ABSTRACT

PENG, PAI. Active Timing Based Techniques for Attack Attribution through Stepping Stones. (Under the direction of Dr. Peng Ning).

The purpose of the research is to study the active timing based techniques used for attack attribution through stepping stone computers, where attackers sequentially connect through multiple intermediate hosts to hide their traces. The difficulties of tracing back such attacks come from not only the normal operations of networks and stepping stones, but also the intentional interference of the attackers. Encryption, repacketization, timing perturbation, and meaningless chaff packets could all significantly affect attribution result.

In this dissertation, I have investigated multiple research problems related to the active timing based attack attribution. First, I present a correlation scheme that can successfully identify stepping stone connections even if both chaff packets and timing perturbations are introduced by attackers simultaneously. In this scheme, we enhance the existing active watermark schemes and focus on identifying the possible corresponding packets in the flows to be correlated. We develop a series of algorithms to effectively and efficiently decode the embedded watermarks when chaff packets are inserted, and use theoretical analysis and experimental evaluation to validate these algorithms. We also investigate how our correlation scheme can be used to deal with the countermeasure when stepping stone connections are split and then merged, and propose an approach to mitigate the problem of packet loss and retransmission.

Next, I present the research on the secrecy issues of the quantization based watermark scheme. We propose an attacking approach based on analyzing the one-way packet transit delays between adjacent stepping stones. Our attack contains several techniques that can infer important watermark parameters and remove/duplicate the embedded watermarks. These techniques enable an attacker to defeat the watermarking system in certain cases by removing watermarks from the stepping stone connections, or replicating watermarks to non-stepping stone connections. We have also developed techniques to detect in realtime whether a stepping stone connection is being watermarked for trace-back purpose. Experiments using real-world data are performed and the results demonstrate that for the quantization based watermark scheme, (1) embedded watermarks can be successfully recovered and duplicated when the watermark parameters are not chosen carefully, and (2) the watermark existence in a network flow can be quickly detected.

Third, I present the research result on the secrecy of the probabilistic watermark scheme.
Following the ideas of analyzing the quantization based watermark scheme, we propose attacks that can detect the watermark existence, recovery important watermark parameters, and remove/duplicate watermark to effectively defeat the watermark scheme. We also investigate the problem of realtime watermark recovery and removal, and propose an online attacking algorithm. Experiments are then conducted to validate our analysis.

Finally, I investigate the secrecy issues of the interval based watermark scheme and propose several security enhancements to deter possible timing analysis attacks. I demonstrate that the interval based scheme is not robust against several attacks we construct, which can quickly detect watermark existence, recover watermark parameters and defeat the watermark scheme. Through experiment, we validate that the improved scheme with security enhancements will significantly increase the resistance to all of these severe attacks.
ACTIVE TIMING BASED TECHNIQUES FOR ATTACK ATTRIBUTION THROUGH STEPPING STONES

by

PAI PENG

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

COMPUTER SCIENCE

Raleigh

2006

Approved By:

________________________________________  ______________________________________
Dr. S. Purushothaman Iyer  Dr. Ting Yu

________________________________________  ______________________________________
Dr. Peng Ning  Dr. Douglas S. Reeves
Chair of Advisory Committee
I dedicate this dissertation to my parents for always believing in me and supporting me in everything I do. Without their encouragement, understanding, and most of all love, the completion of this work would not have been possible.
Biography

Pai Peng received his Master’s degree in 2000 and Bachelor’s degree in 1998 both from the Department of Computer Science and Technology at Tsinghua University of China. He is currently a PhD candidate in the Department of Computer Science at North Carolina State University. His research interests are in computer and network security, especially on intrusion detection. Currently he is working on the network attack attribution problem.
Acknowledgements

There are a lot of people I would like to thank for making all of this possible. First, I am deeply grateful to my advisor, Dr. Peng Ning. Without his knowledge, perceptiveness, instruction and support, I would have never finished. I would like to thank Dr. Douglas S. Reeves for giving me numerous advice on my research, paper writing and presentation skills. Thanks to my committee members, Dr. Purush Iyer and Dr. Ting Yu, for their valuable suggestions to my dissertation. Also, thanks to Dr. Xinyuan Wang, Feng Fang, Young June Pyun and Young Hee Park for their generous help on my research.

I would like to thank all my officemates and friends at North Carolina State University. I am only able to list a few here: Kun Sun, Dingbang Xu, Yan Zhai, Donggang Liu, Pan Wang, and Qinghua Zhang. Many thanks to them for their inspiration to my study, and their help to my life here in North Carolina.

Finally, my family is always there for me. They have always been encouraging and supporting me to pursue my dreams. I would like to express my deepest gratefulness to them.
Contents

List of Figures viii
List of Tables x

1 Introduction 1
  1.1 Problem Statement: Attack Attribution through Stepping Stones . . . . . . . 3
  1.2 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
    1.2.1 Robust Correlation Scheme against Possible Countermeasures . . . . . . 5
    1.2.2 Secure Correlation Scheme against Possible Attacks . . . . . . . . . . . 7
  1.3 Summary of Research Results . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
  1.4 Dissertation Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Background 11
  2.1 Trace-Back Techniques through Stepping Stones . . . . . . . . . . . . . . . . . 11
    2.1.1 Host Based Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
    2.1.2 Packet Content Based Approaches . . . . . . . . . . . . . . . . . . . . . 13
    2.1.3 Packet Timing Based Approaches . . . . . . . . . . . . . . . . . . . . . . 14
    2.1.4 A Related Correlation Approach in Wireless Network . . . . . . . . . . . 17
  2.2 Related Work in Anonymous Systems . . . . . . . . . . . . . . . . . . . . . . . 18
  2.3 Related Work in IP Trace-Back . . . . . . . . . . . . . . . . . . . . . . . . . . 19
  2.4 Related Work in Digital Watermarking . . . . . . . . . . . . . . . . . . . . . . 20
  2.5 Active Timing Based Watermark Schemes . . . . . . . . . . . . . . . . . . . . . 21
    2.5.1 Quantization Based Watermark Scheme . . . . . . . . . . . . . . . . . . . 21
    2.5.2 Probabilistic Watermark Scheme . . . . . . . . . . . . . . . . . . . . . . . 23
    2.5.3 Interval Based Watermark Scheme . . . . . . . . . . . . . . . . . . . . . . 26

3 Active Timing Correlation Scheme of Traffic Flows with Chaff Packets 28
  3.1 Problem Statement . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
  3.2 Proposed Approach . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
    3.2.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
    3.2.2 Phase 1: Packet Mapping . . . . . . . . . . . . . . . . . . . . . . . . . . . 32
    3.2.3 Phase 2: Flow Correlation Algorithms Using the Best Watermark . . . . . 33
5.2.2 The SPRT Algorithm ........................................ 83
5.3 Recover Watermark Parameters .................................. 86
  5.3.1 Recover the Delay Value $a$ ................................ 86
  5.3.2 Identify Watermark Embedding Packets ...................... 87
  5.3.3 Recover Other Parameters .................................. 87
5.4 Removing Watermark ............................................... 87
5.5 Duplicate Watermark into Normal Flows ......................... 88
5.6 Experiments ....................................................... 89
5.7 Attack the Watermark Schemes in Realtime ..................... 90
  5.7.1 Realtime Algorithm for The Probabilistic Watermark Scheme 90
  5.7.2 Realtime algorithm for The Quantization Based Watermark Scheme 92
5.8 Summary .......................................................... 92

6  The Secrecy and Enhancements of the Interval Based Watermark Scheme 93
  6.1 Basis of Timing Analysis Attacks ................................ 94
  6.2 Detecting Watermark Existence ................................. 96
    6.2.1 The Selection of $\theta_1$ .................................. 96
    6.2.2 Detecting Watermark by Sequential Probability Ratio Test 97
  6.3 Watermark Parameter Recovery .................................. 99
    6.3.1 Recover the Watermark Packet Delay $\delta$ ................ 100
    6.3.2 Identifying Packets Delayed by Watermark ................ 101
    6.3.3 Recovering the Interval and Offset ...................... 101
    6.3.4 Recovering Watermark Bits .............................. 104
  6.4 Watermark Removal and Duplication ............................ 106
    6.4.1 Watermark Removal ....................................... 106
    6.4.2 Watermark Duplication .................................... 106
  6.5 Security Improvements over the Original Scheme ............... 108
    6.5.1 Enhancement 1: Security Improvements for Watermark Detection 108
    6.5.2 Security Improvements to Resist Watermark Recovery .......... 110
    6.5.3 The Improved Encoding and Decoding Algorithm .............. 111
  6.6 Experiment ........................................................ 111
    6.6.1 Attacking the Interval Based Watermark Schemes ........... 111
    6.6.2 Computation Overhead of Timing Analysis Attacks ........... 115
    6.6.3 The Performance of Improved Interval Based Watermark Scheme 116
    6.6.4 Parameter Selection for the Improved Watermark Scheme ...... 119
  6.7 Summary .......................................................... 119

7  Conclusion and Future Work .......................................... 121
  7.1 Conclusion ....................................................... 121
  7.2 Future Work ..................................................... 122

Bibliography .......................................................... 125

A  Frequently Used Symbols ............................................. 132
# List of Figures

1.1 Attack through a single stepping stone ........................................ 3  
1.2 Attack through a chain of stepping stones .................................... 3  
1.3 The illustration of flow split and merge ....................................... 7  
2.1 Quantization based watermarking: embed 1 watermark bit .................. 22  
2.2 Quantization based watermarking: embed an $L$-bit watermark ............... 23  
2.3 Probabilistic watermarking: IPD differences .................................. 25  
2.4 Interval based watermark scheme: embedding bit 0 .......................... 27  
2.5 Interval based watermark scheme: embedding bit 1 .......................... 27  
3.1 Determining packet mappings ...................................................... 32  
3.2 The largest and smallest IPDs ..................................................... 35  
3.3 Retransmission over one stepping stone (no perturbation) .................... 44  
3.4 The illustration of flow split and merge ....................................... 45  
3.5 (a) True positive rate changing with $\lambda_c$, $\Delta = 7s$; (b) True positive rate changing with $\Delta$, $\lambda_c = 3$. .................................................. 47  
3.6 False positive rate in the second case: (a) changing with $\lambda_c$, $\Delta = 7s$; (b) changing with $\Delta$, $\lambda_c = 3$. .................................................. 47  
3.7 Computation overhead: (a) for correlated flows changing with $\lambda_c$, $\Delta = 7s$; (b) for correlated flows changing with $\Delta$, $\lambda_c = 3$. .................................................. 49  
3.8 Computation overhead (a) for uncorrelated flows changing with $\lambda_c$, $\Delta = 7s$; (b) for uncorrelated flows changing with $\Delta$, $\lambda_c = 3$. .................................................. 49  
3.9 True positive rate for flow split & merge ...................................... 50  
4.1 Overview of our attacks to the watermark scheme ............................ 55  
4.2 Packet delays before clock skews are removed. (a) MIT; (b) Hong Kong .... 57  
4.3 A typical distribution of one-way packet delays after the clock skew is removed . 57  
4.4 Probability density for normal and the sum of normal and uniform distributions . 57  
4.5 Watermark delay distributions ..................................................... 58  
4.6 Clusters of embedding packets ..................................................... 64  
4.7 Hierarchical cluster tree ............................................................ 64  
4.8 1-bit duplication rate ($M = 1$) .................................................. 67
4.9 1-bit duplication rate (M = 4) .............................................. 67
4.10 The watermark capability for different watermark parameters .......... 70
4.11 Duplication rate for Case III (M-K) ........................................ 76
4.12 Duplication rate for Case III (L-K) ........................................ 76
4.13 Watermark duplication rate (M = 4) ....................................... 76
4.14 Watermark duplication rate changing with M and K ....................... 76
4.15 Watermark removal rate (M = 8) .......................................... 77
4.16 Packets processed .............................................................. 78

5.1 Minimum of $M \times L$ ......................................................... 82
5.2 SPRT results ($d^* = 4$) ...................................................... 84
5.3 SPRT false positive when actual $\sigma$ is different from estimated $\sigma$ ................................. 85

6.1 Distributions of $f_X$ and $f_Y$ .................................................. 95
6.2 SPRT detection rate (guessed watermark delay used in SPRT is 200ms) .... 98
6.3 The number of packets processed by SPRT before SPRT terminates .......... 98
6.4 SPRT false positive rate ....................................................... 99
6.5 $N_{conf}$ changing with $O$ when $I = 1000$ms ................................ 103
6.6 $N_{conf}$ changing with $I$ when $O = 2000$ms .............................. 103
6.7 Divide the flow into embedding/non-embedding areas ....................... 103
6.8 Difficulty in watermark duplication ....................................... 107
6.9 Watermark duplication result ............................................... 107
6.10 Reduce packet delays .......................................................... 109
6.11 Using random packet delays ................................................ 109
6.12 SPRT watermark detection rate ............................................. 113
6.13 The number of packets processed by SPRT before it terminates .......... 113
6.14 Estimation error of $\delta$ using EM algorithm .............................. 113
6.15 $N_{conf}$ changing with different $I$ for correct $O$ ........................ 114
6.16 $N_{conf}$ changing with different $O$ for correct $I$ ........................ 114
6.17 Watermark removal for the original and the improved schemes .......... 115
6.18 WM duplication using the heuristic algorithm on both schemes ........ 115
6.19 True positive rate with enhancement 2 ($I_{min} = 700$ms, $I_{max} = 1300$ms) .... 117
6.20 True positive rate with enhancement 2 ($I_{min} = 500$ms, $I_{max} = 1500$ms) .... 117
6.21 True positive rate with enhancement 1 ..................................... 117
6.22 True positive rate with enhancement 3 ..................................... 117
6.23 True positive rates with all 3 enhancements ($I_{min} = 700$ms, $I_{max} = 1300$ms) ... 118
6.24 True positive rates under both timing perturbation and repacketization 118
6.25 False positive rate, $I_{min} = 700$ms and $I_{max} = 1300$ms ............... 119
6.26 False positive rate, $I_{min} = 500$ms and $I_{max} = 1500$ms ............... 119
# List of Tables

1.1 Frequently used notations ........................................ 4

2.1 Parameters for quantization based watermarking ....................... 24
2.2 Parameters for probabilistic watermarking ........................... 25
2.3 Parameters for interval based watermarking .......................... 27

3.1 Parameters for probabilistic scheme ................................. 46
3.2 Parameters for quantization scheme .................................. 46

4.1 SPRT simulation result ............................................. 72
4.2 GEM estimation of $S$ ............................................ 74
4.3 GEM estimation of $\theta$ ......................................... 74
4.4 Watermark recovery rate for Case I ................................ 75
4.5 Recovery rate for Case II (S-L) .................................... 75
4.6 Recovery rate for Case II (S-M) .................................... 75
4.7 SPRT detection rate ................................................ 77

5.1 Parameters for probabilistic watermarking ........................... 80
5.2 Parameters .......................................................... 82
5.3 SPRT false positive .................................................. 85
5.4 EM result ............................................................ 87
5.5 EM result - collateral delay ......................................... 87
5.6 Watermark removal result ........................................... 88
5.7 Watermark removal result - collateral delay .......................... 88
5.8 Duplication Result .................................................... 89
5.9 Duplication Result - collateral delay ................................ 89
5.10 Parameters used in experiment ...................................... 90
5.11 Realtime WM removal (Probabilistic) ............................... 92
5.12 Realtime WM removal (Quantization) ............................... 92

6.1 SPRT detection rate .................................................. 98
6.2 The estimation of $\delta$ using EM algorithm ........................ 101
6.3 Watermark recovery ................................................ 105
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.4</td>
<td>SPRT detection rate</td>
<td>110</td>
</tr>
<tr>
<td>6.5</td>
<td>Parameters for the improved scheme</td>
<td>111</td>
</tr>
<tr>
<td>6.6</td>
<td>Parameters for watermark encoding/decoding</td>
<td>116</td>
</tr>
<tr>
<td>A.1</td>
<td>Symbols used in stepping stone connections</td>
<td>132</td>
</tr>
<tr>
<td>A.2</td>
<td>Symbols used in watermark schemes</td>
<td>133</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Network based intrusions have been growing rapidly in recent years, despite the huge amount of resources that have been put into network and information security. The Internet makes every computer connected to it a potential target for malicious attacks, which could come from any location around the world. Even the most secure Internet sites, such as the servers of transnational corporations and government agencies, have experienced security violations and network intrusions. The increase of network attacks results partly from the lack of accountability on the Internet. The design of the Internet emphasizes predominantly functionality, availability and efficiency. For example, the legitimacy of IP source addresses is usually not enforced. Routing protocols are engineered purposively stateless. All of these facts enable attackers to conceal their identities easily. Besides various technologies such as access control, firewalls and intrusion detection systems (IDSs) that prevent and detect the network based intrusions, attack attribution techniques are also urgently needed to trace back to the real attackers and bring them to justice.

For attackers, hiding their real identities is always among their highest priorities. Various methods have been developed to cover the true source of intrusions. One widely used camouflage technique is IP spoofing, in which the source IP addresses of the packets in the attacking flows are spoofed. Since the IP routing is based mainly on destination addresses, the forged IP packets can still reach the targeted victims. However, as the packets with the disguised IP source addresses traverse the Internet and pass through network routers, their true origins are lost and very little useful information is left for analysis. IP spoofing is widely used in denial of service attacks (DoS) and distributed denial of service attacks (DDoS). The forged IP addresses make it very difficult to either...
block or trace such attacks. The term **IP trace-back** is used to represent the techniques that trace IP packets through a series of network routers. Many IP trace-back schemes have been developed by researchers, such as [44, 48, 46, 64].

Another popular and effective method for attackers to maintain obscurity is to employ intermediate hosts in their intrusions. Such intermediate hosts could be *stepping stones*, *zombies*, or *reflectors*. The difference between these intermediate hosts are given in [51] and [23]. In this dissertation, I am focusing on tracing attacks through interactive stepping stone computers. When targeting high-profile victims, the intruders may not attack directly from their own computers. Instead, they could first login (using protocols such as Telnet, rlogin, or SSH) to another host that they have secretly gained access to beforehand, and then attack from that computer. This process is shown in figure 1.1. In this scenario, even if the stepping stone computer is successfully identified, e.g. through IP trace-back, it would be difficult to trace back to the real origin of the attack from the stepping stone. For better protection, the intruders may employ a sequence of stepping stone computers. They can sequentially connect from one stepping stone to another using secure protocols such as SSH [49]. The malicious instructions are passed along the sequence of stepping stones without disclosing the contents or the source of the attack. Only on the last host, the final attack actions will be issued toward the real victim. This process is shown in figure 1.2. The attackers may use the sequence of stepping stones that traverses the entire world, which will make the attack attribution even harder. A host does not need to be deeply compromised to become a stepping stone. The only thing needed is ordinary user’s privileges to login to the host and to create outgoing network connections. Such kind of access may be obtained from dictionary attacks [9] on password files, by snooping the unencrypted passwords on the Internet, or through certain kinds of social engineering [47].

The solutions of the IP trace-back problem are not capable of tracing attacks through stepping stones. Unlike the traffics between different routers where IP packets are solely duplicated and propagated, the traffics between different stepping stones belong to different TCP connections. Most of the network characteristic will be modified by the stepping stones. Such characteristics include IP/TCP header, the payload (if transmitted in secure channels), the packet size, etc. All these factors make tracing attacks through stepping stones a difficult problem. Moreover, controlled by attackers, the stepping stones may intentional manipulate the flows they receive to deter attack attribution schemes. For example, they can change the timing characteristics of the traffics by delaying the packets or inserting meaningless padding packets.

The problem of tracing back attacks through stepping stones has attracted many researchers.
The focus of current research is to correlate together the different connections that belong to the same stepping stone attack. Various approaches have been developed, such as [61, 11, 1, 60, 41]. However, existing schemes still have many limitations and are far from prefect. The countermeasures of attackers have also advanced continuously along with the correlation schemes. In this dissertation, I will thoroughly investigate the advantage and disadvantage of the existing attack attribution techniques, and develop novel methods that can resist different countermeasures that attackers may utilize.

1.1 Problem Statement: Attack Attribution through Stepping Stones

In this section, I formally define the attack attribution problem through stepping stone computers, and describe the notations used throughout this dissertation.

We use $h_1 \leftrightarrow h_2$ to represent a bi-directional network connection between hosts $h_1$ and $h_2$, and $h_1 \rightarrow h_2$ to represent a uni-directional flow from $h_1$ to $h_2$. In a stepping stone attack, an
attacker employs a series of hosts $h_1, h_2, \ldots, h_k$ by creating a sequence of connections $h_1 \leftrightarrow h_2 \leftrightarrow \cdots \leftrightarrow h_k$. Such a sequence is called a \textit{connection chain}, and the intermediate hosts $h_1, h_2, \ldots, h_k$ are called \textit{stepping stones}. Assuming $j > i$, we call flow $h_i \rightarrow h_{i+1}$ an \textit{upstream} flow of $h_j \rightarrow h_{j+1}$, and $h_j \rightarrow h_{j+1}$ a \textit{downstream} flow of $h_i \rightarrow h_{i+1}$. Intuitively, information is propagated from an upstream flow to its downstream flows. For a packet $p_i$, we use $t_i$ to represent its timestamp. A flow $f$ can also be represented by the sequence of all its packets as $(p_1, p_2, \ldots, p_n)$, where $t_i < t_{i+1}$.

When only packet timing is concerned, we may simply denote $f$ as $(t_1, t_2, \ldots, t_n)$.

Using the above notations, the \textit{attack attribution} problem of a stepping stone connection chain $h_1 \leftrightarrow h_2 \leftrightarrow \cdots \leftrightarrow h_k$, which is also called the \textit{trace-back} problem, can be defined as: \textit{Given the last connection $h_{k-1} \leftrightarrow h_k$, identify other connections that belong to the same connection chain}. A perfect solution can enable us not only to trace back to the real origin of the attack, but also discover every stepping stone exploited. However, any practical approach tends to miss certain connections (not 100\% true positive rate) or falsely identify some connections that do not belong to the connection chain (not 0 false positive rate).

We do not aim directly at the general trace-back problem. Similar to most of existing schemes, we focus on a more basic stepping stone flow correlation problem: \textit{Given a flow $f$ and a suspicious flow $f'$, determine whether $f'$ is a downstream flow of $f$ or not}. A solution to this correlation problem would be very helpful to address the trace-back problem, as long as we can access and process all the connections in the chain.

Some frequently used notations are listed in Table 1.1.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>known stepping stone (upstream) flow</td>
</tr>
<tr>
<td>$f'$</td>
<td>a suspicious downstream flow</td>
</tr>
<tr>
<td>$p_i, t_i$</td>
<td>a packet in flow $f$ and its timestamp</td>
</tr>
<tr>
<td>$p'_i, t'_i$</td>
<td>a packet in flow $f'$ and its timestamp</td>
</tr>
<tr>
<td>$c_i$</td>
<td>a meaningless chaff packet inserted by attacks</td>
</tr>
<tr>
<td>$ipd$</td>
<td>inter-packet delay between two packets</td>
</tr>
</tbody>
</table>
1.2 Motivation

1.2.1 Robust Correlation Scheme against Possible Countermeasures

Tracing attacks through stepping stones is a difficult problem. Since the traffics between stepping stones belong to different TCP connections, information in the IP and TCP headers will be changed after passing a stepping stone and cannot be used for correlation purpose. Moreover, controlled by attackers, the stepping stones can intentionally manipulate the outgoing traffics to alter some important flow characteristics used for correlation. Generally, a working correlation scheme should be able to deal with the following countermeasures that may be used by attackers.

