, in which $M > 1$ and embedding packets are still reused. The identified embedding packets will be processed by our clustering algorithm and the embedding packets used for the same watermark bits are grouped together. Then the average IPDs between adjacent clusters are computed to decode the watermark bits. The result is shown in Tables 6 and 7. In this case, the recovery rates are slightly lower than Case I, and there is no significant variation for the different values of S, M, L tested.

6.4. Watermark Duplication for Case III

In this experiment, a watermark bit is embedded on 1 IPD and the embedding packets are not reused. The duplication algorithm first computes possible first packets for each watermark bit. We then generate a Tcplib [2] synthetic trace, find the optimum delays for all identified embedding packets, and duplicate the watermark. Finally the real watermark decoder decodes watermark W_1 from the synthetic trace and compares it with the real watermark W_0. The duplication rate is the percentage of matched bits between W_0 and W_1. Figure 9 shows the duplicate rate changing with S and K, and Figure 10 shows the duplicate rate changing with L and K. Clearly K has negative impact on duplication. A larger K increases the number of possible first packets, thus reduces the duplication rate. On the contrary, a larger S reduces the number of possible first packets and helps duplication. The impact of L on duplication is not very significant.

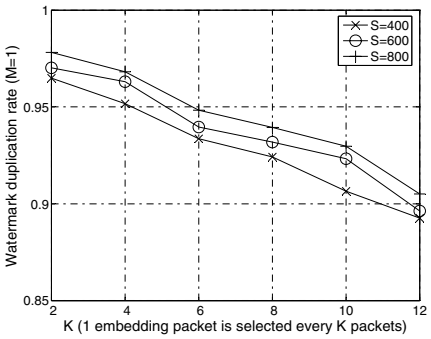


Figure 9. Duplication rate for Case III (M-K)

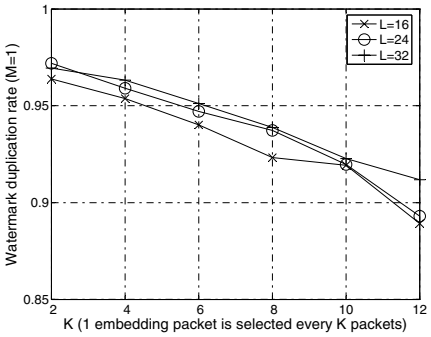


Figure 10. Duplication rate for Case III (L-K)

6.5. Watermark Duplication and Removal for Case IV

In this experiment, we embed a watermark bit using multiple IPDs and embedding packets are not reused. The general procedure is similar to Case III, except here we search for multiple first packets for each watermark bit. The duplication results are shown in Figures 11 and 12. We can see K and M significantly affect the duplication result. Good duplication rates only appear for small M and K. When M and K increase, the number of possible combinations of

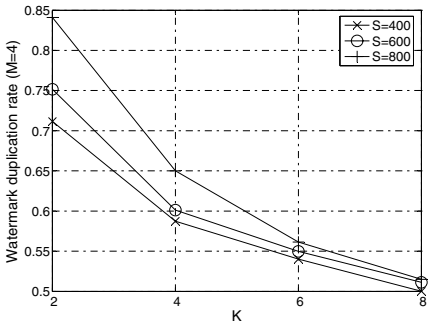


Figure 11. Watermark duplication rate (M = 4)

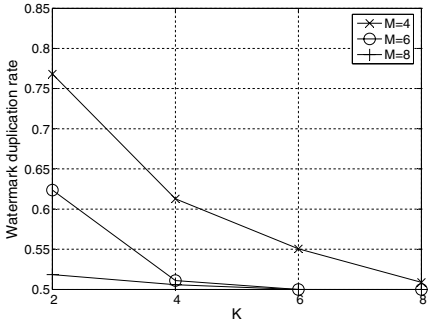


Figure 12. Watermark duplication rate changing with M and K

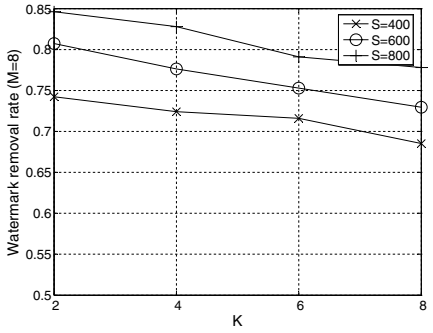


Figure 13. Watermark removal rate (M = 8)

first packets increases very fast, and causes duplication rate to drop dramatically. For larger M and K, our algorithm does not show much difference from random duplication. Figure 13 shows the watermark removal rate (i.e., the percentage of incorrectly decoded watermark bits after our removal algorithm is applied on the flow) when $M = 8$. We can see even when duplication is not successful, the watermark can still be removed effectively.