1. **Payload encryption.** Early methods correlate stepping stone connections based on the payload of the packets in different connections [50, 62]. Since the information is propagated from upstream flows to downstream flows, the packets must contain same or similar contents. However, such methods can only be used when packets are transmitted through unencrypted channels, e.g., using protocols such as Telnet or rlogin. If attackers utilize secure transmission protocols such as SSH or IPsec [18], payload encryption can completely remove the similarity between corresponding packets.

2. **Timing perturbation.** Knowing the limitation of packet content based schemes, researchers switched to investigate the packet timing in the flows between different stepping stones. In normal stepping stone connections, as soon as one stepping stone receives a packet, it will forward the information by sending out another packet. That means the connections in the same connection chain should display similar packet timing characteristics. Several approaches that analyze packet timing information have been proposed [67, 65, 61]. The problem in these methods is that they are only based on normal behavior of stepping stones. If the stepping stones do not forward the packets immediately, the timing characteristics on which these methods rely may be significantly changed. Since attackers can control the stepping stone hosts, they can intentionally alter the packet timing information through simply delaying the packets to be sent out by certain amounts of time.

3. **Chaff packets.** The latest correlation schemes are able to resist up to limited amounts of timing perturbations. These schemes focus on the interactive stepping stone connections, where the timing perturbations added by attackers are bounded. It is because otherwise the attackers will have a lot of trouble in receiving the responses and deciding their next moves.
By assuming a maximal amount of timing perturbation \( \Delta \), a stepping stone must forward a packet received at time \( t_i \) no later than \( t_i + \Delta \). Using this property, researchers have proposed schemes [11] and [1]. On the other hand, if we randomly select \( m \) packets, some packets may experience small timing perturbations while others may have large timing perturbations. Combined together the effects of perturbations are likely to cancel each other. Based on this idea two active schemes [59] and [60] were proposed, in which the flows are embedded with digital watermarks that are robust to timing perturbations.

However, all the above schemes are not robust under extra padding packets, also called \textit{chaff} (In [1] the authors investigated chaff packets, but their approach only worked when chaff packets were less than 10\% of normal packets). Chaff packets are meaningless packets generated by stepping stones and sent out with normal packets. Their solely purpose is to change both the timing characteristics and the number of packets of the flows. The difficulty is that when transmitted in encrypted channel, chaff packets cannot be distinguished from normal packets. If we use \( c_i \) to represent chaff packet, a flow \( \langle p_1, \ldots, p_n \rangle \) may be changed to \( \langle c_1, \ldots, p_1, c_j, \ldots, p_n, \ldots, c_k \rangle \). If encrypted, we have no way to know which packets are chaff packets.

4. Flow split and merge. Another method attackers can use to reduce the effectiveness of existing trace-back schemes is flow split & merge. At a certain split host in the stepping stone connection chain, the outgoing flow is split into several sub-flows. Each sub-flow contains a part of the packets of the original flow. These sub-flows are transmitted independently until they reach a merge host. In the merge host, all sub-flows come together and the “original” flow is restored. This procedure is illustrated in figure 1.3. Since each sub-flow only has an incomplete copy of packets, it is difficult to correlate sub-flows with either the original flow or the restored flow. Flow split & merge can be used to prevent the re-construction of stepping stone chains.

Besides the above countermeasures attackers can utilize, the normal operation of stepping stone computers and network devices will also affect the timing characteristics of the flows. For example, timing variation could results from propagation delays of the stepping stones or the networks. Packets could be lost and then re-transmitted between stepping stones. The packetization between stepping stones could be different according to the network protocols and network usages. For example, two or more IP packets may be merged into one packet if they arrive too closely to
each other. Large packets may also be split to comply with network configuration. In our experiment, we have found out that when large timing perturbation is introduced, repacketization is very common and will affect most of existing correlation schemes. All of the above problems make attack attribution a very complex problem.

1.2.2 Secure Correlation Scheme against Possible Attacks

Another critical problem that motivates my research is the secrecy and security of the existing active timing schemes. Since the watermark schemes actively involve in packet transmission and manipulate the packet timing, it is possible that attackers can be aware of the watermark existence. For example, attackers may hypothesize the presence of a watermark based on whether the packet delays of certain packets between stepping stones vary significantly from other packets or not. If such a watermark detection approach can be developed, the attackers will have a very good chance to abort their attack before being traced back. Moreover, we want to investigate whether the current schemes will leak any important information to the attackers when the existence of watermark can be detected. Such information may be used to effectively and efficiently defeat the watermark scheme, through removing the embedded watermark to decrease the true positive rate, and/or duplicating the watermark to other normal flows to fool the watermark decoders and increase the false positive rate. Such threats of attackers must be fully investigated and the security enhancements must be developed before the watermark schemes can be widely used in practice.

1.3 Summary of Research Results

In this dissertation, I present the research results on the attribution techniques to trace back attacks through stepping stones.
1. **Correlation scheme for traffic flows with chaff packets.** This scheme is developed to correlate stepping stone flows when both timing perturbation and chaff packets are introduced by attackers simultaneously. In this scheme, I improve the active watermark schemes proposed in [59] and [60]. The basic idea is to first actively embed a timing based watermark into an upstream flow. Then we use a packet-matching technique that is similar to the method proposed in [66] to determine the corresponding packets in a suspicious flow. Based on the assumption that the maximum timing perturbation is bounded by $\Delta$, for each packet in the upstream flow $f$, we can decide a set of its possible corresponding packets in the suspicious flow $f'$ when $f'$ is really the downstream flow. Using these sets of corresponding packets, we decode watermarks from all possible combinations of the corresponding packets. From these watermarks, the one that is closest to the original watermark is used to decide the correlation result. Through theoretical analysis and experimental evaluation, we have shown that our scheme is effective for detecting stepping stone connections under both chaff and timing perturbation. I also investigate how our correlation scheme can be extended to identify stepping stone connections when attackers employ the technique of flow split & merge. A part of the results has been published in [37] and [38]. The result of this research is included in paper [39].

2. **The secrecy of the quantization based watermark scheme.** In [59], Wang and Reeves proposed a novel idea to correlate the stepping stone flows when there exist limited amount of timing perturbations. In this approach, a digital watermark is embedded into the upstream flow through manipulating the inter-packet delays (IPD). By quantizing the selected IPDs into certain values, which are multiples of a parameter quantization step, watermark bits 0 or 1 can be embedded. The changes on packet timing will then propagate into all its downstream flows. The watermark embedded should be detected in all the downstream flows, but nowhere else. Through theoretical analysis and experimental evaluation, the authors have shown that this scheme is robust to limited amount of random timing perturbations. However, the threats of smart attackers to the watermark scheme have not been carefully analyzed in their paper. Attackers may be able to detect the watermark delays by comparing the timing characteristics of the upstream/downstream flow entering/leaving a stepping stone. Then the attackers may close their connections immediately so that the watermark detector cannot receive enough packets to identify the downstream flow. Moreover, if the attackers have observed a lot of delays from the watermark scheme, they may be able to recover a part of watermark parameters.
Using these parameters, they could embed the same watermark to other traffic in order to divert the attention of the watermark detector. In this dissertation, I investigate the methods that the attackers could use to detect the watermark existence and infer the watermark parameters. The result is published in [36].

3. The secrecy of the probabilistic watermark scheme. The probabilistic watermark scheme [60] was proposed to provide realtime watermark embedding functionality with better watermark true positive rate and false positive rate than the quantization based watermark scheme. Similarly, this watermark scheme is under the possible attacks of intelligent attackers. We use similar attacking approaches as those developed for the quantization based scheme to investigate the security of this scheme. We will answer three questions: 1) how to determine whether an attacking flow is embedded with watermark or not; 2) how to find out where the watermark is embedded; and 3) how to defeat the watermark scheme by removing the watermark from the attacking flow and/or duplicating the watermark to other normal flows.

4. The secrecy of the interval based watermark scheme and security improvements. The interval based scheme [41] was proposed to provide robust correlation under both random timing perturbation and repacketization. Most of the existing timing based schemes cannot handle repacketization since they assume that packet counts do not change between stepping stones. Unfortunately, the stepping stones may choose to merge multiple packets into a single packet, or even split a large packet into several packets. Such changes on packet counts will cause the existing watermark decoding algorithms out of synchronization and generating false result.

Similarly, we will investigate the attackers’ threats on this watermark scheme, in terms of watermark detection, parameter inference, and watermark removal/duplication. Moreover, we will also develop several security enhancements based on our insights on the weakness of the original scheme. We will demonstrate that the improved scheme can significantly increase the resistance to proposed attacks, while the original scheme is much more vulnerable.

Since my work is closely related to the active watermark schemes proposed in [59], [60] and [41], these three schemes will be described in detail in Section 2.5.
1.4 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 introduces the research literature related to the trace-back problem. Chapter 3 describes our correlation scheme that deals with both timing perturbations and extra chaff packets. Chapter 4 describes our work on attacking and analyzing the quantization based watermark scheme. Chapter 5 gives the research result on the secrecy of the probabilistic watermark scheme. Chapter 6 investigates the secrecy of the interval based watermark scheme and develops several security enhancements. Finally, Chapter 7 concludes my dissertation and summaries the accomplishments I have achieved in my Ph.D. study.
Chapter 2

Background

In this chapter, we introduce the research literatures related to the problem of tracing attacks through stepping stones. In Section 2.1 we briefly introduce various trace-back schemes for stepping stone attacks. In Section 2.2, we introduce the related work in anonymous communication systems. In Section 2.3, we introduce different schemes about IP trace-back. Section 2.4 briefly introduces the background about digital watermarking. Because my work is closely related to the active watermark schemes, Section 2.5 describes the quantization based watermark scheme, the probabilistic watermark scheme, and the interval based watermark scheme in detail.

2.1 Trace-Back Techniques through Stepping Stones

Stepping stone trace-back approaches can be categorized into host based, packet content based or timing based schemes. In a host based scheme, tracing is based on the information collected by the stepping stones themselves. In order to successfully trace back the attacks, all stepping stones in the connection chain must participate in the tracing and report correct information. A content based scheme performs correlation by analyzing the payload of the packets, while a timing based scheme inspects the timestamps of packets. The trace-back approaches can also be classified as active or passive methods. A passive method merely monitors the traffics, thus is unnoticeable to attackers. On the other hand, an active method will manipulate the content or the timing characteristics of the stepping stone connections, usually to make them more distinct from other traffics.
and more similar among themselves. However, it is also more likely for attackers to detect their existence.

### 2.1.1 Host Based Approaches

Initial techniques related to attack attribution are host based approaches, in which the stepping stone hosts themselves are used in tracing.

The Distributed Intrusion Detection System (DIDS) [45] is a host based tracing scheme developed in UC Davis. It monitors all TCP connections and login activities within the supervised network, and keeps track of all the actions and the states of users. The monitored information is then transmitted to the central network intrusion detection system for further analysis. A user may have different IDs (UID) in different computers. To identify the same user throughout the supervised network, DIDS assigns a network-user ID (NID) to a user when he/she first logins to the monitored network. Whenever the users change their UIDs, DIDS associates the UIDs with the NIDs to keep tracks of the users.

The Caller Identification System (CIS) [19] is another host based approach that authenticate the origin of a user when he/she attempts to login to a host. Every host along the connection chain maintains a list about its view of the chain so far. When a user wants to login to the $n$th host, the $n$th host queries the $(n-1)$th host for the list of its view of the predecessor hosts: $n-2, n-3, \ldots, 1$. The $n$th host then queries each of the predecessor host in $(n-2)$’s list for a list of their predecessor hosts. It allows the user to login only if all the lists of predecessor hosts are consistent.

Session Token Protocol (STOP) [3] is a trace-back scheme similar to CIS. It recursively queries the previous hosts in a connection chain for forensic information. A STOP server listens on a pre-defined port of a host, and can be queried about the connections initiated from that host. When queried, STOP returns a secure token consisting of a hash of the user and connection information. If the identified user is remotely connected to the queried host, the STOP server can be directed to query the STOP server on the remote host where the user is coming from. Therefore, each host can record the information of the previous host in the connection chain.

Buchholz and Shields [2] proposed another forensic tool that correlates incoming and outgoing network connections using process information. It associates each process in the system table with origin information if the process is created by a remote user. After a remote user successfully logins to a system, the origin information is recorded and associated with the process. All additional
processes started from the remote session inherit the same origin information. The origin information includes at least five elements: source address, destination address, source port, destination port, and protocol.

The major problem in the host based methods is the trustworthiness of the information provided by the stepping stones. To trace back to the origin of the connection chain, all the information from the stepping stones must be correct. However, it is very possible that attackers can delete or modify such information since they may gain the control in the stepping stones. If one stepping stone provides misleading records, the whole tracing system is fooled. Because host based schemes require all the stepping stones participated in the attacks can be trusted, they are not suitable for usage on the Internet.

2.1.2 Packet Content Based Approaches

The problem of detecting interactive stepping stones was first formulated by Staniford and Heberlein [50]. They proposed a content based approach by creating thumbprints from the payload of packets, and comparing them to find good matches. A flow is first divided into different time periods. At each time period, the frequencies of different characters occurred in that period are saved. Suppose in a given time period, the frequencies for $L$ characters are a vector $(F_1, \ldots, F_L)$, the thumbprint is defined as a linear combination

$$T = \sum_{a=1}^{L} \phi(a)F_a.$$ 

Using principle component analysis, it can decide which $K$ of the $L$ components contribute the most to the variance of the vector. Then such thumbprints $T$ are compared to find correlation.

Another content based scheme is sleepy watermark tracing technique (SWT) [62]. It correlates stepping stone connections by injecting non-displayable watermarks into the contents of packets. The watermark is constructed using virtual null strings. Such strings will have no effect when they are echoed back to the attackers, e.g., “abc\b\b\b”. Therefore attackers will not notice that a sleepy watermark is embedded. The correlation is simply to detect the same watermark in any suspicious flows.

Obviously, such methods require the contents of packets could not change significantly between different flows, which makes them unusable for encrypted traffics such as SSH connections.
2.1.3 Packet Timing Based Approaches

Due to the widely applications of packet encryption, more recent schemes focused on timing characteristic of stepping stone connections. In the following, the first three approaches mainly targeted at the packet encryption, while later schemes also considered timing perturbation.

Zhang and Paxson [67] proposed the first timing based approach that can correlate encrypted traffic. They observed that in an interactive connection, there existed alternate ON and OFF periods. Each time a packet is transmitted, the flow goes into ON period. If there is no packet more than $T_{idle}$ time, the flow goes into OFF period. When an attacker creates a chain of stepping stone connections, the ON/OFF period in each connection should have certain similarities. So the stepping stone connections could be detected by calculating whether their OFF periods coincide or not. For two OFF periods in two connections, they are said to be correlated if the difference between their ending time is less than or equal to a value $\delta$. For two connections, $OFF_1$ and $OFF_2$ stand for the number of OFF periods in each of them, and $OFF_{1,2}$ stands for the number of OFF periods that correlated. These two connections are said to be correlated if

$$\frac{OFF_{1,2}}{\min(OFF_1, OFF_2)} \geq \gamma,$$

where $\gamma$ is a pre-selected threshold value.

Yoda and Etoh [65] proposed a deviation based scheme to detect stepping stone connections. This scheme calculates deviation between a known attacking flow and all other flows appeared around the same time. The deviation measures how far the two flows differ from each other. Since the starting points of the two flows may not be recorded at the same time, their scheme aligns the two flows to compute the smallest deviation. They observed that unrelated flows would usually have deviations large enough to be distinguished from those in the same stepping stone connections.

Wang et al. [61] proposed a correlation scheme based on inter-packet delays (IPDs). In the scheme, it first computes the inter-packet delays for all the adjacent packets in a given flow, and saves the IPDs as a vector $\langle d_1, \ldots, d_n \rangle$. Then the correlation window function is define on the vector as

$$W_{j,s}(\langle d_1, \ldots, d_n \rangle) = \langle d_j, \ldots, d_{j+s-1} \rangle,$$

where $j$ is the starting point of the window, and $s$ is the window size. Given two flows to be correlated, the correlation point function $CPF$ with respect to point $j$ and $k$ is

$$CPF(X, Y, j, k, s) = \phi(W_{j,s}(X), W_{j+k,s}(Y)).$$
Pair \((j, j + k)\) is called a correlation point if \(\max(CPF(X, Y, j, k, s)) \geq \delta_{cp}\), where \(\delta_{cp}\) is a threshold. Then for the two flows, it may compute \(n\) correlation points \((j_1, j_1 + k_1), \ldots, (j_n, j_n + k_n)\). These points are represent using two \(n\)-dimensional vector \(C_x = \langle j_1, \ldots, j_n \rangle\) and \(C_y = \langle j_1 + k_1, \ldots, j_n + k_n \rangle\). Then the correlation between two flows is computed using correlation value function

\[
CVF(C_x, C_y) = \begin{cases} 
0 & n = 0 \\
\rho(C_x, C_y) & n > 1 \\
1 & n = 1
\end{cases}
\]

where \(\rho(C_x, C_y) = \frac{\sum_{i=1}^{n}(j_i - E(C_x)) \times (j_i + k_i - E(C_y))}{\sqrt{\sum_{i=1}^{n}(j_i - E(C_x))^2} \times \sqrt{\sum_{i=1}^{n}(j_i + k_i - E(C_y))^2}}\). The authors tested four different correlation point function: mini/max sum ratio, statistical correlation, normalized dot product 1 and normalized dot product 2, and found out mini/max is the most effective method.

Donoho et al. [11] investigated the theoretical limits of the attackers’ ability to disguise their traffic through timing perturbations. Their correlation scheme was based on wavelet and multi-scale analysis. Let \(N_i(t)\) be the cumulative character counting function on stream \(i\) in time interval \([0, t]\). For an upstream flow (stream 1) and its downstream flow (stream 2), two assumptions hold:

\[N_2(t) \leq N_1(t),\]

which represents the causality, and

\[N_2(t + \Delta) \geq N_1(t),\]

which represents a maximum tolerable delay exists. The authors chose two kinds of wavelets to do the multi-scale analysis. For a wavelet \(\psi(t)\), the multi-scale function is defined as

\[
\psi_{j,k}(t) = \psi((t - k2^j)/2^j)
\]

if \(\psi(t)\) is the boxcar function, and

\[
\psi_{j,k}(t) = \psi((t - k2^j)/2^j)/2^{j/2}
\]

if \(\psi(t)\) is the Haar wavelet. \(j\) is the scale of the multi-scale analysis. Then the wavelet coefficients of each stream \(N_i\) is defined as the inner product of \(\psi_{j,k}(t)\) and \(N_i\)

\[
\alpha_{j,k}^i = \sum_t \psi_{j,k}(t)N_i(t).
\]
The correlation coefficient at scale $j$ is

$$\text{Corr}(j) = \sum_k \alpha_{1,j,k}^1 \alpha_{2,j,k}^2 / \left( \sum_k (\alpha_{1,j,k}^1)^2 \sum_k (\alpha_{2,j,k}^2)^2 \right)^{1/2}.$$  

Using simulation, the flows under small timing perturbations still have correlation score that is very close to 1. The authors also discussed when chaff packets were added, the correlation result would approach to the “normal packets/(normal+chaff packets)” ratio. However, their paper lacked the important discussion about the false positive rate of their method.

Wang and Reeves [59] proposed the first active watermark scheme that was robust to random timing perturbations. They first identified the quantitative trade-offs between the correlation effectiveness (in term of true positive & false positive), maximum timing perturbations added by the adversary, the defining characteristics of the inter-packet timing of flows and the number of packets needed. Their work is also the first one that identifies provable bound on the number of packets needed to achieve desired correlation effectiveness. Wang et al. also proposed another watermark scheme [60] that guaranteed even timing adjustments and had better true positive rate. The details of these two schemes will be discussed in Section 2.5.

Blum et al. [1] proposed to correlate stepping stone connections by counting the packet number differences in certain time intervals. Based on the assumptions that flows are Poisson distributed, the maximum timing perturbation is $\Delta$, and the $p_\Delta$ is the maximum number of packets that may be sent in $\Delta$ time, they provided an algorithm with at most $\delta$ false positive rate. They proved that to achieve $\delta$ false positive rate, no more than $8\frac{p_\Delta^2}{\pi} \log \frac{1}{\delta}$ packets to be observed in the union of the two flows. Then they relaxed the assumption about the Poisson model, and shown that if the flows behaved as sequences of Poisson processes, their algorithm still worked. They also extended their algorithm to deal with small number of chaff packets. However, the number of chaff packets could only consist a very small portions of the entire flow ($< 10\%$). They did not show any experimental evaluation of their algorithms either.

Zhang et al. [66] proposed several algorithms to detect stepping stone connections when timing perturbation or/and chaff packets may be present. Their algorithms were based on finding possible corresponding packets. Their best scheme is scheme S-IV, where the deviation between two flows is computed after all corresponding packets have been decided. They had also developed a fast solution to compute the deviation efficiently. In their experiments, they showed that scheme S-IV was able to detect the downstream flow when timing perturbation and chaff packets were introduced at the same time.
Peng et al. [37] proposed to use active watermark 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 overhead.

In [58], 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. This approach uses the similar ideas as those in the probabilistic watermark scheme [60].

Peng et al. [36] investigated the secrecy issue of the active watermark approaches. They developed multiple attacking techniques and showed that it was possible to detect watermark existence, recover important watermark parameters, and effectively defeat the quantization based watermark scheme in [59].

To deal with TCP repacketization, Pyun et al. [41] proposed an interval based watermark scheme. They proposed to divide the attacking flows into time intervals, and embed watermark by manipulating the numbers of packets falling into different intervals. The details of this scheme will be discussed in Section 2.5.

2.1.4 A Related Correlation Approach in Wireless Network

In this subsection, we introduce a correlation scheme targeting at a similar problem as the tracing problem through stepping stones. The problem is to discover transmitter/receiver pairs based only on the transmission events of nodes in the wireless network. Only the transmission but not the contents in the transmission can be monitored. A node’s transmission may consist of several flows, each flow with an independent destination. Nodes act as routers, receive and forward the transmission to the destination. The goal is to correlate transmitters with receivers to form a path through the wireless network. For this problem, several techniques have been developed in [33]. One method uses a signal processing technique called coherence to perform cross spectral analysis of each node’s transmission. Another method uses a Lagrangian state-space technique, which computes the degree at which two different traces are related using standard windowed time-frequency techniques.
2.2 Related Work in Anonymous Systems

Anonymous communication systems protect the identity information of the participants in network applications. Such applications may include online voting, online auction, etc. Various anonymous systems have been developed, which can provide anonymity for message senders, anonymity for message receivers, or unlinkability between senders and receivers. Similar to tracing attack through stepping stones, researchers interested in anonymous communication systems have also investigated the effects of encryption, timing perturbation, chaff packets, and split & merge. All these methods can be utilized to provide anonymity.

Chaum proposed the first anonymous email application in [4]. The idea is to use relay servers, also known as mixes, to reroute and encrypt the messages with the public keys of the mixes. When a message is sent to the first mix, the mix decrypts the message with its private key to get the address of the second mix and the new message is encrypted using the public key of the second mix. Each mix will decrypt the message and forward to the next mix. Only the last mix will get the real destination of the message. Chaum also proposed return-address and digital pseudonym for users to communicate with each other anonymously. After Chaum, several anonymous email systems were developed. Hughes and Finney [32] built the cypherpunk remailer, which was a distributed mix network with reply function that used PGP to encrypt and decrypt messages. This system is vulnerable to a global passive attack and replay attack. Babel [17] is a relatively full-fledged anonymous email system. The reply in Babel does not need the sender to remember the secret seed to decrypt the reply message, but it is vulnerable to reply attack. Denezis and et al. developed Mixminion [6] which used secure single-use reply blocks. In Mixminion, mix nodes cannot distinguish forward messages from reply messages, so forward and reply messages share the same anonymity set. Despite its problems, the design of Mixminion considers a relatively complete set of attacks.

The idea to use mixes to provide communication anonymity has also been applied to applications other than email. The onion routing project [43] seeks to provide a general anonymous routing protocol for different Internet applications. Onion Routing is a flexible communication infrastructure that is resistant to both eavesdropping and traffic analysis. It accomplishes this goal through separating identification from routing. Connection is always kept anonymous, although communication needs not to be. Communication may be made anonymous by removing identification information from data stream. The onion routing related publications include [54, 15, 55]. Tor [10] is the second generation onion routing. Tor is a circuit based low-latency anonymous
communication system. It improves the first generation by adding perfect forward secrecy, congestion control, directory server, integrity checking, configurable exit policies, and the practical design for location-hidden service via rendezvous points. It works on the real-world Internet, requires no special privileges or kernel modifications, needs little synchronization or coordination between nodes, and provides a reasonable trade-off between anonymity, usability, and efficiency. Besides onion routing, Hordes [24] is another anonymous system that uses forwarding mechanisms similar to onion routing. It is the first to use multicast routing. It has the advantages of shorter transmission latency and less protocol participants, and can provide the similar degree of anonymity.

There are multiple papers that address the timing analysis/attacks to anonymous systems. In [16], the authors quantitatively analyzed anonymous systems with regard to anonymity properties. They used a probabilistic method to investigate the anonymous behavior. In [63], the authors studied the threat of passive logging attacks against anonymous communication. They described a possible defense from breaking the assumption of uniformly random path selection. They showed that their defense improved anonymity when nodes stayed in the system, but failed when nodes left and rejoined. In [68], the authors addressed issues related to flow correlation attacks and the corresponding countermeasures in mix networks. In such attacks, an adversary attempts to analyze the network traffic and correlate the traffic of a flow over an input link at a mix with that over an output link of the same mix. They found a mix with any known batching strategy might fail against such attacks. In [29], the authors showed that the low latency of Tor system made it unsecure against traffic analysis attacks by a global passive adversary. They presented new traffic analysis techniques that allowed attackers with only a partial view of the network to infer which nodes were used to relay the message and greatly reduce the anonymity. Fu et al. [13] studied the degradation of anonymity in a flow based wireless mix network under flow marking attacks. The attack was performed by electromagnetic interference. They found out that the traditional mixes were not effective against such attacks. They then proposed a new countermeasure based on digital filtering technology, and showed it could effectively defend a wireless mix network.

2.3 Related Work in IP Trace-Back

IP trace-back technologies were developed to identify the source of IP packets when the source IP addresses were spoofed by attackers. The result of a packet trace-back was an attack path, which consisted of all the routers traversed by the packet on its journey from the attacker to the vic-
tim. In [44], the packets were probabilistically marked by the routers with partial path information, based on the observation that attackers usually comprise a large number of packets. Although each marked packet only represented a sample of its path, the whole path could be completely re-built after receiving a modest number of packets. In [48], the authors proposed two marking schemes: the Advanced Marking scheme and the Authenticated Marking scheme. Compared with [44], their schemes had the same low network and router overhead, but were more efficient and accurate for attack path reconstruction. Moreover, the authenticated marking supported efficient authentication on the marking of routers, which prevented comprised routers from forging markings from other normal routers. PI [64] (Path Identifier) was a packet marking scheme in which a path fingerprint was embedded in each packet. Each packet traversing the same path was marked with the same identifier, so that the destination host could identify packets traversing the same path on the Internet on a per packet basis, despite IP spoofing. Their scheme also had good efficiency and effectiveness. SPIE (Source Path Isolation Engine) [46] was a hash based technique for IP trace-back that generated audit trails traffic within the network. Their scheme could trace to the origin of a single IP packet delivered by the network in the recent past. Their scheme could also provide explicit trace-back support for packet transforms, generation of attack graphs, no false negatives and minimal false positives. It was also efficient and scalable.

2.4 Related Work in Digital Watermarking

Digital watermarking is a technique to embed hidden information such as copyright notices or other verification messages into carrier signals such as digital audio, video or images [5, 40, 56]. While the additional hidden message does not restrict use of a carrier signal, it provides an approach to track the signal to the original owner. A watermark can be classified into two sub-types: visible and invisible. A watermark scheme should be robust to the attacks aiming at removing or destroying the embedded watermark. For invisible watermark, how to prevent the watermark from being detected is also a problem.

Multiple researchers have been working on modeling and resisting attacks of the watermark schemes. In [22], Kundur et al. investigated attacks such as cropping, filtering and perceptual coding. They proposed to employ principles of diversity and channel estimation to improve performance. Su and Girod [53] investigated the Wiener estimation attack and developed the energy efficient watermarking concept. It was also shown in [52] that Wiener attack is optimal in terms of
least attack energy distortion. In [25], the authors analyzed another type of attack against binary watermarks: the sensitivity attack. They found out there was a way to remove the watermark when the watermark is decoded by comparing the correlation value with a threshold. They proposed to mitigate this problem by using a decision interval instead of decision threshold.

2.5 Active Timing Based Watermark Schemes

2.5.1 Quantization Based Watermark Scheme

Quantization based watermark scheme was published in [59]. This scheme is based on actively embedding a digital watermark into an upstream flow by slightly delaying certain pre-selected packets. The changes on packet timing will then propagate to all of the downstream flows. If the watermark embedded is unique, it should be detected only in all the downstream flows, but nowhere else, with a high probability. Depending on the secrecy of watermark parameters, especially the secret selection of packets to embed the watermark (called embedding packets), the authors had shown that this scheme was robust to random timing perturbations. It is because the timing changes incurred by random perturbations are very likely to cancel the effects of each other, since attackers have no knowledge about where the watermark might be.

The watermark is embedded through manipulating inter-packet delay (IPD). The IPD between a pair of packet \(\langle p_i, p_j \rangle (i < j)\) is defined as: \(ipd_{(i,j)} = t_j - t_i\), where \(t_i\) and \(t_j\) are the timestamps of \(p_i\) and \(p_j\), respectively. For simplicity, when the packets in the IPD are not concerned, we also use \(ipd\) or \(ipd_i\) to represent a single IPD. This scheme uses quantization of IPDs to tolerate certain amount of timing perturbation. Given the parameter quantization step, denoted as \(S\), the quantization function is defined as

\[
q(ipd, S) = \text{round}(ipd/S),
\]  

(2.1)

where function \(\text{round}(x)\) rounds off real number \(x\) to its nearest integer (i.e., \(\text{round}(x) = i\) if \(x \in (i - \frac{1}{2}, i + \frac{1}{2}]\)). It is easy to see that using quantization, a small perturbation may not affect the quantized value.

To embed one binary watermark bit \(w\) (0 or 1), the \(ipd\) is slightly adjusted so that the watermarked IPD (denote as \(ipd^W\)) fulfills the equation \(q(ipd^W, S) \mod 2 = w\). That means when bit 0 is embedded, the watermarked IPD will be the even number of multiples of \(S\), while if bit
Figure 2.1: Quantization based watermarking: embed 1 watermark bit

1 is embedded, the watermarked IPD will be the odd number of multiples of $S$. The watermark encoding function that can achieve this is:

$$e(ipd, w, S) = [q(ipd + S/2, S) + x] \times S,$$

(2.2)

where $x = (w - (q(ipd + S/2, S) \mod 2) + 2) \mod 2$. Here the quantization is on $(ipd + S/2)$ instead of $ipd$. It is to make sure that the watermarked IPD $ipd^W$ obtained is always bigger than the original $ipd$. Then the watermarked IPD is achieved through delaying its second packet by the amount of $(ipd^W - ipd)$. The process to embed one watermark bit is shown in figure 2.1. Hereafter, we call the delay caused by watermark embedding as watermark delay.

The function to decode the watermark bit embedded in an IPD is

$$d(ipd, S) = q(ipd, S) \mod 2.$$  

(2.3)

The decoding function returns exactly the embedded watermark bit when $ipd^W$ is the input. Moreover, as long as the change of the IPD caused by timing perturbation is limited by $(-S/2, S/2]$, it can still return the correct watermark bit.

Therefore a watermark bit embedded in a single IPD can resist up to $S/2$ random timing perturbations. To be stealthy, the selection of $S$ is limited. Generally speaking, larger $S$ will induce bigger delay, which makes the watermark more noticeable to attackers. In order to resist timing perturbations larger than $S/2$ for one watermark bit, the solution is to using multiple IPDs. To embed the watermark bit on $M$ IPDs, we first compute IPD average as

$$ipd_{avg} = \frac{1}{M} \sum_{i=1}^{M} ipd_i.$$  

(2.4)

Then the watermarked IPD average is calculated as $ipd_{avg}^W = e(ipd_{avg}, w, S)$, using equation (2.2). The watermark bit is embedded by adjusting each of these $M$ IPDs by the amount of $(ipd_{avg}^W -$
In watermark decoding, the average of the $M$ watermarked IPDs is computed then equation (2.3) is used: $w = d(ipd_{avg}^W, S)$. Using the same $S$, multiple IPDs can provide higher resistance than a single IPD.

Even using multiple IPDs, a single watermark bit still has limited resistance to timing perturbation. Moreover, from any unwatermarked IPD, we have 50% chance to decode a desired watermark bit. To increase the true positive rate and reduce the false positive rate, multiple-bit watermark is used. An $L$-bit watermark $W = w_1 \ldots w_L$ is embedded by repeating the procedure of embedding single bit $L$ times. Each time a different set of embedding packets should be used. In watermark detection, another $L$-bit watermark $W'$ is decoded from a suspicious flow and compared with $W$. If the hamming distance between $W$ and $W'$ is less than or equal to a pre-defined threshold, we report a stepping-stone flow is detected. The reason that hamming distance is used is to increase the robustness of watermark scheme under timing perturbation. Because the packets used to embed the watermark are kept secret from attackers, this watermark scheme is robust against random timing perturbations. The procedure of choosing embedding packets to embed an $L$-bit watermark is shown in figure 2.2. From this figure, we can see that the embedding packets are formed with a regular pattern. As we will show later, such a pattern could leak critical information to attackers, and make the inference of watermark parameter much easier.

The parameters for the quantization based watermark scheme is listed in Table 2.1.

### 2.5.2 Probabilistic Watermark Scheme

The problem in the quantization based watermark scheme is that the IPD average can only be determined after all the packets used to compute the average IPD are captured. So it is likely
Table 2.1: Parameters for quantization based watermarking

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>Quantization step</td>
</tr>
<tr>
<td>$L$</td>
<td>The number of binary watermark bits</td>
</tr>
<tr>
<td>$M$</td>
<td>Degree of robustness (i.e., the number of IPDs to embed 1 bit)</td>
</tr>
<tr>
<td>$W = w_1 \ldots w_L$</td>
<td>The watermark. Each bit $w_i$ is either 0 or 1</td>
</tr>
</tbody>
</table>

that after the IPD average is computed and the watermarked IPDs are determined, we have already passed the right time to send out the packets. Therefore the encoder may not always be able to embed the watermark in realtime. Knowing this problem, the authors of [59] proposed a realtime watermark scheme [60]. The new scheme also has better true positive rate under timing perturbation and the same false positive rate.

To embed a single watermark bit, we randomly choose $2M$ distinct packets $\langle p_{e_1}, \ldots , p_{e_{2M}} \rangle$, and construct $2M$ packet pairs: $\langle p_{e_i}, p_{e_i+d} \rangle$, where $1 \leq e_1 < \ldots < e_{2M} \leq n - d$. Here $d \geq 1$ is a user-selected parameter. The IPDs of these packet pairs are $ipd_{(e_i,e_i+d)} = t_{e_i+d} - t_{e_i}$ ($i = 1, \ldots , 2M$).

We randomly divide these $2M$ IPDs into 2 groups, $ipd^1$ and $ipd^2$, with each group having $M$ IPDs. We use $ipd^1_i$ and $ipd^2_i$ ($i = 1, \ldots , M$) to denote the IPDs in $ipd^1$ and $ipd^2$, respectively. Apparently, $ipd^1_i$ and $ipd^2_i$ are identically distributed. Therefore $E(ipd^1_i) = E(ipd^2_i)$. The average difference between the IPDs from group $ipd^1$ and $ipd^2$ is defined as

$$D = \frac{1}{2M} \sum_{i=1}^{M} (ipd^1_i - ipd^2_i). \tag{2.5}$$

We should have $E(D) = 0$. Here $M$ is called redundancy number. The bigger $M$ is, the more likely $D$ is equal to 0. Figure 2.3 shows the IPD differences from synthetically generated flows. From this figure, the majority of the IPD differences fall into a small interval around 0.

Now if we increase or decrease $D$ by a value $a > 0$, we can skew the distribution of $D$, and the probability that $D$ will be positive or negative is increased. This observation gives us a way to embed a single bit of watermark probabilistically. To embed a watermark bit 0, we decrease $D$ by $a$. Then for the new average IPD difference $D' = D - a$, it is more likely $D' < 0$. To embed bit 1, we increase $D$ by $a$. Then for $D' = D + a$, it is more likely $D' > 0$. The decrease of $D$ is achieved by decreasing every $ipd^1_i$ and increasing every $ipd^2_i$ by $a$; the increase of $D$ is achieved by increasing every $ipd^1_i$ and decreasing every $ipd^2_i$ by $a$. The increase or decrease of a single IPD can be achieved by delaying the second or first packet in that IPD by the amount of $a$, respectively.
After the watermark bit is embedded, it can be detected by checking if the adjusted $D'$ is less than 0 or not. Bit 0 (or 1, resp.) is decoded when $D' \leq 0$ (or $> 0$, resp.). There exists a slight probability that a watermark bit cannot be correctly embedded. For example, if $D > a$, then bit 0 cannot be embedded since $D' = D - a > 0$. This probability can be reduced by increasing $M$.

Similar to the quantization based watermarking, we embed an $L$-bit watermark by repeating the procedure for one-bit watermark $L$ times. The correlation result is also determined by computing the hamming distance between the decoded watermark and the original watermark. This scheme is also robust to random timing perturbation.

The parameters for the probabilistic watermark scheme is listed in Table 2.2.

<table>
<thead>
<tr>
<th>$a$</th>
<th>Extra watermark delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>The number of binary watermark bits</td>
</tr>
<tr>
<td>$M$</td>
<td>Degree of robustness (i.e., the number of IPDs to embed 1 bit)</td>
</tr>
<tr>
<td>$W = w_1 \ldots w_L$</td>
<td>The watermark. Each bit $w_i$ is either 0 or 1</td>
</tr>
</tbody>
</table>
2.5.3 Interval Based Watermark Scheme

The interval based watermark scheme [41] is proposed to provide robust correlation under both random timing perturbation and repacketization that can change the numbers of packets between stepping stone flows. Most of the existing timing based schemes assume that packet counts do not change. Unfortunately, the stepping stones may choose to merge multiple packets into a single packet, or even split a large packet into several packets. Such changes on packet counts will cause the watermark schemes described previously to be out of synchronization and generate false result.

The idea of the interval based watermark scheme is to use time instead of packet count to achieve synchronization between watermark encoders and decoders. To embed a watermark, a flow is divided into multiple equal length intervals. A watermark bit is embedded by making the number of packets falling into one interval larger than the number of packets in another interval. The number of packets can be increased by “pulling” one or more packets into this interval from its previous interval, and can be decreased by “pushing” one or more packets out of this interval to its next interval. More specifically, the watermark scheme first decides an offset value $O$ and an interval length $I$. The scheme will omit the beginning part of the flow as specified by $O$. The rest part of the flow will be divided into multiple intervals with length $I$. The watermark encoder uses three consecutive intervals for each watermark bit. The first interval is used only in watermark encoding so that the packets in this interval can be pushed to the next interval. The second and third intervals are used in both encoding and decoding. To embed bit 0, the encoder delays all packets in the first interval by a pre-selected delay value $\delta$. So some packets in the first interval may be pushed into the second interval and the number of packets in the second interval increases. The encoder also delays all the packets in the third interval by $\delta$ to reduce the number of packets. To embed bit 1, the encoder only delays all the packets in the second interval by $\delta$ to simultaneously decrease the number of packets in the second interval and increase the number of packets in the third interval. The encoding process is illustrated in Figures 2.4 and 2.5, where the packets in the grey intervals will be delayed.

To decode the watermark, a decoder simply compares the numbers of packets in the second and third intervals. If the second interval has more packets, it decodes 0. Otherwise, it decodes 1.

To increase the robustness of the watermark scheme under timing perturbation and repacketization, one watermark bit will be repetitively embedded on multiple intervals. For an $L$-bit wa-
termark, the encoder first embeds the entire watermark using the first $3L$ intervals. Then it embeds the entire watermark on the next $3L$ intervals, and so on. This process is repeated until it reaches the end of the flow. In watermark decoding, the decoder will compare the number of packets in all the second and third intervals for one watermark bit to decide whether 0 or 1 is embedded.

Table 2.3 summarizes the parameters used in this watermark scheme.

Table 2.3: Parameters for interval based watermarking

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>Interval length</td>
</tr>
<tr>
<td>$O$</td>
<td>Offset value</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Extra watermark delay</td>
</tr>
<tr>
<td>$L$</td>
<td>The number of binary watermark bits</td>
</tr>
<tr>
<td>$w_1 \ldots w_L$</td>
<td>The watermark, in which each bit $w_i$ is either 0 or 1</td>
</tr>
</tbody>
</table>
Chapter 3

Active Timing Correlation Scheme of Traffic Flows with Chaff Packets

In order to detect encrypted stepping stone connections, current approaches are based on flow timing analysis. There are two major countermeasures that may defeat the packet timing based approaches. First, attackers can perturb packet timing through intentionally delaying some or all packets. Such timing perturbation may severely alter the timing characteristics used for correlation analysis. Second, attackers can insert meaningless padding packets, also called chaff, into their flows. Chaff packets are exceedingly difficult to be differentiated from normal packets when transmitted through encrypted channels, and can substantially alter both packet number and packet timing. Some previous work [59, 1, 60, 66] considered how to deal with these two countermeasures separately; however, none of them have investigated robust trace-back techniques that can handle both timing perturbation and chaff at the same time.

In this chapter, we propose a practical solution that can correlate stepping stone connections even when both timing perturbation and chaff packets are introduced simultaneously. Our scheme is an enhancement over two existing timing based active watermark approaches [59, 60], since they are quite capable of dealing with timing perturbation alone. In these two watermark approaches, a secret watermark is embedded in a stepping stone flow by changing the timing for certain packets selected by the watermark encoder. Such timing changes (watermark) will then
propagate to all the following flows in the stepping stone connection chains. Therefore stepping stone flows can be identified by searching suspicious flows for the embedded watermark.

In our scheme, we first use a watermark approach to embed a timing based watermark into the flow to be correlated, then try to detect the watermark in the suspicious flows. However, when extra chaff packets are inserted, the original watermark detection algorithm will fail because the exact location of the embedded watermark cannot be determined. In order to still detect the watermark, we first use packet timestamps to identify multiple possible watermark locations. From each possible location, we decode a watermark and compare it with the original watermark embedded. If the similarity between the decoded and original watermarks is high, a stepping stone flow is identified.

We develop a series of algorithms aimed at efficient watermark detection when there are both timing perturbation and chaff. We demonstrate that our approach can successfully correlate stepping stone flows through theoretical analysis and experimental evaluation. We also discuss how our approach can be utilized to identify stepping stone connections under another extreme concealment technique called flow split and merge, through which an intruder may evade correlation by splitting intrusion instructions over multiple parallel connections at some place in the stepping stone chain and merging them into a common connection at some other place.

The rest of this chapter is organized as follows. Section 3.1 formally defines the correlation problem under chaff packets. Section 3.2 introduces our approach, which consists of a packet mapping procedure and several watermark decoding algorithms. Section 3.3 gives out the theoretical analysis about the performance in terms of true positive rate, false positive rate, and the computation complexity. Section 3.4 discusses how to tolerate certain situations of packet loss. Section 3.5 discusses how our approach can be extended to identify stepping stone connections under flow split and merge. Section 3.6 provides experimental evaluations and comparisons. Section 3.7 summarizes our results.

3.1 Problem Statement

The existing quantization and probabilistic watermark schemes are not capable to detect stepping stone connections when chaff packets are introduced. It is because the assumption used by the watermark schemes that watermarks can be correctly located in downstream flows is now violated. The previous results do not apply any more. Since chaff packets cannot be distinguished
from normal packets in encrypted flows, how to choose the right packets to decode the embedded watermark becomes a problem. Although we can keep using the same schemes to embed watermarks, watermark detection obviously fails if we still decide the embedding packets in a suspicious flow by counting the packet number or TCP sequence number as done in the original schemes.

In other words, suppose an upstream flow \( f = \langle p_1, \ldots, p_n \rangle \), and \( f' = \langle p'_1, c_1, \ldots, p'_n, c_j, \ldots \rangle \) is a downstream flow, where \( c_i \) are chaff packets. The sequence of normal packets \( \langle p'_1, \ldots, p'_n \rangle \) is a subsequence of \( f' \). We may specify it as the subsequence of \( f \) in \( f' \). All other subsequences of \( f' \) with \( n \) packets will at least include one chaff packet. Ideally, if the subsequence \( \langle p'_1, \ldots, p'_n \rangle \) can be solely identified, we can correctly decode the watermark. However, due to the difficulty of distinguishing chaff from normal packets, it is very unlikely, if possible, to find the correct subsequence exclusively.

We target at how to correlate stepping stone flows when chaff packets are inserted along with timing perturbation. Similar to previous trace-back approaches such as [11, 59, 60, 1], we focus on interactive connections and assume an upper bound on the timing perturbation that attackers can introduce. Besides the intentionally added delays from timing perturbation, packet delays also come from networks and intermediate hosts that the connection chain passes through. In our scheme, we assume the clock offsets (i.e., the difference between the clocks at a specific time) for different hosts are known so that packet timestamps in different flows can be adjusted for comparison. In the following, whenever we compare two packet timestamps in different flows, the clock offset is already removed. Compared with clock offsets, clock skews (i.e., the first derivative of the offset) are usually much smaller and not significant for the lifetime of the flows. We assume the errors from packet timestamp adjustment (e.g., errors from clock skews), the timing perturbation from attackers, and delays from all other sources are then collectively bounded by a single maximum delay \( \Delta \). In summary, we have the following three assumptions:

1. The delay between packet \( p_i \) in an upstream flow and its corresponding packet \( p'_i \) in a downstream flow is bounded by \( \Delta \), i.e., \( t'_i - t_i < \Delta \). We call this the timing constraint.