6.6. Detecting Watermark Existence

We evaluate the performance of the SPRT algorithm in this experiment. The watermark parameters used are the same as in the previous subsections. The real value of θ depends on K and whether embedding packets are reused or not. If packets are reused, $\theta \approx \frac{1}{K}$; $\theta \approx \frac{1}{2K}$ otherwise. When packets are not reused, the first $M \times K$ packet delays are all normal network delays. This may cause problem if the SPRT terminates as soon as it accepts $\theta = \theta_0$. Therefore we test both the original SPRT and the modified SPRT that restarts each time $\theta = \theta_0$ is accepted. We set $\theta_0 = 0.02$ based on our observation for network delays, and set $\theta_1 = 0.1$ to make it less than $\frac{1}{K}$ for the largest K. Since S is unknown to attackers at this stage, $S = 1000$ms is used. We set $\alpha = \beta = 0.05$.

Table 8. SPRT detection rate

	original SPRT			modified SPRT		
K	2	4	6	2	4	6
M = 1	1	1	0.997	1	1	1
M = 4	1	0.989	0	1	1	1
M = 6	0.986	0.984	0	1	1	1
M = 8	0.986	0	0	1	1	1

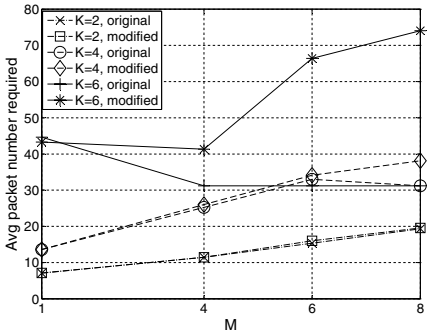


Figure 14. Packets processed

Table 8 shows the detection rates for both the original and the modified SPRT. For the original SPRT algorithm, the detection rate suddenly drops to 0 for 4 values of M and K. It is because there are many packets in the beginning of the flow that are not embedded with watermark. However, using the modified SPRT, watermark is detected in all the cases. Figure 14 compares the average number of packets processed before the SPRT algorithms terminate. For the 4 values of M and K where the original SPRT's detection rate is 0, more packets have been processed by the modified SPRT in order to detect the watermark existence.

7. Related Work

The problem of detecting interactive stepping stones was first formulated by Staniford and Heberlein [24]. Their content-based approach compared thumbprints created from packet payload. Another content-based scheme was Sleepy Watermark Tracing [31], which injected non-

displayable contents into packets. These methods are vulnerable to encrypted traffic such as SSH connections. More recent schemes focused on timing characteristic of packets. Zhang and Paxson [35] proposed an ON/OFF based approach to correlate encrypted traffic. Yoda and Etoh [34] proposed a deviation-based scheme which calculated deviation between an attacking flow and all other flows appeared around the same time. Wang et al. [30] showed that timing characteristics of IPDs were preserved across multiple stepping stones, and could be used for correlation. The problem of these methods is that they are all vulnerable to timing perturbation. Donoho et al. [5] investigated the theoretical limits of the attackers' ability to disguise their traffic through timing perturbation and chaff packets. Using wavelet and multiscale analysis, they demonstrated that stepping stone correlation was still possible by observing long term behavior. Wang and Reeves [28] proposed an active watermarking scheme that was robust to random timing perturbation. They analyzed the trade-offs between true positive rate, the maximum timing perturbation added by attackers, and the number of packets needed to successfully decode the watermark. Their work is also the first one that identifies the provable bound on the number of packets needed to achieve desired correlation effectiveness. Blum et al. [1] proposed to correlate stepping stone connections by counting the packet number differences in certain time intervals. Wang et al. [29] also proposed a probabilistic watermarking scheme with even timing adjustments and better true positive rate. Peng et al. [18] proposed to use active watermarking scheme and packet matching to detect stepping stone connections when there existed both timing perturbation and extra chaff packets. Several algorithms were proposed with different trade-offs between detection rate, false positive rate and computation cost. In [27], the authors provided a watermark approach that could effectively identify the encrypted peer-to-peer VoIP calls even when they were transmitted through low-latency anonymizing networks.