2. Every packet in an upstream flow will go to its downstream flow as a single packet. In Section 3.4 we will discuss how to relax this assumption to deal with packet loss.

3. The packet order in an upstream flow is kept the same in a downstream flow. We call this the order constraint.
3.2 Proposed Approach

3.2.1 Overview

In this chapter, we target at stepping stone correlation when both timing perturbation and chaff may be inserted by attackers. Using the two watermark schemes as our starting point, we are looking for possible enhancements so that the embedded watermark can still be decoded when chaff packets are inserted. We define flow mapping as a function that maps each packet in one flow to some packet in another flow. A flow mapping can be represented by a sequence of packets in the second flow. In our case, the correct mapping is the one that properly maps all the packets in an upstream flow to the corresponding packets in the downstream flow, even when there is chaff. For example, suppose an upstream flow is $f = \langle p_1, \ldots, p_n \rangle$, and $f' = \langle p'_1, c_1, \ldots, p'_n, c_j, \ldots \rangle$ is a downstream flow, in which each $p'_i (i = 1, \ldots, n)$ corresponds to $p_i$ in the upstream flow $f$ and each $c_j (j \geq 1)$ is a chaff packet. The correct mapping will be the sequence $\langle p'_1, \ldots, p'_n \rangle$. If the correct mapping is identified, we can properly decode the watermark.

Since it is exceedingly difficult to uniquely identify the correct mapping, our goal is to find a set of candidate mappings that includes the correct mapping. We then decode a watermark from each mapping. If any of these watermarks is close enough (in terms of hamming distance) to the original watermark embedded, we determine a stepping stone flow is detected. In another word, the watermark detection is based on the best watermark, which has the smallest hamming distance to the original watermark. The rationale is that since the correct mapping is included in the candidate mappings, the correct watermark can be decoded sometime. Then by using the best watermark, we ensure at least the same true positive rate as the original watermark schemes. That is, if a flow can be identified without chaff, it can still be identified after chaff is inserted. However, since multiple mappings are checked, our approach is more likely to decode the watermark from unwatermarked flows and may cause the false positive rate to increase. Therefore we try to reduce the number of the candidate mappings as much as possible by using the constraints in Section 3.1.

In our correlation scheme, we do not make any assumption about the attackers’ ability to add chaff packets. It is worth noticing an attacker cannot increase the false positive rate with more chaff if he/she is the only one to insert chaff. However, if there are multiple attackers adding chaff to different flows, the false positive rate will increase with the overall amount of chaff packets. Therefore, we investigate the relation between false positive rate, chaff and timing perturbation in two cases: 1) Only one attacker adds chaff and perturbation; 2) Multiple attackers can add chaff and
perturbation simultaneously.

Our approach consists of two phases. First, in the packet mapping phase, we determine for every packet in a flow $f$ which packets in $f'$ could be its corresponding packets. Second, using the packet mappings, we create candidates of the correct mapping and compute the correlation result.

### 3.2.2 Phase 1: Packet Mapping

We first determine which packets could be the corresponding packets in the flows to be correlated according to our assumptions in Section 3.1. Suppose $f = \langle p_1, p_2, \ldots, p_n \rangle$ is the upstream flow embedded with watermark $W$. The suspicious flow to be correlated is $f' = \langle p'_1, p'_2, \ldots, p'_{n'} \rangle$. Based on our assumption that every packet in the upstream flow will go into the downstream flow, $f$ and $f'$ are in the same stepping stone chain only if $n' \geq n$. Then for every packet in $f$, we try to determine its possible corresponding packets in $f'$.

For a packet $p_i$ in $f$, we call the packets in $f'$ that possibly correspond to $p_i$ as the mapping packets of $p_i$. For convenience, we call the set of mapping packets as the mapping set for $p_i$. According to the timing constraint, $p_i$'s mapping packets can only appear within the time interval $[t_i, t_i + \Delta]$. Therefore, we may compute the mapping set for $p_i$ as

$$M(p_i) = \{ p'_j \mid t_i \leq t'_j \leq t_i + \Delta \},$$

(3.1)

where $t'_j$ is the timestamp of $p'_j$. The packet mapping phase is illustrated in Figure 3.1. In each mapping set, packets are ascendingly ordered according to their timestamps. If a mapping set is empty, we can immediately decide that $f$ and $f'$ are not in the same connection chain.

Figure 3.1: Determining packet mappings

Now we discuss how the packet mapping sets can be quickly computed. We first compute the packet mapping set of the first packet $p_1$ by scanning $f'$ from the first packet $p'_1$. For all the other
packets $p_i (i > 1)$, we can scan from the first packet in the packet mapping set $M(p_{i-1})$ (i.e., the one having the smallest timestamp), because no other packets with smaller timestamps can satisfy our timing constraint. Moreover, sometimes it is more efficient to scan backward from the last packet in $M(p_{i-1})$. The following heuristics are used to reduce overhead. To compute $M(p_{i+1})$, we compare the timestamps $t_i$ and $t_{i+1}$ of $p_i$ and $p_{i+1}$.

1. If $t_{i+1} - t_i \leq \frac{\Delta}{2}$, the first mapping packet of $p_{i+1}$ is likely to be close to $p_i$’s first mapping packet. So we scan forward from $p_i$’s first mapping packet.

2. If $\frac{\Delta}{2} < t_{i+1} - t_i \leq \Delta$, the first mapping packet of $p_{i+1}$ is likely to be close to the last mapping packet of $p_i$. We scan backward from the last mapping packet of $p_i$.

3. If $t_{i+1} - t_i > \Delta$, the mapping sets $M(p_i)$ and $M(p_{i+1})$ do not overlap. We start directly from the packet next to $p_i$’s last mapping packet.

For example, as shown in Figure 3.1, since $p_{i+1}$ is not very close to $p_i$, we compute the first mapping packet of $p_{i+1}$ by scanning backward from $p_i$’s last mapping packet.

Because of the order constraint, we further eliminate the duplicated first or last mapping packets from packet mapping sets. If $p'_j$ is the first packet in both $M(p_i)$ and $M(p_{i+1})$, $p'_j$ cannot be used as the mapping packet for $p_{i+1}$ since otherwise $p_i$ will have no mapping packet. So any duplicated first mapping packet will be removed from all but the earliest mapping set it is in. Similarly, duplicated last mapping packets will be removed from all but the last mapping set it is in. Removing duplicated first mapping packets is combined with the original procedure. Removing duplicated last mapping packets only requires one extra backward scan from the last packet $p_n$.

### 3.2.3 Phase 2: Flow Correlation Algorithms Using the Best Watermark

After all the packet mapping sets are computed, we create candidate mappings from $f$ to $f'$ to decode the best watermark. Since this procedure can be very time consuming, several algorithms are developed and their advantage and disadvantage are compared. In Sections 3.2.4-3.2.7, we discuss a series of algorithms for the probabilistic watermark scheme, where each later algorithm improves the previous one to achieve better trade-off between effectiveness and efficiency. We also apply similar ideas for the quantization based scheme, and present the algorithm with the best trade-off in Section 3.2.8.
3.2.4 Algorithm 0: Brute Force Algorithm

This algorithm simply enumerates all possible mappings using the packet mapping sets. To form a flow mapping, the selection of mapping packets must not incur conflict with the order constraint discussed in Section 3.1. For example, suppose \( M(p_1) = \{ p'_1, p'_2 \} \) and \( M(p_2) = \{ p'_1, p'_2, p'_3 \} \), only three candidate mappings for \( (p_1, p_2) \) can be created: \( \langle p'_1, p'_2 \rangle, \langle p'_1, p'_3 \rangle, \) and \( \langle p'_2, p'_3 \rangle \).

This algorithm obviously suffers from high computation overhead due to its combinatorial nature. If on average the packet number in mapping sets is \( c \), this algorithm tends to have a complexity of approximately \( O(c^n) \).

3.2.5 Algorithm 1: Greedy Algorithm

Since the correlation result is determined only by the best watermark, we propose a much faster greedy algorithm that forms only one candidate mapping. For every embedding packet in \( f' \), we use the original watermark to help us choose the most promising mapping packet that can generate the same watermark bit as the one embedded. Which mapping packet is selected depends on how an embedding packet is used and whether bit 0 or 1 is embedded.

Suppose an \( ipd = t_{i+d} - t_i \) is used to embed a watermark bit 1 using the probabilistic watermark scheme. To decode bit 1, it is desirable that the IPDs in the first group (\( ipds^1 \)) is bigger, and the IPDs in the second group (\( ipds^2 \)) is smaller. Therefore, the embedding packets should make the IPDs in \( ipds^1 \) be the largest among all possible values, and the IPDs in \( ipds^2 \) be the smallest among all possible values. Then the most promising mapping packet is either the first one or the last one in a mapping set, based on whether the largest IPD or the smallest IPD is wanted. Similarly, we can decide the most promising mapping packets when bit 0 is embedded. The procedure to choose the mapping packets for the largest or smallest IPD is shown in Figure 3.2.

The greedy algorithm then selects the first or the last mapping packets to form one candidate mapping, and computes the best watermark. The computation complexity of this algorithm is \( O(n) \). The watermark decoded by this algorithm will not be worse than the watermark decoded by Algorithm 0 in terms of hamming distance to the original watermark, since it uses the most promising embedding packets. However, the order constraint is not enforced so that the mapping packet chosen for \( p_i \) could be earlier than that for \( p_{i-1} \). This algorithm sacrifices the correctness of the candidate mapping for smaller computation overhead, and tends to have higher false positive rate.
3.2.6 Algorithm 2: Improved Greedy Algorithm Using Order Constraint

Knowing the problems in the above two algorithms, we propose an improvement over the greedy algorithm. This algorithm consists of two steps. First, it scans the candidate mapping created by the greedy algorithm to identify any violations of the order constraint. Such violations are then eliminated by changing the selections of the mapping packets with incorrect order. Second, we construct other candidate mappings by enumerating all the possible choices of mapping packets and compute all the watermarks. Therefore this algorithm combines the ideas of the brute force and the greedy algorithms. By selectively using the mapping packets, we can dramatically reduce the number of candidate mappings to be enumerated in the second step, so that this algorithm can have much lower computation overhead than the brute-force algorithm.

Before we discuss how to remove the conflict with the order constraint, there are two things we can infer from the greedy algorithm. Suppose the candidate mapping created by the greedy algorithm is \( Q_1 \) and the watermark decoded is \( W_1 \). First, if the greedy algorithm returns the conclusion that the flow is not a stepping stone flow, we can terminate immediately and omit all later process. Second, we can also omit the unmatched watermark bits between \( W_1 \) and the original watermark \( W \). Therefore, we only focus on the matched watermark bits and their embedding packets.

The conflicts with the order constraint are removed by scanning the sequence of packets in the candidate mapping \( Q_1 \). When we detect a conflict between two packets \( p_i \) and \( p_{i+1} \) such that \( p_i \)’s mapping packet selected by the greedy algorithm comes later than that of \( p_{i+1} \), we keep \( p_i \)’s mapping packet unchanged and push back the mapping packet of \( p_{i+1} \) until there is no conflict with the order constraint. For better performance, the conflicts are only checked for the watermark embedding packets. We use an example to illustrate this process. Suppose \( M(p_1) = \{ p'_1, p'_2, p'_3, p'_4 \} \),
\[ M(p_2) = \{p'_3, p'_4, p'_5\}, \text{ and } M(p_3) = \{p'_6, p'_7\}. \]
Packets \( p_1 \) and \( p_3 \) are the watermark embedding packets and their mapping packets selected by the greedy algorithm are \( p'_4 \) and \( p'_5 \), respectively. A conflict is detected for \( p_1 \) and \( p_3 \) because the subscript difference for the mapping packets (i.e., \( 5 - 4 \)) is less than that of the original packets (i.e., \( 3 - 1 \)). Therefore no mapping packet is available for the non-embedding packet \( p_2 \). Then we check other mapping packets for \( p_3 \) and find out \( p'_6 \) is the first eligible packet. The new mapping sequence for \( \langle p_1, p_2, p_3 \rangle \) is then \( \langle p'_4, p'_5, p'_6 \rangle \). Note that changing the mapping of one embedding packet may further invalidate the mappings of other packets and invoke collateral changes. However, the packet mapping sets guarantee that all the conflicts can be removed without backtracking. A new watermark \( W_2 \) is then decoded from the new candidate mapping formed by the newly selected packets, and compared with the original watermark. If the hamming distance is less than or equal to the threshold, we report that a stepping-stone flow is detected. Otherwise, we enter the next step to find better watermarks.

In the second step, we only focus on the watermark bits of \( W_2 \) (and their embedding packets) that do not match the original watermark \( W \). All mapping packets are enumerated for those embedding packets to compute different watermarks. The reason that enumeration can be used here is because the searching space for the mapping packets can be significantly reduced after the first step. Note we also omit the matched watermark bits in \( W_1 \). For example, suppose the watermark \( W_1 \) computed from the greedy algorithm has 4 unmatched bits, and the watermark \( W_2 \) computed from the new candidate correct mapping subsequence has 7 unmatched bits. (The 4 unmatched bits in \( W_1 \) must still remain unmatched in \( W_2 \).) So in the second step, we only check the 3 watermark bits left and their embedding packets.

### 3.2.7 Algorithm 3: Improved Greedy Algorithm with Heuristics

Although the first step of Algorithm 2 can substantially reduce the number of embedding packets to be enumerated, the computation overhead in the worse case can still be exponential. Therefore, we utilize other heuristics in the second step to quickly identify the candidate mappings that can produce the best watermark, instead of enumerating all the mapping packets. We expect the computation overhead to be greatly reduced compared with the previous algorithm.

The first step of this algorithm is the same as the previous one. We first use the greedy algorithm to generate a candidate mapping and decode a watermark \( W_1 \). The conflicts with the order constraint is then removed to form a new mapping sequence and decode a watermark \( W_2 \). Only the watermark bits that match in \( W_1 \) but not match in \( W_2 \) will be considered in the second
In order to increase the probability of quickly constructing candidate mappings, we decide which embedding packets should be processed first based on their corresponding IPDs. Recall that in the probabilistic watermark scheme, a watermark bit is decoded based on whether the corresponding average IPD difference is greater than 0 or not. For the watermark bits to be processed, we first sort their corresponding average IPD differences ascendingly. Our intuition is that if the average IPD difference $D_i$ is closer to 0 than $D_j$, then it is more likely we can inverse the sign of $D_i$ than $D_j$ by changing the selections of mapping packets. So the embedding packets for the $i$-th watermark bit are processed earlier than those of the $j$-th watermark bit. For the same watermark bit, we process the embedding packet with the earliest timestamp first.

In addition, we find out that the collateral changes (i.e., changing the mapping packet of one embedding packet affects other embedding packets because of conflicts with the order constraint) may greatly increase the computation overhead. So we do not allow collateral changes if it will result in a previously matched watermark bit becoming unmatched. The embedding packet will stick to its previous mapping packet that does not invoke such a collateral change. Our algorithm then continues for the next embedding packet. This process is illustrated in the following.

1. If the current mapping packet of an embedding packet $p_k$ is already the best one (i.e., same as in the greedy algorithm), we stick to it, and continue for the next embedding packet.

2. Otherwise, we select the next mapping packet of $p_k$, if any. If other packets are collaterally affected, we also adjust their mapping packets to the first ones without conflict. If the changes cause any previously matched watermark bit to be unmatched, we cancel all the changes made so far, and go ahead to process the next embedding packet.

3. Repeat step 2 for packet $p_k$ until (i) the watermark bit matches the original watermark bit, or (ii) $p_k$ has no other mapping packet left. If the watermark bit is still unmatched, we continue with the next embedding packet. Otherwise, we skip all its embedding packet left and go forward to the watermark bit with the next smallest average IPD difference.
3.2.8 Algorithm 4: Decoding the Best Watermark For Quantization Based Watermark Scheme

In Sections 3.2.4-3.2.7, we have developed a series of algorithms for the probabilistic watermark scheme, based on decoding the best watermark. This idea can also be used for the quantization based watermark scheme [59]. To correlate stepping stone flows under chaff packets, we can embed a quantization based watermark on the upstream flow. Because of chaff packets, identifying the exact location of the watermark embedded is impossible. Therefore, we also use the best watermark to determine if a suspicious flow is a stepping stone flow or not.

This algorithm also consists of two steps: 1) constructing a candidate mapping $Q_1$ to decode a watermark $W_1$; 2) improving the watermark $W_1$ by constructing other mappings. Unlike the probabilistic based watermark scheme, mapping $Q_1$ in this algorithm is simply constructed by choosing the first mapping packet for each embedding packet. The reason is that we cannot quickly decide which initial setting of the mapping packets is more likely to generate the desired watermark. So we cannot propose a greedy solution as for the probabilistic watermark scheme. Having the initial mapping sequence, a watermark $W_1$ is then decoded and compared to the original watermark. Only the watermark bits that do not match the original ones will be processed in the second step. In the second step, we also utilize the heuristics so that we do not allow a switch on the mapping packets if the incurred collateral changes will cause a previously matched bit to be unmatched. Although such heuristics may slightly affect the best watermark that can be decoded (e.g., two unmatched bits become matched at the cost of one matched bit changing to unmatched), it can greatly decreases computation overhead.

In the second step, we try to change each unmatched watermark bit by adjusting its corresponding IPDs. An IPD can be changed by switching the mapping packets of its first or second packet. For a watermark bit, we require all the first packets in the corresponding IPDs are processed before any of the second packets. It is because changing the first or the second packet has inverse effect on a watermark bit and should not be combined together. Suppose packets $P(i,1), \ldots, P(i,M)$ and $P(j,1), \ldots, P(j,M)$ are used to embed one watermark bit. The $M$ IPDs are constructed as $ipd_1 = t(j,1) - t(i,1), \ldots, ipd_M = t(j,M) - t(i,M)$. The second step is executed as follows:

1. Change the mapping packet for $P(i,1)$ to the next one available. Keep changing if the watermark bit is still unmatched with the embedded one.
2. Change the mapping packets for \( p_{(i,k)} \), where \( k = 2, \ldots, M \), to the next ones available, i.e., enumerate all the mapping packets for all the first packets if the watermark bit is still unmatched.

3. If the watermark bit is still unmatched, revoke all the changes, i.e., reset all mapping packets to the first ones in the mapping sets.

4. Process the mapping packets for the second packets in the IPDs, i.e., \( p_{(j,1)}, \ldots, p_{(j,M)} \), following the above procedures.

### 3.3 Analysis

#### 3.3.1 False Positive Rate of Packet Mapping

Based on our assumptions in Section 3.1, the packet mapping procedure should always correctly construct mapping sets between an upstream flow and its downstream flows. However, it may also construct mapping sets between the upstream flow and a normal flow. In this subsection, we briefly analyze the false positive rate between two independent flows \( f_1 \) and \( f_2 \). We model \( f_1 \) and \( f_2 \) as two independent Poisson processes with parameters \( \lambda_1 \) and \( \lambda_2 \), respectively. It is easy to see that we must have \( \lambda_1 \leq \lambda_2 \) in this case.

For the first packet \( p_1 \) in \( f_1 \), the probability we can construct its mapping set is that at least one packet in \( f_2 \) appears within time interval \([t_1, t_1 + \Delta] \). So

\[
P_{\Delta,1} = 1 - e^{-\lambda_2 \Delta}.
\]

For two packets \( p_1 \) and \( p_2 \), the probability that there are at least one packet appears within time interval \([t_1, t_1 + \Delta] \) and another packet appears within \([t_2, t_2 + \Delta] \) is then:

\[
P_{\Delta,2} = (P_{\Delta,1})^2 - Pr(t_1' \in [t_2, t_1 + \Delta - t_2] \land t_2' > t_2 + \Delta) \leq (P_{\Delta,1})^2.
\]

So for all the \( n \) packets in \( f_1 \), the probability that every packet can find at least one mapping packet exclusively is:

\[
P_{\Delta,n} \leq (P_{\Delta,1})^n = (1 - e^{-\lambda_2 \Delta})^n,
\]

which is the upper bound of the false positive rate of the packet mapping procedure for Poisson distributed flows.
3.3.2 Algorithms for Probabilistic Watermark

In this subsection, we briefly analyze the true positive rate and false positive rate of probabilistic watermark based algorithms. Suppose random variable $Y_i$ denotes the IPD difference used to embed a watermark bit: $Y_i = ipd^1_i - ipd^2_i$. Based on the analysis in [60], for randomly selected embedding packets, $Y$ is iid, $E(Y) = 0$, and $\text{var}(Y) = \sigma^2$. By using our algorithms, the IPD differences used to decode the best watermark may not be the same as the original IPD differences. Now suppose random variable $X_i$ denotes the changes on IPD differences. We assume mapping packets are independent, which is true if the embedding packets are not close to each other with respect to the maximum delay $\Delta$. We also assume chaff packets are independently inserted. So $X$ is iid, and $X$ and $Y$ are independent. The new IPD difference is then represented by $Y'_i = Y_i + X_i$. Suppose $E(X_i) = \mu_x$ and $\text{var}(X_i) = \sigma_{x}^2$. Then we have

$$E(Y'_i) = E(Y_i) + E(X_i) = \mu_x,$$

and

$$\text{var}(Y'_i) = \text{var}(Y_i) + \text{var}(X_i) + 2\text{cov}(Y_i, X_i) \leq (\sigma + \sigma_x)^2.$$

In the best watermark, the new average IPD difference for one bit is $D' = \frac{1}{2M} \sum_{i=1}^{M} (Y'_i)$, and we have

$$E(D') = \frac{1}{2} E(Y'_i) = \frac{\mu_x}{2},$$

and

$$\text{var}(D') = \frac{1}{4M} \text{var}(Y'_i) \leq \frac{(\sigma + \sigma_x)^2}{4M}.$$

Without loss of generality, we assume the watermark bit is 1. The true positive rate is represented by $\Pr[D' > -a]$. Here we use Chebyshev inequality to give an estimation. From Chebyshev inequality, we have

$$\Pr[|D' - E(D')| \geq E(D') + a] \leq \frac{\text{var}(D')}{(E(D') + a)^2}.$$

Due to the symmetry of the probability distribution of $D'$, we have

$$\Pr[D' \leq -a] = \Pr[D' - E(D') \leq -E(D') - a] \leq \frac{\text{var}(D')}{2(E(D') + a)^2}.$$

Thus, the true positive rate satisfies the following condition:

$$\Pr[D' > -a] \geq 1 - \frac{\text{var}(D')}{2(E(D') + a)^2} \geq 1 - \frac{(\sigma + \sigma_x)^2}{2M(\mu_x + 2a)^2}.$$
With random timing perturbation with expected value 0 and variance $\sigma_d^2$, the above inequality becomes
\[
\Pr[D' > -a] \geq 1 - \frac{(\sigma + \sigma_x + \sigma_d)^2}{2M(\mu_x + 2a)^2}.
\] (3.2)

Similarly, the false positive rate that we falsely identify a normal flow as the stepping stone flow satisfies the following condition:
\[
\Pr[D' > 0] \geq 1 - \frac{\text{var}(D')}{2E(D')} \geq 1 - \frac{(\sigma + \sigma_x)^2}{2M\mu_x^2}.
\] (3.3)

Equations 3.2 and 3.3 give lower bound for single-bit true positive rate and false positive rate. The values of $\mu_x$ and $\sigma_x$ depend on packet timings, chaff, and the maximum delay $\Delta$. These values can be obtained empirically. Generally speaking, more chaff and larger $\Delta$ will increase $\mu_x$ but decrease $\sigma_x$. As a result, both the true positive rate and the false positive rate will increase.