Related to tracing through stepping stones, IP trace-back technologies are developed to identify the origin of attacks when source IP addresses are spoofed by attackers, such as in DDoS attacks. One class of IP trace-back approaches is based on probabilistic packet marking. In these approaches (e.g. [11, 21, 23, 33]), a router marks each processed packet with certain information about its path. Such information is used to rebuild the complete path when the packets are received. Another type of IP trace-back is based on packet logging, where each router logs message digests of the packets in order to construct the packet path (e.g., [14, 22]).

Various anonymous communication systems have been proposed to provide privacy for the users of different Internet services. Among them, Onion Routing [25] and its second generation Tor [4] are seeking to provide a general anonymous routing protocol for different Internet applications. Tarzan [8] is a peer-to-peer anonymous network overlay and provides general-purpose and transparent anonymizing service. It also uses cover traffic to prevent traffic analysis of a global observer. Multiple papers have investigated the timing analysis/attacks to anonymous systems. In [32], the authors studied the threats of passive logging attacks and described a defense from breaking the assumption of uniformly random path selection. Levine et al. [13] investigated the passive timing based attacks on low-latency anonymous systems. Fu et al. [9] studied the degradation of anonymity in a flow-based wireless mix network under flow marking attacks. In [16], the authors showed that the low latency of Tor made it insecure against traffic analysis attacks by a global adversary.

8. Conclusion

Timing-based active watermarking schemes are effective for tracing back through stepping stones or an anonymizing network. However, the schemes themselves are also under the threats of attackers. In this paper, we analyze the secrecy of a timing-based active watermarking scheme for tracing through stepping stones [30]. Based on the models for normal network delays and watermark delays, attackers can successfully estimate the important watermark parameter S with the (G)EM algorithm. The four cases of watermark embedding are investigated, and watermark recovery and duplication methods are developed. We also investigate the interrelation between the watermark capability and the minimum number of packets to be delayed, and propose to use the SPRT algorithm to quickly detect the watermark existence. In experiments, we have shown our attack scheme can efficiently and effectively identify the watermark. When watermark parameters are not selected cautiously, almost the entire watermark can be recovered or duplicated. The watermark removal algorithm also puts major threat on the watermark scheme.

Although developed for a specific scheme, the general ideas in our attack can be extended and are potentially useful to compromise other approaches that actively manipulate packet timing. In the future, we would like to investigate the secrecy of other watermark approaches, such as [27, 29].

Acknowledgment: The authors would like to thank Xinyuan Wang and the anonymous reviewers for their valuable comments.

References

[1] A. Blum, D. Song, and S. Venkataraman. Detection of Interactive Stepping Stones with Maximum Delay Bound: Algorithms and Confidence Bounds. In *Proceedings of the 7th*

International Symposium on Recent Advances in Intrusion Detection (RAID), September 2004.

[2] P. B. Danzig and S. Jamin. Tcplib: A library of TCP/IP traffic characteristics. *USC Networking and Distributed Systems Laboratory TR CS-SYS-91-01*, October, 1991.

[3] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm(with discussion). *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Proceedings of the 13th USENIX Security Symposium*, August, 2004.

[5] 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 *Proceedings of the 5th International Symposium on Recent Advances in Intrusion Detection (RAID)*, October 2002.

[6] R. Duda, P. Hart, and D. Stork. *Pattern Classification (2nd Edition)*. John Wiley & Sons, 2001.

[7] FindNot.com. FindNot–Anonymous Surfing, Anonymous Email & Anomymous Internet. `http://www.findnot.com/`.

[8] M. Freedman and R. Morris. Tarzan: A Peer-to-Peer Anonymizing Network Layer. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, 2002.

[9] X. Fu, Y. Zhu, B. Graham, R. Bettati, and W. Zhao. On Flow Marking Attacks in Wireless Anonymous Communication Networks. In *Proceedings of the 25th International Conference on Distributed Computing Systems (ICDCS)*, 2005.

[10] B. K. Ghosh and P. K. Sen. *Handbook of Sequential Analysis*. Marcel Dekker, 1991.

[11] M. Goodrich. Efficient Packet Marking for Large-Scale IP Traceback. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, 2002.

[12] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 1990.