### 3.3.3 The Algorithm for Quantization Based Watermark

Similarly, we use random variables $X$ and $T$ to denote the impacts on the IPDs to embed one watermark bit by our algorithm and by random timing perturbation, respectively. We have $E(X) = \mu_x$, $\text{var}(X) = \sigma_x$, $E(T) = 0$, $\text{var}(T) = \sigma_d$. The new average IPD is then $ipd'_{\text{avg}} = \frac{1}{M} \sum_{i=1}^{M} (ipd_i + X_i - T_i)$. In order to decode the embedded watermark bit, the combined effect of $X$ and $T$ should still keep $ipd'_{\text{avg}}$ within the same range of $(-S/2, S/2]$. Let $Z = \frac{1}{M} \sum_{i=1}^{M} (X_i - T_i)$. The true positive rate is then $\Pr[|Z| \leq S/2]$.

Using Chebyshev inequality, from
\[
\Pr[|Z - E(Z)| \geq S/2 - E(Z)] \leq \frac{\text{var}(Z)}{(S/2 - E(Z))^2}
\]
we have
\[
\Pr[Z \geq S/2] = \Pr[Z - E(Z) \geq S/2 - E(Z)] \leq \frac{\text{var}(Z)}{2(S/2 - E(Z))^2}
\] (3.4)
and from
\[
\Pr[|Z - E(Z)| \geq S/2 + E(Z)] \leq \frac{\text{var}(Z)}{(S/2 + E(Z))^2}
\]
we have
\[
\Pr[Z \leq -S/2] \leq \frac{\text{var}(Z)}{2(S/2 + E(Z))^2}.
\] (3.5)

---

1For simplicity, we do not consider the small probability when $ipd'_{\text{avg}}$ falls into the ranges such as $(3S/2, 5S/2]$, in which the watermark bit can still be decoded.
Combining equations 3.4 and 3.5 together, we may derive the following relation for the true positive rate:

\[
\Pr[|Z| \leq \frac{S}{2}] \geq 1 - \frac{\text{var}(Z)}{2(S/2 - E(Z))^2} - \frac{\text{var}(Z)}{2(S/2 + E(Z))^2} \\
\geq 1 - \frac{2(\sigma_x + \sigma_d)^2}{M(S - 2\mu_x)^2} - \frac{2(\sigma_x + \sigma_d)^2}{M(S + 2\mu_x)^2}.
\]

(3.6)

Now we discuss the false positive rate when a normal flow is falsely identified as a stepping stone flow. The new average IPD is changed to \(Z = \frac{1}{M} \sum_{i=1}^{M} (ipd_i + X_i)\). For simplicity, we only consider \((-S, S]\), where the watermark bit is decoded correctly in \((0, S]\). So the false positive rate is \(\Pr[Z > 0]\). Using Chebyshev inequality, we can compute

\[
\Pr[Z > 0] \geq 1 - \frac{\text{var}(Z)}{2E(Z)^2} \geq 1 - \frac{(\sigma + \sigma_x)^2}{2\mu_x^2},
\]

(3.7)

where \(\sigma^2\) is the variance of a normal flow.

### 3.3.4 Computation Complexity

In this subsection, we investigate the time complexity of the packet mapping and the flow correlation algorithms.

In the packet mapping procedure, for each packet in the upstream flow \(f\), the number of packets to be checked in the downstream flow depends on the number of chaff packets inserted and the timestamp difference between adjacent packets. Suppose the average packet arrival rate in the upstream flow is \(\lambda_1\), and the chaff packets insertion rate is \(\lambda_c\). Then on average a mapping set contains \((\lambda_1 + \lambda_c)\Delta\) mapping packets. However, since heuristics are used in the mapping procedure, the number of packets checked for each mapping set is usually smaller. When \(\lambda_c\) and \(\Delta\) are not related to the number of packets in \(f\), the complexity of the mapping procedure is \(O(n)\).

For the correlation algorithms to decode the best watermark, the greedy algorithm (Section 3.2.5) has a complexity of \(O(n)\), since it only forms one candidate mapping. Algorithm 2 (Section 3.2.6) has to enumerate all possible mapping packet for the embedding packets. Suppose in the second step, there are \(k\) embedding packets \(p_{e_1}, \ldots, p_{e_k}\). In the worst case, totally \(\prod_{i=1}^{k} |M(p_{e_i})|\) mapping packets have to be checked, where \(|M(p_{e_i})|\) represents the number of packets in a mapping set. So Algorithm 2 still has an exponential complexity. On the other hand, by using heuristics, Algorithm 3 (Section 3.2.7) only requires to access each mapping packet at most once. So it needs to check \(\sum_{i=1}^{k} |M(p_{e_i})|\) number of packets in the worse case. For the quantization based watermark
scheme, Algorithm 4 (Section 3.2.8) also checks all the mapping packet at most once. Therefore
Algorithms 3 and 4 have linear complexity $O(n)$. Note Algorithms 2, 3 and 4 usually have much
better performance than their worst case scenarios.

3.4 Compensation for Packet Loss

Our assumptions in Section 3.1 require that every packet in an upstream flow appears in
a downstream flow for the packet mapping procedure to perform correctly. This means there is no
packet loss between the upstream and the downstream flows. In practice, packet loss does happen.
In this section, we investigate the impact of packet loss on our scheme and discuss possible methods
to tolerate packet loss during the correlation of stepping stone flows.

Since packet loss cannot be directly detected, we infer its occurrences based on packet
retransmissions. However, watermark embedding may also cause packet retransmissions. When
the delays of packets are longer than the initial value of retransmission timer, TCP will retransmit
unacknowledged packets, even though they are not lost. (Note that timing perturbations added by
attackers will not cause retransmissions if they are added above the TCP layer.) Due to TCP adaptive
retransmission mechanism, the retransmission caused by watermark will quickly disappear after the
TCP timer is set to a higher value. In the following, we assume packet loss is the only reason for
packet retransmissions. This can be done by intentionally delay some packets at the beginning of
the flow before we add the watermark to increase the TCP timer.

We first consider when packet loss happens in the upstream flow $f$ or in the suspicious
flow $f'$ where it can be detected by watermark encoder or decoder. To compensate for the packet
loss, the timing of certain packets in flow $f$ or $f'$ has to be adjusted to ensure correct execution of our
scheme. It does not mean we will actively change flow timing by delaying certain packets. Instead,
the timing adjustment only happens to the recorded timestamps used in watermark detection.

Whenever an encoder detects the retransmitted packet $p_j$ of packet $p_i$, it adjusts the packet
timestamps that will be used later for watermark detection. It first removes packet $p_i$’s timestamp
time $t_i$ from $f$, since $p_i$ will not be forwarded to the downstream flow $f'$. It also reduce the number
of packets received so far by 1, so that later watermark encoding can be executed as if there is no
retransmission. Second, according to Telnet and SSH protocols, the packets between $p_i$ and $p_j$ will
not be immediately forwarded to the next connection. They are delayed after $p_j$ in order to keep the
correct packet order. As shown in figure 3.3, if $p_1$ is lost and $p_4$ is its retransmission, $p_2$ and $p_3$ will
be forwarded to the next flow after \( p_4 \). So in \( f \), we delay the timestamps of the packets between \( p_i \) and its retransmission \( p_j \) so that these packets are right after \( p_j \). After the timestamp adjustment, we can use our scheme to detect the embedded watermark. Note that if \( p_i \) and the packets before \( p_j \) may be used for encoding watermark, the adjustment may affect the effectiveness of our scheme. In case the retransmission of a packet \( p_i \) happens more than once, we will only use the last one and remove all the previous duplicates.

![Figure 3.3: Retransmission over one stepping stone (no perturbation)](image)

If a decoder detects duplicated packets \( p'_i \) and \( p'_j \) in the suspicious flow \( f' \), it simply removes \( p'_j \)’s timestamp \( t'_j \) from \( f' \), since packet retransmission in a downstream flow will not affect watermark decoding. Similarly, if multiple retransmissions appear, we only use the first one and remove other duplicates.

If neither encoder nor decoder can detect packet loss, we will not be able to adjust packet timestamps to compensate for lost packets. Further research is needed to fully address the problem of packet loss.

### 3.5 Correlation Under Flow Split & Merge

Although our scheme is proposed to correlate stepping stone flows under timing perturbation and chaff, it may also be applicable to mitigate the impacts of other countermeasures. In this section, we discuss how our algorithms can be utilized on an extreme countermeasure called *flow split & merge*.

Network flows transmitted through a sequence of stepping stone hosts can be split or merged in transmission. For example, a stepping stone host may decide to mix multiple flows together and transmit them as a single flow. The new flow then contains the packets from all the mixed flows. To utilize such a countermeasure to conceal traces, an attacker may choose two hosts to do the split and merge operations. At a certain *split* host in the stepping stone connection chain,
the attacker split the outgoing flow into several sub-flows. Note that the split host could be the first host in the stepping stone chain. Each sub-flow contains only a part of the commands and data in the original flow, and is transmitted in a distinct path until it reaches a *merge* host. In order to launch the attack, all the sub-flows must come together at the merge host so that the “original” flow can be restored. Since each sub-flow only includes an incomplete copy of the commands and data in the original flow, it is difficult to correlate the restored flow with any of the sub-flows to trace the attack. The procedure to split and merge the stepping stone flows is illustrated in Figure 3.4.

![Figure 3.4: The illustration of flow split and merge](image)

Our scheme can mitigate the effect of flow split & merge and correlate the sub-flows with the restored flow. It is because in the restored flow, the packets from the sub-flows other than the sub-flow to be correlated can be simply seen as “chaff” packets. So our scheme should be able to correlate these flows together as long as the assumptions in Section 3.1 are still satisfied. Our scheme does not specifically target at flow split and merge. Nevertheless, because of the potential discussed above, we will investigate the applicability to flow split and merge through experiments.

### 3.6 Experiments

In this section, we experimentally evaluate the performance of our algorithms. In Section 3.6.1 we test the ability of our algorithms to correlate stepping stone flows when chaff packets and timing perturbation are both inserted by attackers. In Section 3.6.2, we test the effectiveness of our algorithms when they are used to identify stepping stone flows under flow split and merge.

#### 3.6.1 Flow Correlation under Chaff and Timing Perturbation

We evaluate the performance of our algorithms under chaff packets and timing perturbation using three metrics, including *true positive rate*, *false positive rate*, and *computation overhead*. 
We focus on Algorithms 2, 3 and 4. We also include the original probabilistic and quantization based watermark schemes in order to demonstrate the failure of the original schemes under chaff packets.

We evaluate our algorithms using 91 real SSH/Telnet flows from Bell Labs-1 Traces of NLANR [30]. All the flows have more than 1,000 packets, and the average packet arrival rate is 1.22 packet/second. For each flow, we first embed a 24-bit randomly generated watermark using the probabilistic or the quantization based watermark schemes. The watermark parameters for these two schemes are shown in Table 3.1 and Table 3.2. Each flow is then added with different timing perturbation and chaff packets. The timing perturbation is uniformly distributed with different maximum delays $\Delta = 0, 1, 2, 3, 4, 5, 6, 7, 8$ seconds. The chaff packets introduced are Poisson distributed with arrival rates $\lambda_c = 0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5$ packets per second. We believe such combinations on timing perturbation and chaff can cover most of possible attack scenarios. For simplicity, the maximum delay in our algorithms is set the same as the maximum delay of each type of timing perturbation.

Table 3.1: Parameters for probabilistic scheme

<table>
<thead>
<tr>
<th>Watermark</th>
<th>24 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy ($M$)</td>
<td>4</td>
</tr>
<tr>
<td>IPD distance ($d$)</td>
<td>1</td>
</tr>
<tr>
<td>Threshold of hamming dist.</td>
<td>7</td>
</tr>
<tr>
<td>WM delay ($a$)</td>
<td>600ms</td>
</tr>
</tbody>
</table>

Table 3.2: Parameters for quantization scheme

<table>
<thead>
<tr>
<th>Watermark</th>
<th>24 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Redundancy ($M$)</td>
<td>8</td>
</tr>
<tr>
<td>Threshold of hamming dist.</td>
<td>7</td>
</tr>
<tr>
<td>Quantization step ($S$)</td>
<td>600ms</td>
</tr>
</tbody>
</table>

**True positive rate.** To evaluate the true positive rate, for each setting of parameters $\Delta$ and $\lambda_c$, we calculate the correlation between each original flow and its perturbed and chaffed flows. For a specific $\Delta$ and $\lambda_c$, there are 91 flow pairs to be correlated. We then compute the average true positive rate for every setting of $\Delta$ and $\lambda_c$. Here we only show two figures that illustrate the tendency of the true positive rate changing with different values of timing perturbations and changing with different amounts of chaff packets.

Figure 3.5(a) shows the true positive rate changing with $\lambda_c$ when $\Delta = 7$ seconds for real flows. The true positive rates of the probabilistic and quantization based watermark schemes fall near 0 as soon as chaff packets appear, which shows that chaff can destroy current watermark schemes completely. The true positive rates of our algorithms increase with the number of chaff packets and achieve 100% when $\lambda_c \geq 2$. When there is no chaff, the true positive rate is pretty low because of huge amount of timing perturbation. The existence of chaff packets helps the true
Figure 3.5: (a) True positive rate changing with $\lambda_c$, $\Delta = 7s$; (b) True positive rate changing with $\Delta$, $\lambda_c = 3$.

Figure 3.6: False positive rate in the second case: (a) changing with $\lambda_c$, $\Delta = 7s$; (b) changing with $\Delta$, $\lambda_c = 3$.

positive rate by improving the best watermark we can decode from the chaffed flows.

Figure 3.5(b) shows the true positive rate changing with the maximum timing perturbation $\Delta$ when $\lambda_c = 3$. In this case, our algorithms can successfully detect the stepping stone flow most of the time, while the basic watermark schemes only have very low true positive rates.

**False Positive Rate.** Here we discuss the false positive rate in two cases. First, we assume there is only one flow can be added with timing perturbation and/or chaff packets. All other suspicious flow are normal flows. Second, we test the situation where multiple flows can be added with perturbation and chaff. For example, this can happen when multiple attackers launch the stepping stone attacks and try to hide their traces at the same time. In this case, a false positive also includes the situation when a flow in attack A is falsely identified as a downstream flow in attack B.
In the first case, the false positive rate is computed by correlating the upstream stepping stone flow with normal flows. Obviously the false positive rate will not change with different amount of timing perturbation and chaff packets. For the flows tested, the false positive rate in this case is 0.

To evaluate the false positive rate (collision rate) in the second case, we test an extreme case. For each setting of parameters $\Delta$ and $\lambda_c$, we calculate the correlation between each watermarked flow and the perturbed and chaffed flows of other 90 flows. Totally there are 8,190 unrelated flow pairs to be correlated. We then compute the average false positive rate for each setting of $\Delta$ and $\lambda_c$. Figure 3.6(a) shows the false positive rate changing with $\lambda_c$ when $\Delta = 7$ seconds. Figure 3.6(b) shows the false positive rate changing with $\Delta$ when $\lambda_c = 3$. In these figures, the false positive rates of our algorithms increase with both $\lambda_c$ and $\Delta$. Therefore when multiple flows are added with perturbation and chaff, our algorithms may falsely attribute stepping stone flows. This obviously decrease the usefulness of our scheme. It is the trade-off in our scheme that both true positive rate and false positive rate increase with the amount of chaff and timing perturbation introduced.

**Computation Overhead.** To eliminate the bias of different implementation details, we evaluate computation overhead by counting how many times we have to access packets and retrieve their timestamps to decode watermarks. We distinguish the computation overhead between correlated and uncorrelated flow pairs, i.e., according to whether a flow is identified as a stepping stone flow or not. For each algorithm, we compute the average computation overhead for each setting of $\lambda_c$ and $\Delta$. We replace the basic watermark scheme with the packet mapping procedure, since determining packet matching is a critical and time consuming step. We also bound the execution of Algorithm 2 by setting a maximum overhead of 1,000,000.

Figures 3.7(a) and 3.7(b) show the computation overhead for correlated flows, in which Algorithm 4 has the smallest values. Algorithm 2 shows bumps in its curve, especially when there exist small number of chaff packets. This is because at the beginning, when more and more chaff packets are inserted, packet mapping sets grow bigger and bigger. Therefore the number of packets to be enumerate also increase and cause the computation overhead to raise quickly. However, when even more chaff packets are added, our optimization techniques begin to work and the correlation results can be determined without enumerations. Algorithm 3 also shows a smaller bump in figure 3.7(a) for a similar reason, except that the heuristics used significantly decreases computation overhead. Please note there are certain cases that algorithm 2 cannot finish within its bound of computation cost, which explains why sometimes it has lower true positive rate than Algorithm 3 in Figure 3.5(a).
Figure 3.7: Computation overhead: (a) for correlated flows changing with $\lambda_c$, $\Delta = 7s$; (b) for correlated flows changing with $\Delta$, $\lambda_c = 3$.

Figure 3.8: Computation overhead (a) for uncorrelated flows changing with $\lambda_c$, $\Delta = 7s$; (b) for uncorrelated flows changing with $\Delta$, $\lambda_c = 3$.

Figures 3.8(a) and 3.8(b) show the computation overhead for uncorrelated flows. It is worth noticing that the overhead for some algorithms is 0. This is because the execution of these algorithms depends on the packet mapping procedure. If the mapping procedure fails to find at least one mapping packet for any packet, we immediately return that this is not a stepping stone flow without further execution of any algorithms. We also find out that the computation overhead of algorithm 2 can reach the maximum bound when the amount of chaff and/or timing perturbation is big. Although we try to improve the computation efficiency, Algorithm 2 still cannot handle large amount of chaff and timing perturbation. On another hand, Algorithm 3 and 4 have much smaller overhead.

\footnote{In order to draw the figures in logarithm scale, we change 0 to 1.}
Overall Performance

From all the experiment results, Algorithms 3 and 4 show overall the best performance in terms of true positive rate, false positive rate and computation overhead. The experiments demonstrate that our approach based on the probabilistic and the quantization watermark schemes is capable of correlating stepping stone flows under the existence of not only timing perturbation, but also chaff packets.

Correlation Under Flow Split & Merge

In this subsection, we experimentally evaluate our algorithms’ capability under flow split & merge by correlating a split flow with its merged (restored) flow. Here we focus on Algorithms 3 and 4. The watermark parameters are the same as shown in Tables 3.1 and 3.2. Synthetic flows generated by Tcplib are used to make sure in a split flow there will be enough packets to completely embed a watermark.

We first generate a synthetic flow using Tcplib and split it into \( k \) sub-flows. We choose one sub-flow to embed a watermark. Then we add uniformly distributed timing perturbation to all sub-flows independently. The maximum delay of the timing perturbation is 3 seconds. The perturbed sub-flows are then merged together to restore the original flow. We then try to correlate the watermarked sub-flow with the restored flow.

In flow split, we first use a parameter \( \text{block size} \ b \) to specify how a flow to be split into \( k \) sub-flows. We assign the first \( b \) packets to the first sub-flow, assign the next \( b \) packets to the second sub-flow, and so on. After each sub-flow has \( b \) packets, we start from the first sub-flow again.
example, if \( k = 2 \), \( b = 1 \), there will be two sub-flows: \((p_1,p_3,p_5,\ldots)\) and \((p_2,p_4,p_6,\ldots)\). We could also randomly split the flow into \( k \) sub-flows. In this case every packet has \( 1/k \) probability to be split in each sub-flow. In order to completely embed the watermark to any sub-flow, we make sure every sub-flow will have enough number of packets. Therefore the number of packets in the original flow has to change with the parameter \( k \).

In our experiment, we generate 1000 synthetic flows. The true positive rate is measured by computing the average result when we correlate a sub-flow with its corresponding restored flow. The false positive rate is measured by computing the average result when we correlate a sub-flow with all other restored flows but its corresponding restored flow.

Figure 3.6.1 shows the true positive rate when we set block size \( b \) to 1, and change the number of sub-flows \( k \) to be split. We add the watermarks to the first, the middle, or the last sub-flow, respectively. In this case, the number \((k-1)\) is the ratio between chaff and normal packets. For all cases, Algorithm 4 shows very good true positive rate. However, the true positive rate for Algorithm 3 decrease with \( k \). We find out the reason is in the probabilistic watermark scheme. Recall that a watermark is embedded through delaying certain packets by the amount of \( a \) to make the IPD difference become negative or positive. But we cannot always change the IPD as wanted. The probability that a watermark cannot be successfully embedded increases with IPDs and decrease with \( a \). When flows are split as above, the IPDs in a sub-flow will increase with the number of sub-flows \( k \). Therefore the true positive rate constantly decreases.

The false positive rate in the experiment remains at 0. We have also tested randomly split and obtained similar results. From the experiment, the algorithm based on the quantization watermark is more capable of dealing with flow split & merge than the algorithm based the probabilistic watermark, since the probabilistic watermark scheme is affected by the IPDs used to embed watermarks.

### 3.7 Summary

Tracing attacks through stepping stones is a difficult problem. Encryption, timing perturbation and chaff packets can all be employed by intruders to hide their identities. To defeat these countermeasures, we propose a correlation scheme based on packet mapping and active timing based watermark schemes. To recover the watermark under chaff packets, we choose to use the best watermark to improve the true positive rate. We have developed a series of algorithms to com-
pute the best watermark for two watermark schemes. Heuristics were used to reduce computation overhead. The experimental evaluation has demonstrated that our scheme is capable of correlating stepping stone flows when both timing perturbation and chaff packets are introduced by attackers. We have compared the effectiveness and the efficiency of our algorithms and find out Algorithm 3 and 4 have overall the best performance. We have also investigated how our approach can be extended to mitigate the countermeasure of flow split & merge. Through experiment, we show the sub-flow can still be correlated to the restored flow after flow split & merge, especially by using Algorithm 4.
Chapter 4

On the Secrecy of the Quantization Based Watermark Scheme

Starting from this chapter, we begin to investigate the secrecy issues of the active watermark schemes. As we have discussed earlier, attackers may put serious threats on the active schemes since the flow timing is changed. We need to fully understand such threats and then deter them with security enhancements.

In [59], Wang et al. have proposed a novel idea to correlate a known stepping stone flow with its possible downstream flows. In this approach, a digital watermark is embedded into the known stepping stone flow through manipulating the inter-packet delays (IPD). By quantizing the selected IPDs into certain values, which are multiples of a parameter quantization step, watermark bits 0 or 1 can be embedded. The changes on packet timing will then propagate into all its downstream flows. We should be able to detect the watermark embedded in all the downstream flows, but nowhere else. Through theoretical analysis and experimental evaluation, the authors have shown that this scheme is robust to limited amount of random timing perturbations.

However, their research on the 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 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 [59, 58] 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 here is to investigate how an attacker being traced by a timing based active watermark system can detect and recover the watermark parameters, using the technique proposed in [59] 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 watermark scheme?

In this chapter, we propose an attack scheme that compromises timing based active watermark 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., true positive & false positive rates) 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 watermark scheme, it can potentially be extended to compromise other timing based active watermark 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 chapter is organized as follows. Section 4.1 presents an overview of the proposed attack. Section 4.2 discusses our approach for inferring unknown watermark parameters and recovering/duplicating watermark. Section 4.3 describes our approach for quickly detecting the existence of (unknown) watermarks. Section 4.4 presents our experiments used to validate the proposed approaches. Section 4.5 concludes the research results we have achieved.

### 4.1 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 *true positive rate*) or duplicate the watermark in benign flows (i.e., in-
crease the *false positive rate*). In both cases, the attacker can successfully defeat the tracing system. In this chapter, 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.

![Diagram](image)

**Figure 4.1:** Overview of our attacks to the watermark scheme

As discussed in Section 1.1, when a series of stepping stone hosts are used, an 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 and vice versa. The attacker can 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. If having total control of stepping stone computers, the attacker may be able to process the packets and retrieve the accurate packet delays in network layer. An easier way that requires less privilege is to collect the packet delays in application layer. For example, as illustrated in Figure 4.1, 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 $d = t^r - t^s$. Although the delays collected in application layer are not as precise as those collected in network layer, we will see later that they can still give critical information to the attacker. 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 [34] 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.
4.1.1 Normal Network Delays

We examined the one-way packet transit delays between computers in the Internet. There are literatures that have investigated Internet packet dynamics including one-way or round-trip delays before (e.g., [26, 28, 35]). However, no model has been established that can fit all types of communications. Therefore we have launched our own experiments to understand the one-way packet transit delays for interactive SSH connections on the Internet. In these experiments, we employed various computers joined in the PlanetLab community from the U.S. and from China as stepping stones. We slightly modified the SSH client program in OpenSSH[31] suite so that it pads the current system time as the sender’s timestamps in the packets for both the upstream and the downstream directions of communications. Such a modification is very simple and the modified SSH client can be executed with only user privilege. When receiving a padded packet, the SSH client also extracts the timestamp padded and computes the packet delays. Therefore in a stepping stone chain, we can acquire packet delays between any adjacent stepping stones. This scenario can be easily follow by any attackers who want to derive the packet delay information.

The packet delays from one computer in MIT and from one computer in Hong Kong to our computer before the clock skews are removed are shown in figures 4.2(a) and 4.2(b). We can see that both of the remote computers are around 0.003% faster than our computers, and the packet delay variation of the host in MIT is smaller than those of the host in Hong Kong. Figure 4.3 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:

\[ f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \] (4.1)

In our experiments, such approximations pass Kolmogorov-Smirnov[12] 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 \( \mu \) and variance \( \sigma^2 \)) of the delay distribution. To distinguish such packet delays from the delays introduced by watermark embedding, hereafter we call them normal network delays.
4.1.2 Watermark Delays

When a timing based active watermark 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 hypothesize 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 use experiment to validate our hypothesis. We randomly generate 100 synthetic traces using Tcplib and compute the watermark delays for watermarks with all 0’s, all 1’s, or randomly generated watermark with equal probability of 0’s and 1’s, respectively. The results are shown in Figure 4.5. The quantization step $S = 400$ms. Each number on $x$-axis represents a time interval of 50ms, e.g., $x = 16$ stands for the interval of $[749, 800)$ ms. This result passes Kolmogorov-Smirnov goodness-of-fit test for uniform distribution with a significance level of 0.05. Therefore
we can assume watermark delay is uniformly distributed. Its probability density is:

\[
f_Y(x) = \begin{cases} 
\frac{1}{2S} & 0 \leq x < 2S \\
0 & \text{otherwise} 
\end{cases},
\]

(4.2)

where \( S \) is the quantization step.

Figure 4.5: Watermark delay distributions

4.1.3 The Combination of Watermark Delay and Normal Network Delay

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:

\[
f_Z(x) = \int_{-\infty}^{+\infty} f_X(y) f_Y(x-y) dy = \frac{1}{2\sigma S \sqrt{2\pi}} \int_{x-2S}^{x} e^{-\frac{(y-\mu)^2}{2\sigma^2}} dy,
\]

(4.3)

where \( f_X \) is the probability density of a normal distribution with mean \( \mu \) and variance \( \sigma^2 \). \( S \) is the quantization step. The mean \( \mu \) 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 4.4 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.)

4.1.4 Attacking the Quantization Based Watermark Scheme

In summary, we will investigate the following problems using the packet delays between adjacent stepping stone computers.

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 watermark 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.

4.2 Inferring Watermark Parameters

As discussed earlier, an attacker can observe the packet delays and obtain the distribution (i.e., $\mu$ and $\sigma$). 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

$$f(x, \theta) = (1 - \theta) f_X(x) + \theta f_Z(x),$$  (4.4)
where \( \theta \) 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 \( \theta \), then identify the packets delayed due to watermark, and finally recover the remaining watermark parameters or duplicate the watermark (without knowing all the parameters).

### 4.2.1 Estimating the Quantization Step \( S \) and the Proportion Parameter \( \theta \)

We propose to use Expectation Maximization (EM) algorithm [8] to estimate the quantization step \( S \) and the proportion parameter \( \theta \) 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 \( \Psi \) is \( \mathcal{L}(\Psi) = \prod_{j=1}^n f(d_j \mid \Psi) \). The EM algorithm estimates \( \Psi \) 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 \( \log \mathcal{L}(\Psi) = \sum_{j=1}^n \log \left( (1 - \theta) f_X(d_j) + \theta f_Z(d_j) \right) \). 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 \( \Psi \) is transformed into

\[
\log \mathcal{L}_c(\Psi) = \sum_{j=1}^n (1 - z_j) \log f_X(d_j) + \sum_{j=1}^n z_j \log f_Z(d_j)
\]

\[
+ \sum_{j=1}^n (1 - z_j) \log(1 - \theta) + \sum_{j=1}^n z_j \log \theta. \tag{4.5}
\]

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 \( i = 0, 1, 2, \ldots \), the algorithm first performs the E-step to calculate the expectation of the log likelihood \( Q(\Psi \mid \Psi^{(i)}) = E_{\Psi^{(i)}} \{ \log \mathcal{L}_c(\Psi) \mid \mathbf{d} \} \). Then the algorithm performs the M-step to maximize \( Q(\Psi \mid \Psi^{(i)}) \) with respect to \( \Psi \), that is, find \( \Psi^{(i+1)} \) such
that $Q(\Psi^{(i+1)}|\Psi^{(i)}) \geq Q(\Psi|\Psi^{(i)})$ for all possible values of $\Psi$. It terminates when the difference between $L(\Psi^{(i+1)}) - L(\Psi^{(i)})$ is small enough.

In our case, the E-step simply computes the current conditional expectation of $z$ using the observed packet delays $d$. The $i$-th round value of $z^{(i)}$ is $z^{(i)}_j = \theta^{(i)} f_X(d_j)/f(d_j|\Psi^{(i)})$, $1 \leq j \leq n$. With the value of $z^{(i)}$, 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) [27], in which the M-step is modified to find $\Psi^{(i+1)}$ such that $Q(\Psi^{(i+1)}|\Psi^{(i)}) \geq Q(\Psi|\Psi^{(i)})$. More specifically, we use the GEM algorithm based on one Newton-Raphson step [42] in the M-step to compute $\Psi^{(i+1)}$ by $\Psi^{(i+1)} = \Psi^{(i)} + \delta^{(i)}$, where

$$
\delta^{(i)} = -\left[\frac{\partial^2 Q(\Psi|\Psi^{(i)})}{\partial \Psi \partial \Psi^T}\right]_{\Psi=\Psi^{(i)}}^{-1} \left[\frac{\partial Q(\Psi|\Psi^{(i)})}{\partial \Psi}\right]_{\Psi=\Psi^{(i)}}.
$$

(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 $\theta$. For this $\theta^{(0)}$, the average packet delay is $2\mu + S$. Therefore we simply use the average value of the packet delays $d$ as the initial value $S^{(0)}$ since $\mu$ is small compared with $S$ in practice.

### 4.2.2 Identifying Watermark Delayed Packets

We can determine the probability densities $f_Z$ and $f$ with the estimated $S$ and $\theta$. 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 [12] in our decision process to minimize the cost of wrong decisions. Let $\lambda_{ij}$ 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 $\lambda_{ij}$ ($i,j = 1, 2$). According to the Bayesian decision theory [12], we can decide a packet $p_i$ as a watermark delayed packet if its packet delay $d_i$ satisfies

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

Here $\theta$ and $1 - \theta$ can be seen as the priori probabilities for watermark delays and normal network delays, respectively. From inequality 4.6 we can numerically compute a threshold value $\tilde{d}$. Packet $p_i$ is identified as a watermark delayed packet when $d_i \geq \tilde{d}$.

We may also develop a threshold that works for the same $S$ but different $\theta$, and thus avoid computing a new threshold for each $\theta$ when the same $S$ is used for multiple flows. Specifically, we
use the minimax [12] solution for inequality 4.6 to achieve good performance over all values of $\theta$. Briefly speaking, the minimax solution searches for the decision threshold with which the maximum risk is minimized. Figure 4.4 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.

4.2.3 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, and then 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 [59]) 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.

**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 4.2.1, and the embedding packets \(p_{e_1}, \ldots, p_{e_L}\) have been identified using the approach discussed in Section 4.2.2. 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 \(a_i(2 \leq i \leq L)\) are all integers. Since an even (or odd) \(a_i\) represents that \(w_i\) is 0 (or 1), 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 \(ipd_2, ipd_3, \ldots, ipd_L\). It is easy to see that \(\gcd(ipd_2, \ldots, ipd_L) = y \cdot S\), where \(y = \gcd(a_2, \ldots, a_L)\). 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 \(\gcd(ipd_2, \ldots, ipd_L)\) to get

\[
\frac{ipd_i}{\gcd(IPDs)} = \frac{a_i \cdot S}{y \cdot S} = \frac{a_i}{y}, \quad \text{for} \quad 2 \leq i \leq L.
\]

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.

**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 4.2.2, 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 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 $S = 400\text{ms}$, $M = 8$, $L = 24$, and normal network delays with $\mu = 0$, $\sigma = 10\text{ms}$. Figure 4.6 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) [20] 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 4.6 is shown in Figure 4.7. 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 \( \sigma \) estimated in Section 4.1 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 \( \sigma \), 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, 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.

**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 4.2.2. 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 $\mu$ and variance $\sigma^2$, 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 $P_c = \frac{\sqrt{2\sigma}}{S}$. Let $q$ be the number of packets between two adjacent identified embedding packets. The probability that exactly $i$ IPDs are close is $P_c(i) = \binom{q}{i} \left(1 - P_c\right)^{q-i}$. 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 4.8 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 4.8. For example, when $L = 24$ and $h = 5$, the original false positive rate is only $\sum_{i=0}^{h} \binom{L}{i} \left(\frac{1}{2}\right)^L = 0.33\%$. If the 1-bit duplication rate
is $P_{dup} = 0.9$, the false positive rate dramatically increases to 97.23%.

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 4.9. Compared with Figure 4.8, the duplicate rates are lower and decrease more quickly with increasing
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.

Case V

We do not have a 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.

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.

4.3 Detecting Watermark Existence

In Section 4.2, 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.

4.3.1 The Watermark Capability and the Minimum Number of Watermark Delayed Packets

Two things define the usefulness of a watermark. First, the true positive rate $T_p$ defines how well a stepping stone flow can be identified. A true positive rate is only meaningful when considered under certain amount of timing perturbation. Second, the false positive rate $F_p$ specifies how likely an arbitrary flow may be wrongly identified as the stepping stone flow. Here we name the true positive 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

$$F_p = \sum_{i=0}^{h} \binom{L}{i} \left( \frac{1}{2} \right)^L,$$

where $h$ is the threshold of hamming distance. Given a false positive rate, we can derive the relation between $L$ and $h$. In Figure 4.10(a), we give out the maximum values of $h$ that fulfill $F_p \leq 1\%$ for different values of $L$.

Now we consider the true positive rate. Assuming the timing perturbation, which comes from attackers and normal network delays, is distributed with mean 0 and variance $\sigma_p^2$, the probability that a single watermark bit can be correctly decoded is

$$p_t \approx 2 \Phi \left( \frac{S \sqrt{M}}{2 \sqrt{2 \sigma_p}} \right) - 1,$$

where $\Phi()$ is the cumulative distribution function of the normal distribution. For an $L$-bit watermark, the true positive rate is the probability that no more than $h$ bits are incorrectly decoded, i.e.,

$$T_p = \sum_{i=0}^{h} \binom{L}{i} p_t^{L-i} (1 - p_d)^i.$$

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 true positive 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 $T_p \geq 95\%$, the timing perturbation is uniformly distributed in $[0, 2]$ seconds (i.e., $\sigma_p = 2000/\sqrt{12}$ ms), and $S = 1000$ ms, Figure 4.10(b) shows the
Figure 4.10: The watermark capability for different watermark parameters

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 4.10(c) lists the minimum $M \times L$ for different false positive rates and true positive 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 $\theta$ of the watermark delayed 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 $\theta$ 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 $\theta$. 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.

4.3.2 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 is 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 $\theta$ of the watermark delayed packets, which is shown in equation 4.4. 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 $\theta$. When $\theta$ 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 [14] concept. Each time an observation (i.e., a packet delay) is obtained, we make one of the 3 decisions about the hypothesis on $\theta$: (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 [57], which minimizes the expected number of observations needed to make the decision.

Intuitively, we can choose a threshold $\theta'$ 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 [57], 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 $\theta_0$ and $\theta_1$ is a critical issue, and prior knowledge has to be used. We
choose $\theta_0$ 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 $\theta_1$ 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 total number of packets in their flows and compute $\theta_1$.

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

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

(4.8)

where $f$ is specified in equation 4.4. 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 $\alpha$ and false positive rate $\beta$ 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 $\theta$ 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 4.1 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 $\theta_0$ and $\theta_1$, 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 $(\theta_0, \theta_1)$.

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>0</th>
<th>0.05</th>
<th>0.1</th>
<th>0.15</th>
<th>0.2</th>
<th>0.25</th>
<th>0.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>output(avg)</td>
<td>0</td>
<td>0</td>
<td>0.8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>packet #</td>
<td>30</td>
<td>56</td>
<td>438</td>
<td>68</td>
<td>26</td>
<td>17</td>
<td>13</td>
</tr>
</tbody>
</table>

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 4.1). 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.

4.4 Experiments

To evaluate the performance of our approaches to attack the watermark scheme, we have launched a series of experiments. From Sections 4.4.1-4.4.5, we test the ability to recover and duplicate watermarks. In Section 4.4.6 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 4.1 and estimate mean \( \mu \) and variance \( \sigma^2 \). All other traces from the same host will directly use \( \mu \) and \( \sigma \) without estimating their own parameters.

In our experiment, different values of \( S = 400, 600, 800 \text{ms}, M = 1, 4, 6, 8 \) 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.

4.4.1 Estimating \( S \) and \( \theta \)

After a watermark is embedded, we first evaluate our GEM algorithm, which estimates the quantization step \( S \) and the mixture ratio \( \theta \). The result of \( S \) is shown in Table 4.2, in which \( S' \) is the estimated value. Since the value of \( \theta \) changes with other parameters, here we only pick three different values of \( \theta \) and show the estimation result in Table 4.3. We can see the estimation of \( S \) and \( \theta \) are very close to the real values. Having the estimated \( S \) and \( \theta \), we then apply the Bayes decision rule described in Section 4.2.2 to identify the watermark delayed packets. Zero-one loss is used for \( \lambda_{ij} \) in Equation 4.6.
Table 4.2: GEM estimation of $S$

<table>
<thead>
<tr>
<th>$S$</th>
<th>400</th>
<th>600</th>
<th>800</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean of $S'$</td>
<td>389.1</td>
<td>587.8</td>
<td>782.9</td>
</tr>
<tr>
<td>std dev of $S'$</td>
<td>18.44</td>
<td>17.67</td>
<td>24.86</td>
</tr>
</tbody>
</table>

Table 4.3: GEM estimation of $\theta$

<table>
<thead>
<tr>
<th>$\theta$</th>
<th>0.1578</th>
<th>0.2366</th>
<th>0.4723</th>
</tr>
</thead>
<tbody>
<tr>
<td>mean of $\theta'$</td>
<td>0.1573</td>
<td>0.2367</td>
<td>0.4694</td>
</tr>
<tr>
<td>std dev of $\theta'$</td>
<td>0.0021</td>
<td>0.0023</td>
<td>0.0041</td>
</tr>
</tbody>
</table>

4.4.2 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 4.4 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.

4.4.3 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 4.5 and 4.6. 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.
Table 4.4: Watermark recovery rate for Case I

<table>
<thead>
<tr>
<th>$L$</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = 400$</td>
<td>0.9723</td>
<td>0.9679</td>
<td>0.9670</td>
</tr>
<tr>
<td>$S = 600$</td>
<td>0.9712</td>
<td>0.9729</td>
<td>0.9705</td>
</tr>
<tr>
<td>$S = 800$</td>
<td>0.9712</td>
<td>0.9733</td>
<td>0.9740</td>
</tr>
</tbody>
</table>

Table 4.5: Recovery rate for Case II (S-L)

<table>
<thead>
<tr>
<th>$L$</th>
<th>16</th>
<th>24</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = 400$</td>
<td>0.9317</td>
<td>0.9326</td>
<td>0.9401</td>
</tr>
<tr>
<td>$S = 600$</td>
<td>0.9323</td>
<td>0.9421</td>
<td>0.9355</td>
</tr>
<tr>
<td>$S = 800$</td>
<td>0.9341</td>
<td>0.9317</td>
<td>0.9460</td>
</tr>
</tbody>
</table>

Table 4.6: Recovery rate for Case II (S-M)

<table>
<thead>
<tr>
<th>$M$</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S = 400$</td>
<td>0.9310</td>
<td>0.9413</td>
<td>0.9319</td>
</tr>
<tr>
<td>$S = 600$</td>
<td>0.9421</td>
<td>0.9369</td>
<td>0.9309</td>
</tr>
<tr>
<td>$S = 800$</td>
<td>0.9395</td>
<td>0.9311</td>
<td>0.9413</td>
</tr>
</tbody>
</table>

4.4.4 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[7] 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 4.11 shows the duplicate rate changing with $S$ and $K$, and Figure 4.12 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.

4.4.5 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 4.13 and 4.14. 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 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 4.15 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.

### 4.4.6 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 $\theta$ 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, \( S = 1000 \text{ms} \) is used. We set \( \alpha = \beta = 0.05 \).

Table 4.7 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 4.16 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.
4.5 Summary

Timing based active watermark schemes are effective for tracing back through stepping stones or an anonymizing network. However, the schemes themselves are also under the threats of attackers. The quantization based watermark scheme is proposed to correlated stepping stone flows under timing perturbation. Because packet timing is actively manipulated, this scheme could be detected and compromised by intelligent attackers. In this chapter, we analyze the secrecy of a timing based active watermark scheme for tracing through stepping stones [61]. 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.
Chapter 5

On the Secrecy of the Probabilistic Watermark Scheme

In this chapter, we investigate the secrecy and security of the probabilistic watermark scheme. The probabilistic watermark scheme [60] is proposed to provide realtime watermark embedding functionality and improved watermark true positive rate and false positive rate than the quantization based watermark scheme. Because of the manipulation on packet timing, this scheme is also under the possible attacks of intelligent attackers. From the extra delays incurred by the watermark embedding, an attacker may try to figure out 1) whether an attacking flow is embedded with watermark or not; 2) how the watermark is embedded; and 3) how to defeat the watermark scheme by removing the watermark from the attacking flow and/or duplicating the watermark to other normal flows.

We adopt the techniques developed for the quantization based scheme in the previous chapter to attack the probabilistic scheme. Due to the difference between these two schemes, modifications need to be applied. We are still focusing on three problems:

1. How can an attacker that connects through stepping stones determine whether he/she is being traced by a timing based active watermark system as quickly as possible?

2. How can an attacker infer the watermark parameters and how much can be recovered?
3. How can an attacker duplicate a watermark to mislead the trace-back system? How well?

The probabilistic watermark scheme uses several parameters, which are listed in Table 5.1. These parameters are the target of our attacks.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>Extra watermark delay</td>
</tr>
<tr>
<td>( L )</td>
<td>The number of binary watermark bits</td>
</tr>
<tr>
<td>( M )</td>
<td>Degree of robustness (i.e., the number of IPDs to embed 1 bit)</td>
</tr>
<tr>
<td>( W = w_1 \ldots w_L )</td>
<td>The watermark. Each bit ( w_i ) is either 0 or 1</td>
</tr>
</tbody>
</table>

This chapter is organized as follows. In Section 5.1, we derive the relation between the watermark capability and the number of packets to be embedded with watermark. Section 5.2 discusses how to detect watermark existence using the SPRT algorithm. Section 5.3 discusses how to recover some of the important watermark parameters. Section 5.4 and Section 5.5 discuss how to defeat the watermark scheme by removing and duplicate using the recovered watermark parameters. Section 5.6 gives the experiment result. Section 5.7 discusses how the attacks can be easily done in realtime. Section 5.8 summaries our research results.

## 5.1 Capability and Packet Number

It has been shown in the last chapter that the watermark detection is related to the watermark capability, which is defined by the required false positive rate and the required true positive rate under given timing perturbation. Therefore in this section, we first investigate the relation between watermark parameters and the watermark capability for the probabilistic watermark scheme.

Suppose the required false positive rate is \( F_p \), the maximum random timing perturbation is uniformly distributed with a mean of 0 and a variance of \( \sigma^2_p \), and the required true positive rate is \( T_p \). We derive the relation between the watermark capability and the minimal number of packets \( N_{\text{min}} \) to be delayed by the watermark encoder. The number \( N_{\text{min}} \) is very critical to the watermark detection algorithm that will be discussed a little bit later.

In the probabilistic watermark scheme, the false positive rate is computed by

\[
F_p = \sum_{i=0}^{h} \binom{L}{i} \frac{1}{2^L} ,
\] (5.1)
where $h$ is the threshold of the Hamming distance, $L$ is the number of binary bits in the watermark. Using this equation, we can find the minimum of $h$ that can satisfy the required $F_p$ for different values of $L$. In other words, for any $F_p$, we can find multiple $L$ and $h$ pairs.