[13] B. Levine, M. Reiter, C. Wang, and M. Wright. Timing Analysis in Low-Latency Mix Systems. In *Proceedings of the 8th International Conference on Financial Cryptography*, Feb 2004.

[14] J. Li, M. Sung, J. Xu, and L. Li. Large-Scale IP Traceback in High-Speed Internet: Practical Techniques and Theoretical Foundation. In *Proceedings of IEEE Symposium on Security and Privacy (S&P)*, 2004.

[15] G. J. Mclachlan and T. Krishnan. *The EM Algorithm and Extensions*. John Wiley & Sons, 1997.

[16] S. J. Murdoch and G. Danezis. Low-Cost Traffic Analysis Of Tor. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (S&P)*, 2005.

[17] V. Paxson. On Calibrating Measurements of Packet Transit Times. In *Proceedings of SIGMETRICS '98*, June 1998.

[18] P. Peng, P. Ning, D. S. Reeves, and X. Wang. Active Timing-Based Correlation of Perturbed Traffic Flows with Chaff Packets. In *Proceedings of 2nd International Workshop on Security in Distributed Computing Systems (SDCS)*, 2005.

[19] Planet-Lab: An Open Platform for Developing, Deploying, and Accessing Planetary-scale Services. `http://www.planet-lab.org`.

[20] S. Rai and D. Matthews. Improving the EM algorithm. *Biometrics*, 49:587–591, 1993.

[21] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical Network Support for IP Traceback. In *Proceedings of the ACM SIGCOMM 2000*, April 2000.

[22] A. Snoeren, C. Partridge, L. Sanchez, C. Jones, F. Tchakountio, B. Schwartz, S. Kent, and W. Strayer. Single Packet IP Traceback. *ACM/IEEE Trans. on Networking*, 10(6):721–734, Dec. 2002.

[23] D. Song and A. Perrig. Advanced and Authenticated Marking Scheme for IP Traceback. In *Proceedings of IEEE INFOCOM'01*, April 2001.

[24] S. Staniford-Chen and L. Heberlein. Holding Intruders Accountable on the Internet. In *Proceedings of the 1995 IEEE Symposium on Security and Privacy (S&P)*, pages 39–49, Oakland, CA, 1995.

[25] P. Syverson, D. Goldschlag, and M. Reed. Anonymous Connections and Onion Routing. In *Proceedings of the 18th Annual Symposium on Security and Privacy (S&P)*, May 1997.

[26] A. Wald. *Sequential Analysis*. John Wiley & Sons, 1947.

[27] X. Wang, S. Chen, and S. Jajodia. Tracking Anonymous Peer-to-Peer VoIP Calls on the Internet. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, 2005.

[28] X. Wang and D. S. Reeves. Robust Correlation of Encrypted Attack Traffic Through Stepping Stones by Manipulation of Inter-packet Delays. In *Proceedings of the 2003 ACM Conference on Computer and Communications Security (CCS)*, pages 20–29, 2003.

[29] X. Wang, D. S. Reeves, P. Ning, and F. Feng. Robust network-based attack attribution through probabilistic watermarking of packet flows. Technical Report TR-2005-10, North Carolina State University, Department of Computer Science, February 2005.

[30] X. Wang, D. S. Reeves, and S. F. Wu. Inter-Packet Delay Based Correlation for Tracing Encrypted Connections Through Stepping Stones. In *D. Gollmann, G. Karjoth and M. Waidner, editors, 7th European Symposium on Research in Computer Security - ESORICS*, October 2002.

[31] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill. Sleepy Watermark Tracing: An Active Network-Based Intrusion Response Framework. In *Proceedings of 16th International Conference on Information Security (IFIP/Sec)*, June 2001.

[32] M. Wright, M. Adler, B. N. Levine, and C. Shields. Defending Anonymous Communications Against Passive Logging Attacks. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy (S&P)*, May 2003.

[33] A. Yaar, A. Perrig, and D. Song. Pi: A Path Identification Mechanism to Defend against DDoS Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2003.

[34] K. Yoda and H. Etoh. Finding a Connection Chain for Tracing Intruders. In *F. Guppens, Y. Deswarte, D. Gollmann and M. Waidners, editors, 6th European Symposium on Research in Computer Security - ESORICS*, October 2000.

[35] Y. Zhang and V. Paxson. Detecting Stepping Stones. In *Proceedings of 9th USENIX Security Symposium*, pages 171–184, August 2000.