Now we discuss true positive rate. For a single watermark bit, its true positive rate under the given amount of timing perturbation is computed by

$$t_p = \Phi\left( \frac{a\sqrt{M}}{\sqrt{\sigma^2_{ipd} + \sigma^2_p}} \right),$$

(5.2)

where $\Phi(x)$ is

$$\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{u^2}{2}} du,$$

and $\sigma^2_{ipd}$ is the variance of IPD differences. It has been shown in Figure 2.3 that when multiple IPDs are used for one bit, the distribution of IPD differences is similar to Gaussian distribution with 0 mean. For an $L$-bit watermark, the overall true positive rate is computed by

$$T_p = \sum_{i=0}^{h} \binom{L}{i} t_p^L (1 - t_p)^i.$$

(5.3)

From this equation, we can derive the one-bit true positive rate $t_p$ that satisfies the required true positive rate for different values of $L$ and the minimal value of $h$ that we have derived for the required false positive rate. Then we can use the Equation 5.2 to further compute the minimum value of $M$ for each value of $L$ (by assuming the values of $\sigma_{ipd}$ and $a$). Therefore, we are able to compute the minimal value of $N_{\text{min}} = M \times L$ for any given true positive and false positive rates. This number represents the minimal number of packets to be delayed by the watermark.

Note multiple variables affect the watermark scheme in the above equations. Therefore in our analysis, we have investigated a variety of values of these variables. These values are shown in Table 5.1. We believe the values that have been tested can cover most of the possible situations that may happen in reality. We have computed the minimal values of $N_{\text{min}} = M \times L$ for all parameters listed. A part of the result is shown in Figure 5.1.

5.2 Watermark Detection Using the SPRT Algorithm

5.2.1 Packet Delay Distributions

Follow the discussion in the previous chapter, we first model the distributions of normal network delays and watermarked delays. We still use Gaussian distribution to approximate normal
Table 5.2: Parameters

| $\sigma_{ipd}$ | 200, 300, 400, 500, 600, 700, 800 |
| $\sigma_p$     | 200, 300, 400, 500, 600, 700, 800, 900, 1000 |
| $a$            | 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400 |

True positive: 0.89, 0.90, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99
False positive: 0.11, 0.10, 0.09, 0.08, 0.07, 0.06, 0.05, 0.04, 0.03, 0.02, 0.01

Figure 5.1: Minimum of $M \times L$

network delays as

$$f_X(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \quad (5.4)$$

Different from the quantization based scheme, all the packets manipulated in the probabilistic scheme will simply be delayed by $a$. Therefore for those packets delayed by the watermark scheme, their total delays is the combination of normal network delay and watermark delay:

$$f_Y(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-a-\mu)^2}{2\sigma^2}}. \quad (5.5)$$

For a flow embedded with watermark, suppose $\theta$ portion of the packets are delayed the watermark encoder. Therefore the packet delays in the entire flow is:

$$f(x, \theta) = (1 - \theta)f_X(x) + \theta f_Y(x). \quad (5.6)$$

Then attackers will try to detect the watermark existence by hypothesize testing the value
of $\theta$. Note the mean of normal network delay $\mu$ is not important to the attackers. Therefore we always set $\mu = 0$.

### 5.2.2 The SPRT Algorithm

Similar to attacking the quantization based watermark scheme, we use the Sequential Probability Ratio Test (SPRT) algorithm to detect watermark existence. On average, the SPRT algorithm requires the smallest number of observations before it makes a decision. Therefore the watermark existence can be detected more quickly.

In order to use SPRT, two parameters $\theta_0$ and $\theta_1$ need to be decided, which indicate the threshold ratios for network delays without and with watermark. The parameter $\theta_0$ is determined by how well the normal network delays fit the normal distribution. In order to set $\theta_1$, we use the minimal number of packets $N_{\text{min}}$ to be delayed by the watermark to satisfy certain watermark capability. The rational is that an attacker can control the number of packets in the attacking flow. So he/she can control the ratio between packets with and without the watermark delays, which is the value of $\theta_1$. After both $\theta_0$ and $\theta_1$ is determined, we use SPRT to detect the watermark existence. The procedure is shown in the following.

Each time we receive a packet $p_j$ and extract its packet delay $d_j$, we compute

$$ r_j = \log \frac{f(d_1, \theta_1) \cdots f(d_j, \theta_1)}{f(d_1, \theta_0) \cdots f(d_j, \theta_0)}. \quad (5.7) $$

Then we compare $r_j$ with two thresholds $A$ and $B$ ($B < A$). If $r_j \leq B$, we terminate SPRT and decide there is no watermark. If $r_j \geq A$, we terminate and decide a watermark is detected. Otherwise, we continue the testing and wait for the next packet. In order to deal with the situation when the encoder intentional leaves the beginning of the flow unwatermarked, the SPRT is modified so that it does not terminate when $r_j \leq B$. Instead, it clears all the previous result and restarts for the next packet.

We determine the values of variable used in SPRT as follows. First, as discussed earlier, an attacker infers $\theta_1$ from the minimal values of $N_{\text{min}} = M \times L$ by assuming the total number of packets in the stepping stone flow. The attacker will also need to assume the value of $\sigma_{ipd}$ and $\sigma_p$. Note that the watermark encoder does not know the values of $\sigma_{ipd}$ and $\sigma_p$ either, and it also needs these two values in order to choose $M$ and $L$ correctly to achieve the desired capability. Since both the attacker and the encoder have to guess the values of $\sigma_{ipd}$ and $\sigma_p$, in our later analysis we assume both of them guess correctly, so that it will be fair to both the attacker and the encoder. The attacker
has to assume the value of \( a \) as well. Following the discussion in the previous chapter, we may assume a maximal possible value of \( a \), i.e., the watermark delays will be directly noticeable for all bigger values of \( a \). However, if the actual \( a \) used by watermark encoder is much smaller than the guessed \( a \), then most of the result computed by \( f(d_1, \theta_1) \) could be 0. We may not be able to detect the watermark successfully in this case. Another way to choose \( a \) is to use a small \( a \) to make sure we can detect watermark even if actual \( a \) is small. (Note in this case, if actual \( a \) is very big, we may also get many 0 from \( f(d_1, \theta_1) \). We need to replace such values of 0 to certain bigger values, because the delays are much bigger than the \( a \) we choose for the SPRT algorithm.) In the experiment, we compare the SPRT results of using both small and large values of guessed \( a \).

In the experiment of SPRT, we choose \( \theta_0 = 0.02 \) and \( \theta_1 = 0.1 \). A part of the results are shown in Figure 5.2. Here we list both the SPRT detection rate (labeled as “result”) and the number of packets processed by SPRT before it detects the watermark (labeled as “sample”). We can see when large guessed \( a \) is used and the actual \( a \) is small, we will miss all the watermarks. However, if we use a small value of guessed \( a \), we have 100% detection rate for all the values of \( a \) we tested.

Since watermark detection rate only shows half of the story, we have also tested the false positive rate of the SPRT algorithm by directly applying the SPRT algorithm on normal flows without watermark. First, we set the standard deviation of normal network delay \( \sigma = 20 \), and test different \( a \) from 25 to 400ms. Only for \( a = 25 \)ms, we have a non-zero false positive rate of 1.72%. Second, we increase \( \sigma \) to 100ms, and test for the same values of \( a \). The non-zero false positive
result is shown in Table 5.3. We can see the false positive rate is still very small. Therefore when network flows follow the Gaussian distribution nicely, SPRT will have very good false positive rate.

<table>
<thead>
<tr>
<th>$a$</th>
<th>fp ($\sigma = 20\text{ms}$)</th>
<th>fp ($\sigma = 100\text{ms}$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>1.72%</td>
<td>2.34%</td>
</tr>
<tr>
<td>50</td>
<td>0</td>
<td>2.81%</td>
</tr>
<tr>
<td>75</td>
<td>0</td>
<td>2.03%</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>1.09%</td>
</tr>
<tr>
<td>125</td>
<td>0</td>
<td>0.47%</td>
</tr>
</tbody>
</table>

We are also interested in testing the false positive rate when the network delay variance is not obtained correctly in the SPRT algorithm. We fix the $\sigma$ used in SPRT at 20ms, while the real $\sigma$ varies from 20 to 70ms. The result is shown in Figure 5.3. In this case, the wrongly estimated network delay has big impact on the false positive rate on SPRT, especially when we use small value of guessed $a$. However, we do not expect $\sigma$ to be incorrectly estimate so much in practice.

![Figure 5.3: SPRT false positive when actual $\sigma$ is different from estimated $\sigma$](image)

Figure 5.3: SPRT false positive when actual $\sigma$ is different from estimated $\sigma$
5.3 Recover Watermark Parameters

5.3.1 Recover the Delay Value $a$

The most important watermark parameter is the delay value $a$. It can be estimated by the EM algorithm. The input to the EM algorithm is packet delays, which are a mixture of normal packet delays and watermarked packet delays. Suppose the ratio of watermark delayed packets is $\theta$, then we can represent the distribution of input packet delays as

$$f(x) = (1 - \theta)f_X(x) + \theta f_Y(x). \tag{5.8}$$

In the above equation, the parameters $\theta$ and $a$ are unknown and will be estimated. The EM algorithm estimate the parameters by maximizing the log likelihood of the above equation for all the packet delays $d_j$ received:

$$\log L(\Psi) = \sum_{j=1}^{n} \log ((1 - \theta)f_X(d_j) + \theta f_Y(d_j)),$$

where $\Psi$ present the parameters to be estimated, in our case, $\Psi = (a, \theta)^T$. In order to use the EM algorithm, we also introduce an additional array of variables $z_j$ for each packet delay $d_j$. The value of $z_j$ is 0 or 1, which specifies whether the corresponding $d_j$ is from distribution $f_X$ or $f_Y$. And the above log likelihood is further changed to a complete log likelihood

$$\log \mathcal{L}_c(\Psi) = \sum_{j=1}^{n} (1 - z_j) \log(1 - \theta) + \sum_{j=1}^{n} z_j \log \theta + \sum_{j=1}^{n} (1 - z_j) \log f_X(d_j) + \sum_{j=1}^{n} z_j \log f_Y(d_j). \tag{5.10}$$

EM algorithm begins with arbitrary values of $a$ and $\theta$. In the E-step (expectation), we use the packet delays $d_j$ to compute the expected value of $z_j$. In the M-step (maximization), we use the computed $z_j$ to update the values of $a$ and $\theta$. EM algorithm stops when the increase of $\log \mathcal{L}_c(\Psi)$ is very small (e.g., $< 10^{-4}$).

We have done simulations to test the EM algorithm. First, we test the EM algorithm when collateral delays (i.e., the extra delays incurred when a packet has to be pushed back by the delaying of its previous packet in order to maintain the packet order) are not considered in watermark embedding. The result of average values of $a$ estimated by the EM algorithm is shown in Table 5.4. The initial value of $\theta$ is 0.5, the initial value of $a$ is 100ms. The estimated $a$ in Table 5.4 is extremely close to the real values of $a$. Next, we include the collateral delays into consideration, and show the result of EM algorithm in Table 5.5. We can see that parameter $a$ can still be correctly estimated most of the time.
5.3.2 Identify Watermark Embedding Packets

After $a$ is determined, the distributions of watermark delays are determined. We will find a threshold $\bar{d}$ for the packet delays, so that a packet $p_i$ is identified as a watermark embedding packet when $d_i > \bar{d}$. Similar to the previous chapter, we choose to use the Bayesian decision rule to minimize the overall cost for wrong decisions.

5.3.3 Recover Other Parameters

Besides $a$, we find out that other watermark parameters including $L$, $M$, and the actual watermark $W$ embedded are difficult to be recovered. This is because all the packets are delayed by the same amount. Therefore theoretically there is no constraint on which packets can be used to embed which watermark bits.

5.4 Removing Watermark

In the section, we consider how attackers can use the knowledge about the watermark scheme to remove the watermark from the attacking flow.

In the probabilistic watermark scheme, a watermark cannot be removed if we further delay the packets identified as delayed by watermark. If we further delay these packets, we actually increase the strength of the watermark. To remove the watermark, what we will do is to delay
the first packets in the IPDs used to embed the watermark. However, since these packets are not delayed, we cannot correctly identify them.

A possible method to remove watermark is to delay all the packets except those packets identified as embedded with watermark delays. The disadvantage of this method is the high overhead for attackers. We validate this idea by simulation. Table 5.6 shows the true positive rate when the legitimate watermark decoder decode the watermarks after an attacker removes the watermark by delaying all the packets except the identified watermark embedding packets. Collateral delays are not considered here. We can see the watermark removal algorithm can dramatically destroy the watermark scheme.

Table 5.6: Watermark removal result

<table>
<thead>
<tr>
<th>a</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_p$</td>
<td>0.0580</td>
<td>0.0525</td>
<td>0.0544</td>
<td>0.0589</td>
<td>0.0558</td>
<td>0.0548</td>
<td>0.0540</td>
<td>0.0469</td>
</tr>
<tr>
<td>a</td>
<td>225</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>325</td>
<td>350</td>
<td>375</td>
<td>400</td>
</tr>
<tr>
<td>$T_p$</td>
<td>0.0467</td>
<td>0.0456</td>
<td>0.0473</td>
<td>0.0446</td>
<td>0.0419</td>
<td>0.0397</td>
<td>0.0383</td>
<td>0.0350</td>
</tr>
</tbody>
</table>

When collateral delays are considered, we get the watermark removal results as shown in Table 5.7. Although the true positive rates for small values of $a$ are slight higher than those when collateral delay is not considered, the watermark removal is still very successful.

Table 5.7: Watermark removal result - collateral delay

<table>
<thead>
<tr>
<th>a</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_p$</td>
<td>0.1696</td>
<td>0.1312</td>
<td>0.0561</td>
<td>0.0518</td>
<td>0.0585</td>
<td>0.0494</td>
<td>0.0507</td>
<td>0.0466</td>
</tr>
<tr>
<td>a</td>
<td>225</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>325</td>
<td>350</td>
<td>375</td>
<td>400</td>
</tr>
<tr>
<td>$T_p$</td>
<td>0.0506</td>
<td>0.0508</td>
<td>0.0491</td>
<td>0.0494</td>
<td>0.0460</td>
<td>0.0432</td>
<td>0.0419</td>
<td>0.0382</td>
</tr>
</tbody>
</table>

5.5 Duplicate Watermark into Normal Flows

Another way to defeat the watermark scheme is to duplicate watermark into normal flows to increase the false positive rate. For the probabilistic watermark scheme, duplicating the watermark to normal flows is easy. After the packets delayed by the watermark is identified, we simply delay the corresponding packet in a normal flow by $2a$. For example, we use the watermark decoder
to decode the watermark from unwatermarked flows after watermark duplication. The successful duplication rate when collateral delays are not considered is shown in Table 5.8. We can see that a watermark can be easily copied to normal flows.

Table 5.8: Duplication Result

<table>
<thead>
<tr>
<th>a</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dup</td>
<td>0.9264</td>
<td>0.9421</td>
<td>0.9554</td>
<td>0.9752</td>
<td>0.9736</td>
<td>0.9760</td>
<td>0.9835</td>
<td>0.9802</td>
</tr>
<tr>
<td>a</td>
<td>225</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>325</td>
<td>350</td>
<td>375</td>
<td>400</td>
</tr>
<tr>
<td>Dup</td>
<td>0.9843</td>
<td>0.9893</td>
<td>0.9868</td>
<td>0.9917</td>
<td>0.9926</td>
<td>0.9909</td>
<td>0.9917</td>
<td>0.9901</td>
</tr>
</tbody>
</table>

Table 5.9: Duplication Result - collateral delay

<table>
<thead>
<tr>
<th>a</th>
<th>25</th>
<th>50</th>
<th>75</th>
<th>100</th>
<th>125</th>
<th>150</th>
<th>175</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dup</td>
<td>0.3678</td>
<td>0.7000</td>
<td>0.9455</td>
<td>0.9562</td>
<td>0.9488</td>
<td>0.9570</td>
<td>0.9595</td>
<td>0.9512</td>
</tr>
<tr>
<td>a</td>
<td>225</td>
<td>250</td>
<td>275</td>
<td>300</td>
<td>325</td>
<td>350</td>
<td>375</td>
<td>400</td>
</tr>
<tr>
<td>Dup</td>
<td>0.9554</td>
<td>0.9529</td>
<td>0.9364</td>
<td>0.9512</td>
<td>0.9438</td>
<td>0.9661</td>
<td>0.9661</td>
<td>0.9479</td>
</tr>
</tbody>
</table>

If we also consider the effect of collateral delays, we can get the duplication results as shown in Table 5.9. Similar to watermark removal, only for very small values of $a$ the duplication result is not very good. For bigger values of $a$, the watermark can still be copied to normal flows.

5.6 Experiments

In the experiment, we use computers in the PlanetLab to test the performance of the attack to the probabilistic watermark scheme. A TCP connection between our computer and one computer in MIT is created. Packets are generated so that the IPDs between consecutive packets follow Tcplib telnet distribution. We record sender’s timestamps and receiver’s timestamps for all the packets, and computer the packet delays. Totally 41 flows each with 1000 packets are created. The watermark parameters used in the experiment are shown in Table 5.10.

When the attacker uses a guessed value of $a = 200$ms, the watermarks in all the flows can be successfully detected using the SPRT algorithm, i.e., the watermark detection rate is 100%.

The average of the estimated values of $a$ using the EM algorithm is 345.20ms, the estimation error is $\frac{400 - 345.20}{400} = 13.7\%$, which is much larger than the result in our simulation. This
Table 5.10: Parameters used in experiment

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>24</td>
</tr>
<tr>
<td>$M$</td>
<td>6</td>
</tr>
<tr>
<td>$a$</td>
<td>400ms</td>
</tr>
<tr>
<td>watermark</td>
<td>0101010101010101010101010101</td>
</tr>
<tr>
<td>$h$</td>
<td>5</td>
</tr>
</tbody>
</table>

is because the normal network delays in some of the real flows do not follow Gaussian distribution very well.

Having identified the watermark embedding packets, we then try to remove the watermark. After watermark removal, we use the legitimate watermark decoder to detect the watermark. The watermark true positive rate is only around 5.5%, i.e., the watermark removal rate is around 94.5%.

To test watermark duplication, we create 20 normal flows for each watermarked flow and duplicate the embedded watermark. After duplication, we use the original decoder to decode the watermark and obtain a watermark duplication rate of 72.5%.

In conclusion, the attacks to the probabilistic watermark scheme can also significantly deteriorate the performance of the watermark scheme.

### 5.7 Attack the Watermark Schemes in Realtime

An issue we have not addressed before in our attacking approaches to the watermark scheme is that the watermark parameter estimation and watermark removal is done after we have received all the packets in the flow. Therefore a valid argument is that how useful are the approaches if they can not be applied in realtime. In this section, we investigate how to deal with the realtime issue. We will develop realtime attacking algorithms and show that attackers can still successfully defeat the watermark schemes.

#### 5.7.1 Realtime Algorithm for The Probabilistic Watermark Scheme

It is easy to develop the realtime attacks to the probabilistic watermark scheme. First, we use the SPRT algorithm to detect any watermark existence. If in any point we decide we have seen
a watermark, we begin to estimate watermark parameters, and then remove the watermark from our attacking flow. The algorithm is described by Algorithm 1.

The result of realtime watermark detection and removal is shown in Table 5.11. We generated 100 synthetic flows. From the result, we can see that even in realtime, our attack approaches can still significantly affect the effectiveness of the probabilistic watermark scheme. We have found out that since all the watermark delayed packets are delayed by the same amount, even with partial number of packets, we can still get good estimation result about the value of $a$. Therefore watermark can still be removed effectively. The result also shows that by only delaying $a$, when $a$ increases, the removal result decreases. However, an easy enhancement of the removal algorithm is to delay those non-embedding by more than the estimated $a$. In the experiment, for $a = 600$ms, we have also tested delaying 1000ms. In this case, we can increase the removal rate to 98%.

**Algorithm 1** Realtime WM Attacking Algorithm

```plaintext
for all packet $p_i$ do

    if WM not yet detected then

        Continue SPRT algorithm with packet delay $d_i$

    end if

    if WM detected then

        Estimate $a$ with $\{d_1, \ldots, d_i\}$

        Use the estimated $a$ to decide whether $p_i$ is WM delayed

        if $p_i$ is NOT a WM delayed packet then

            remove WM by delaying $p_i$ by $a$

        end if

    end if

end for
```
Table 5.11: Realtime WM removal (Probabilistic)

<table>
<thead>
<tr>
<th>$L$</th>
<th>24 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>5</td>
</tr>
<tr>
<td>$h$ (hamming threshold)</td>
<td>5</td>
</tr>
<tr>
<td>Embedding pkts</td>
<td>1, 5, 9, 13, . . .</td>
</tr>
<tr>
<td>$a$</td>
<td>300ms, 400ms, 500ms, 600ms</td>
</tr>
<tr>
<td>Removal rate</td>
<td>98%, 87%, 72%, 56%</td>
</tr>
</tbody>
</table>

Table 5.12: Realtime WM removal (Quantization)

<table>
<thead>
<tr>
<th>$L$</th>
<th>24 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M$</td>
<td>5</td>
</tr>
<tr>
<td>$h$ (hamming threshold)</td>
<td>5</td>
</tr>
<tr>
<td>$S$</td>
<td>300ms, 400ms, 500ms, 600ms</td>
</tr>
<tr>
<td>Removal rate</td>
<td>98%, 99%, 100%, 100%</td>
</tr>
</tbody>
</table>

5.7.2 Realtime algorithm for The Quantization Based Watermark Scheme

Similarly, we can derive a realtime attacking scheme for the quantization based watermark scheme. Only two things need to be changed from Algorithm 1. First, we use the EM algorithm to estimate the value of $S$ instead of $a$, based on the distribution function of watermark delays. Second, the watermark is remove by further delaying those watermark embedding by $S$. In the experiment, we have tested 100 synthetic generated flows for watermark removal. From the result in Table 5.12, we can see that almost all the embedded watermarks can be removed.

5.8 Summary

In this chapter, we have investigated the secrecy of the probabilistic watermark scheme. Following the same ideas as those used to attack the quantization based scheme, we construct attacking techniques that can successfully detect watermark existence, recover the watermark delay value, and remove/duplicate the watermark. We have also proposed simple realtime attacking algorithms and demonstrated that all the constructed attacks can be done in realtime and can still significant impair the performance of the watermark schemes.
Chapter 6

The Secrecy and Enhancements of the Interval Based Watermark Scheme

The objective of this chapter is to investigate the secrecy issues of the newest watermark scheme developed: the *interval based watermark scheme* [41]. One major advantage of this scheme is that it can deal with repacketization, i.e., packets are merged together or split apart after they pass a stepping stone. Most of the existing passive and active timing based approaches cannot deal with this problem since they assume that the numbers of packets in different stepping stone flows do not change. In the interval based scheme, the entire flow is divided into different time intervals and embedded with watermark by changing the packet numbers in these intervals. It has been shown that this scheme can successfully resist both timing perturbation and repacketization.

This chapter begins with multiple attacking techniques proposed to compromise the interval based watermark scheme. We develop a suite of algorithms to detect the existence of watermark, infer watermark parameters, and remove/duplicate watermark. After the vulnerability of the interval based scheme is carefully analyzed, we then propose several enhancements over the original scheme. In the experiment, we show that the original scheme can be easily detected and the parameters can be recovered. With proposed improvements, the new scheme is much less detectable to attackers and its parameters are much more difficult to be inferred. Watermark removal and duplication also become less effective. The improved scheme has the same false positive rate and lower
true positive rate under timing perturbations. This is the trade-off for better secrecy.

The rest of this chapter is organized as follows. Section 6.1 describes the timing analysis attacks to the interval based scheme. Sections 6.2, 6.3 and 6.4 discuss how to detect watermark existence, recover parameters and defeat the interval based watermark scheme, respectively. Section 6.5 describes the security enhancements over the original scheme. Section 6.6 presents our experiments used to validate the proposed attacks and the security improvements. Section 6.7 summaries the results we have obtained.

6.1 Basis of Timing Analysis Attacks

Since the only change before and after a watermark is embedded is on packet timing, we construct attacks to the interval based scheme by analyzing the packet timestamps between adjacent stepping stone computers. We use the same method to obtain packet timestamps as described in Section 4.1. Basically, an attacker can modify his SSH client program to pad and extract the packet timestamps in the packets themselves. Before sending out a packet \( p_i \), the SSH client will read the current system time \( t^s_i \) and pad that time into the packet. The SSH client on the next stepping stone will extract the timestamp and compare it with the local system time \( t^r_i \). This process is illustrated in Figure 4.1. Then the packet delay is computed as \( d_i = t^r_i - t^s_i \). We also need to remove the clock skew between two computers by using the technique in [34]. Note the clock offset has no impact on the timing analysis and will be excluded from consideration.

Next, we construct statistic models for packet delays with/without watermark. The packet delays without watermark embedded can be approximated with a Gaussian distribution with mean \( \mu \) and variance \( \sigma^2 \):

\[
 f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},
\]

in which we use \( X \) to represent the packet delays without watermark embedding. Here we call such delays as normal network delays. Note there may exist certain network delays that cannot be well modeled by a Gaussian distribution. We will consider this case when we propose the watermark detection algorithm in Section 6.2.

For those packets that are delayed by the interval based watermark scheme, their packet delays should be the summation of normal packet delays and the extra watermark delay \( \delta \). We use
to represent the packet delays with watermark. The distribution is

\[ f_Y(x) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\delta-\mu)^2}{2\sigma^2}}. \] (6.2)

An example of the probability distributions of \( f_X \) and \( f_Y \) are shown in Figure 6.1.

![Figure 6.1: Distributions of \( f_X \) and \( f_Y \)](image)

When a network flow is embedded with watermark, some packet delays will follow the distribution of \( f_X \) and others will follow the distribution \( f_Y \). Therefore, when we consider the entire packet delays in a potentially watermarked network flow, we can model them by

\[ f(x) = (1 - \theta)f_X(x) + \theta f_Y(x). \] (6.3)

Here, \( \theta \) indicates the proportion of the packets in the flow that are delayed by the watermark scheme. Ideally, if no watermark is embedded, \( \theta = 0 \). However, since sometimes the normal network delays may not be well modeled by \( f_X \), we need to relax this restriction. A threshold value \( \theta_0 > 0 \) is chosen so that we consider a watermark is not embedded when \( \theta \leq \theta_0 \).

We do not consider \textit{collaterally delayed packets} (i.e., a packet delayed by the watermark scheme makes some packets after it to also be delayed in order to maintain the original packet order). Note the delay values of such collaterally delayed packets are between normal packet delays and watermark delays, and are only determined by the actual packet timing. It may be possible to identify these packets according to the specification of the watermark schemes. For example, the interval based watermark scheme always inserts 5ms delays before any collaterally delayed packets, thus gives attackers a good hint. We find out that the collaterally delayed packets only have limited impact on the timing analysis attacks and can be safely omitted from consideration.
Having created the statistic models for packet timing, we propose several attacking schemes to the interval based watermark scheme. More specifically, we focus on the following problems:

1. How can attackers detect that watermarks are embedded on the attacking flows? How soon can they decide that they are being traced?

2. Is it possible that important watermark parameters can be recovered by attackers?

3. How can attackers defeat the watermark scheme by reducing the true positive rate and/or increasing the false positive rate of the watermark scheme? That is, can watermarks be removed from the attacking flow and/or duplicated on other normal flows?

In practice, attackers must estimate the standard deviation $\sigma$ of the normal network delays before they can attack the watermark scheme. Note that the mean $\mu$ of normal packet delays is not important to attackers in any of their attacks. Therefore we can always assume $\mu = 0$.

### 6.2 Detecting Watermark Existence

From Equation 6.3, the packet delays in the flow embedded with watermark contain $\theta$ proportion of watermark delays. Therefore, the existence of watermark can be determined by estimating the value of $\theta$. We have already discussed that a small threshold value $\theta_0$ is necessary to compensate for the situation when Gaussian distribution cannot model all the normal packet delays well. This value can be determined empirically. For example, in Chapter 4, it is shown that usually only 2% of packet delays cannot be modeled by Gaussian distribution. So we could use $\theta_0 = 0.02$. Now we also need a threshold value $\theta_1$ to tell us when we can be sure that a watermark exists.

#### 6.2.1 The Selection of $\theta_1$

In Chapter 4, we have also faced how to choose the approximate values of $\theta_0$ and $\theta_1$ to detect the watermark for the quantization based scheme. Similarly, the threshold $\theta_0$ is to specify that there is no watermark when $\theta \leq \theta_0$. The value of $\theta_0$ can be decided by checking how well the normal packet delays fit the chosen statistical model. For example, when Gaussian distribution is used, about 2% of the packet delays cannot be well modeled. So we pick $\theta_0 = 0.02$. The second threshold $\theta_1$ indicates that watermark exists when $\theta \geq \theta_1$. Its value is based on certain assumptions.
about the watermark capability (i.e., the true positive rate and the false positive rate, given certain amount of timing perturbation) and the length of the attacking flow. From the watermark capability, we can compute the minimal number of packets to be delayed. Then $\theta_1$ is derived by dividing this minimal number by the total number of packets in the flow.

On the contrary, the interval based scheme does not explicitly choose the embedding packets. Instead it uses time interval and repetitively embeds watermark. For each watermark bit in each round of embedding, 3 intervals are used and the packets in 1 or 2 intervals will be delayed according to the bit embedded. The watermark embedding continues for the entire flow, except for the beginning part of the flow specified by offset value $O$. Since $O$ is usually much smaller than the length of the flow, the ratio between the embedding and non-embedding packets roughly varies between 1/3 (all bits are 1’s) and 2/3 (all bits are 0’s). So we could simply choose $\theta_1 = 0.33$ to cover all possible watermarks.

### 6.2.2 Detecting Watermark by Sequential Probability Ratio Test

Having the thresholds $\theta_0$ and $\theta_1$, we use the Sequential Probability Ratio Test (SPRT) [57] to detect the existence of the watermark scheme. This process is similar to the watermark detection approach used in [36]. Each time the attack receives a packet $p_j$, the packet delay $d_j$ is derived and the following is computed:

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

where $f$ is specified in Equation 6.3. Then $r_j$ is compared with two parameters $A$ and $B$ ($B < A$). If $r_j \leq B$, we terminate the testing and accept $\theta = \theta_0$ (i.e., not watermark). If $r_j \geq A$, we terminate and accept $\theta = \theta_1$ (i.e., watermark is detected). 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 $\alpha$ and false positive rate $\beta$ for the SPRT algorithm itself: $A = \log((1 - \beta)/\alpha)$ and $B = \log(\beta/(1 - \alpha))$. The above algorithm is further modified a little so that it will not terminate when $r_j \leq B$. Instead, it cleans all previous result and starts again for the next packet. In this way, the watermark can be detected even if it is only embedded on a part of the flow.

We have run simulations to validate the usefulness of the SPRT algorithm. For flows with normal network delays with $\sigma = 50$ms, we embed different watermarks with watermark delay $\delta$ changing from 0 to 500ms (0 stands for no watermark). Since the attacker does not know the value of watermark delay $\delta$, a randomly guessed value $\delta' = 200$ms is used. The thresholds are $\theta_0 = 0.02$ and $\theta_1 = 0.33$. The SPRT false negative rate and false positive rate are selected as $\alpha = \beta = 0.05$. 
We generate 1000 synthetic flows with 1000 packets in each flow using Tcplib, and then embed randomly generated 24-bit watermarks. The SPRT detection rate is shown in Figure 6.2. From the result, we can see that the SPRT algorithm have 100% detection rate when the actual watermark delay $\delta$ is greater than $\delta'$ guessed by the attacker. But the detection rate is not very good for small $\delta$. When no watermark is embedded, SPRT has 0 false positive rate in this case. Figure 6.3 shows the average number of packets processed by the SPRT algorithm when it detects the watermark or reaches the end of the flow. We can see for large $\delta$, the watermark can also be detected very efficiently.

Next, we find out the relation between SPRT result and the number of packets in the flow. We choose $\delta = 150$ms and $\delta' = 400$ms. The result is shown in Table 6.1. We can see that at first more packets will help watermark detection. But when the packet number is large enough, the detection rate becomes stable.

![Figure 6.2: SPRT detection rate (guessed watermark delay used in SPRT is 200ms)](image1)

![Figure 6.3: The number of packets processed by SPRT before SPRT terminates](image2)

<table>
<thead>
<tr>
<th>Pkt #</th>
<th>1000</th>
<th>2000</th>
<th>3000</th>
<th>4000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dt</td>
<td>24.4%</td>
<td>36.5%</td>
<td>59.1%</td>
<td>61.1%</td>
<td>60.7%</td>
</tr>
<tr>
<td>Pkt #</td>
<td>6000</td>
<td>7000</td>
<td>8000</td>
<td>9000</td>
<td>10000</td>
</tr>
<tr>
<td>Dt</td>
<td>57.8%</td>
<td>59.1%</td>
<td>58.3%</td>
<td>63.1%</td>
<td>61.7%</td>
</tr>
</tbody>
</table>

We have also tested the false positive rate of the SPRT algorithm. We first generate 100 flows whose packet delays are Gaussian distributed with $\sigma = 50$ms. Then we randomly increase 5% of the delays to 100ms to simulate the packet delays that cannot be modeled by Gaussian
distribution. The SPRT algorithm is then executed over the flows. Different guessed values of $\delta' = 100, 150, 200$ ms are used in SPRT. Other parameters are the same as shown in the previous paragraphs. The number of packets in the flows varies from 1000 to 10000. The false positive rates are shown in Figure 6.4. First we can see that when small value of $\delta'$ is used, it will greatly increase the false positive rate. Second, similar to the detection rate, the false positive rate first increases with the packet number and then becomes relatively stable. There is also a trade-off between watermark detection rate and false positive rate for attackers. They cannot simple choose very small $\delta'$ since it will cause intolerable high false positive rate.

![Figure 6.4: SPRT false positive rate](image)

6.3 Watermark Parameter Recovery

In this subsection, we investigate whether, how, and how well an attacker can recover the important watermark parameters. As discussed earlier, the attacker can observe the normal packet delays and obtain the distribution (i.e., $\mu$ and $\sigma$). 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 any of the watermark parameters described in Table 2.3.

In the following, we first estimate the watermark delay $\delta$ and the proportion parameter $\theta$. Then we identify which packets are embedded with watermark. Finally we discuss how the values of watermark interval $I$, offset value $O$, and actual watermark bits can be recovered.
6.3.1 Recover the Watermark Packet Delay $\delta$

We propose to use the Expectation-Maximization (EM) algorithm [8] to estimate the watermark delay $\delta$ and the proportion parameter $\theta$ from the 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 $\Psi = (\delta, \theta)^T$ be the vector of parameters to be estimated. Given observed packet delays $d = (d_1, \ldots, d_n)^T$, the likelihood function for $\Psi$ is $L(\Psi) = \prod_{j=1}^{n} f(d_j | \Psi)$. The EM algorithm estimates $\Psi$ by finding the value that can maximize the likelihood $L(\Psi)$. This can be done by solving the equation $\frac{\partial L(\Psi)}{\partial \Psi} = 0$, or equivalently, $\frac{\partial \log L(\Psi)}{\partial \Psi} = 0$, where $\log L(\Psi) = \sum_{j=1}^{n} \log \left( (1-\theta)f_X(d_j) + \theta f_Y(d_j) \right)$. In order to utilize the EM algorithm to estimate parameters in a mixture of two probability densities, we need additional parameters $z = (z_1, \ldots, z_n)^T$, where $z_j$ is 0 (or 1, resp.) indicating that $d_j$ is from the distribution $f_X$ (or $f_Y$, resp.). Note that the values in $z$ cannot be observed. They account for the “missing data”. By including $z$, the complete-data log likelihood for $\Psi$ is

$$
\log L_c(\Psi) = \sum_{j=1}^{n} \log \left( f_X(d_j | \delta) \right) + \sum_{j=1}^{n} z_j \log f_Y(d_j) + \sum_{j=1}^{n} \log \left( (1-\theta) \right) + \sum_{j=1}^{n} z_j \log \theta .
$$

The EM algorithm begins with an arbitrary initial value $\delta = \delta^{(0)}$ and $\theta = \theta^{(0)}$. In round $i + 1$, where $i = 0, 1, 2, \ldots$, the algorithm first performs the E-step to calculate the expectation of $z$ as

$$
z_j^{(i+1)} = \frac{\theta^{(i)} f_Y(d_j; \delta^{(i)})}{f(d_j; \delta^{(i)})} .
$$

Then the algorithm performs the M-step to maximize $Q(\Psi \mid \Psi^{(i)})$ with respect to $\Psi$. In our case, we simply compute

$$
\theta^{(i+1)} = \frac{1}{n} \sum_{j=1}^{n} z_j^{(i+1)}
$$

and

$$
\delta^{(i+1)} = \frac{\sum_{j=1}^{n} z_j^{(i+1)} d_j}{\sum_{j=1}^{n} z_j^{(i+1)}} .
$$

EM algorithm terminates when the difference between $L(\Psi^{(i+1)}) - L(\Psi^{(i)})$ is small enough.
Table 6.2 shows the estimation result using the EM algorithm, where $\delta'$ is the estimated value of $\delta$. The parameters used are the same as what we have used for the SPRT algorithm in the previous section, e.g., $\delta$ is from 100 to 500ms, the standard deviation of packet delays is $\sigma = 50$ms. The initial guessed values are $\delta^{(0)} = 100$ms and $\theta^{(0)} = 0.5$. From the result, we can see that except $\theta = 100$ms, the estimation produced by the EM algorithm is very accurate.

Table 6.2: The estimation of $\delta$ using EM algorithm

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>100</th>
<th>200</th>
<th>300</th>
<th>400</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta'$</td>
<td>61.2</td>
<td>196.7</td>
<td>297.4</td>
<td>396.9</td>
<td>490.1</td>
</tr>
<tr>
<td>error</td>
<td>38.8%</td>
<td>1.65%</td>
<td>0.87%</td>
<td>0.77%</td>
<td>1.98%</td>
</tr>
</tbody>
</table>

6.3.2 Identifying Packets Delayed by Watermark

After the values of $\delta$ and $\theta$ have been estimated, the probability densities $f_Y$ and $f$ are determined. Now we decide for each packet whether it has been delayed by the watermark encoder or not. Here we use the Bayes decision rule [12] to minimize the cost of wrong decisions. Suppose $\lambda(i, j)$ is the loss function when we decide state $i$ while 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). By using the Bayes decision rule, the expected loss can then be minimized upon our probability density functions and certain loss function $\lambda$. More specifically, we decide a packet $p_i$ to be a watermark delayed packet if its packet delay $d_i$ satisfies

$$\frac{f_Y(d_i)}{f_X(d_i)} \geq \frac{\lambda(2, 1) \cdot (1 - \theta)}{\lambda(1, 2) \cdot \theta}. \quad (6.9)$$

For our distribution functions $f_X$ and $f_Y$, we can numerically compute a threshold delay value $\overline{d}$ from Inequality 6.9. Then we decide that packet $p_i$ is a watermark delayed packet when its corresponding delay $d_i \geq \overline{d}$.

6.3.3 Recovering the Interval and Offset

Now we examine how the watermark interval $I$ and offset value $O$ can be recovered. At this stage, the attacker has the packet timestamps and knows which packets are delayed by the watermark scheme. We call these packets watermark embedding packets, or simply embedding packets. Those without watermark delays are simply called non-embedding packets.
Brute-force algorithm

The intuitive method to estimate $I$ and $O$ is to enumerate every possible values. For each tentative value pair of $I$ and $O$, we divide the entire flow into different intervals and check if there exist any intervals that contain both embedding and non-embedding packets. For the real values of $I$ and $O$, none of the intervals should include both kinds of packets. Therefore the number of the different kinds of packets that fall into same intervals can be used as an indicator for the correct values of $I$ and $O$. We call it conflict packet number and denote as $N_{conf}$.

We test the value of $N_{conf}$ with different values of $I$ and $O$ when the real values are $I = 1000$ms and $O = 2000$ms. Figure 6.5 shows the result when correct value of $I$ is used. The value of $N_{conf}$ drops to near 0 for $O' = O + mI$, where $m$ is an integer. It is because when $O$ changes by the amount of $I$, all packets will once again have a very good alignment. Figure 6.6 shows the result for different $I$ when the correct value of $O$ is used. Generally, the value of $N_{conf}$ decreases when $I$ decreases. But it will suddenly drop to near 0 for the correct $I$ and $I' = I/m$, where $m$ is a positive integer.

From these results, we can see that the number $N_{conf}$ can indeed provide a very good estimation about both $I$ and $O$. Although the minimum may appear for multiple combinations of $I$ and $O$, wrong values can be easily identified by using additional heuristics. For example, the interval should be the largest one that can achieve the minimal $N_{conf}$. However, we also find out that the minimal value of $N_{conf}$ is very sensitive to the value of $I$. Only a very little deviation (e.g., 1ms) from the correct $I$ will cause $N_{conf}$ to increase dramatically. Therefore we need to check a lot of possible values of $I$ to make sure we can hit the correct value. Combining the possible values of $O$, generally the computation overhead will be quite large (note there are heuristics that can be used to improve the efficiency as shown later).

In summary, using $N_{conf}$ can provide very accurate estimation about both $I$ and $O$. Though finding the correct values can be quite time consuming.

Heuristic algorithm with variable-length intervals

To improve the computation efficiency, we propose another method that only does an approximate estimation about the intervals. Note the objective of attackers to recover the watermark parameters is to effectively defeat the watermark scheme. The recovery of $I$ and $O$ will give attacks the opportunity to duplicate the same watermark on other normal flows, thus increase the false positive rate. Therefore in the new method, we focus more on assisting watermark duplicate rather than recovering the exact values. We use heuristics to find good approximation of the interval $I$ and
divide the entire flow into intervals with various lengths.

First, we compute a good estimation about the real $I$. Having the flow whose packets are labeled as either “embedding” or “non-embedding”, we divide the flow into different areas that contain only embedding or non-embedding packets. However, the boundary of two adjacent areas overlaps with each other. For an embedding area, its boundaries are defined by the last non-embedding packet before it and the first non-embedding packet after it. It is similar for non-embedding areas. This process is illustrated in Figure 6.7, in which the black bars represent embedding packets. Here we only label the embedding areas.

Figure 6.7: Divide the flow into embedding/non-embedding areas

Clearly one embedding area may contain multiple embedding and non-embedding intervals, since there may not exist any packet in those non-embedding intervals. But for the smallest embedding areas, usually they give us a good (but larger) approximation about the real interval. Therefore we could choose the shortest length of embedding areas as the estimation $\tilde{I}$. For our estimation to be more robust against errors from falsely identifying non-embedding packets as embedding packets or vice versa, we use the median of the smallest 5% of all embedding areas. In our simulation using synthetic flows generated by Tcplib, on average the estimated $\tilde{I}$ is only about 6%
larger than the real $I$. Note $\tilde{I}$ can also be used to accelerate the brute-force algorithm.

Now we begin to divide the flow into intervals using $\tilde{I}$. We first find the smallest timestamp difference between two adjacent embedding/non-embedding packets. Intuitively the middle point between such embedding/non-embedding packets tells us where the interval boundary exists. Therefore we start from this middle point, and divide intervals in both backward and forward directions. Each time, we form an interval of $\tilde{I}$ and check whether it contains both embedding and non-embedding packets. If it does, we shrink this interval a little bit so that it just contains only one kind of packets. We keep doing until we reach the beginning or the end of the flow. Therefore the entire flow will be divided into intervals with different lengths. We also label each interval as “embedding”, “non-embedding”, or “unknown” based on whether it contains embedding, non-embedding or no packets, respectively. Although we do not know the exact value of $I$ and $O$, we will show later that such intervals can significantly increase watermark duplication on normal flows.

### 6.3.4 Recovering Watermark Bits

Now we consider how the embedded watermark bits can be recovered using the information about interval and offset. We assume the values of $I$ and $O$ are recovered by using the brute-force algorithm. The basic idea is that since the watermark is embedded over and over again with the same watermark bit order, we can guess a watermark length and divide the entire flow into multiple pieces, where each piece corresponds to one round of watermark embedding. If we guess correctly, by comparing these pieces, we should find the embedding and non-embedding intervals in same positions match with each other. So we use the number of matched time intervals to infer the watermark length $L$.

With the correct $I$ and $O$, we divide the entire flow into different intervals, and label each interval as “embedding” “non-embedding” or “unknown” based on whether it contains embedding packets, non-embedding packets or no packet. For simplicity, we use single-character labels: ‘e’, ‘n’ and ‘u’. The entire labels are then represented by a string. Our goal is to split this string into multiple equal-length substrings so that all the substrings match with each other on characters ‘e’ and ‘n’. The character ‘u’ can match either ‘e’ and ‘n’. We want to find the partition with the smallest number of unmatched ‘e’ and ‘n’. For example, suppose the string is “enuuuneueuuunnuueneueuuuenuuunun” for a watermark ‘0101’. It can be divide into 3 substrings with perfect match:

“enuuuneuuuu”
From these 3 substrings, we are able to remove all “unknown” characters and obtain “enenenenenen”, which is exactly the embedded watermark. Note the original string may also be divided into 6 substrings which still have a perfect match, since the watermark repeats pattern ‘01’ twice. Therefore as long as the watermark contains repeatable patterns, we will find multiple good matches and cannot decide the actual watermark used. However, this information can still make attackers be able to duplication the entire watermark.

For arbitrary flows and watermarks, we may not be able to determine the values of all $u$’s through substring partition. However, we also know that there do not exist 3 or more consecutive intervals of all $e$ or $n$, such as ’eee’ or ’nnnn’. So more “unknown” intervals can be determined, and watermark duplication will have higher chance to be successful.

We use a simple string partition algorithm, which checks all substring lengths from small to large, computes and finds the minimal number of unmatched ‘$e$’ and ‘$n$’ intervals. Note the selection of watermark length is relatively limited (e.g., at least 20 bits may be necessary for reasonable true positive and false positive rates, and 50 bits may be considered long enough for watermark encoders). Therefore this simple algorithm has reasonable overhead and its computation complexity is only $O(N)$, where $N$ is the number of packets in the flow.

We validate this algorithm with simulations. We use 24-bit randomly generated watermarks. The watermark encoding is repeated from 5 to 20 times. For each repetition number, 1000 watermarks are tested. We first label all intervals with ‘$e$’ and ‘$n$’. Then the labels of 50% of the randomly selected intervals are changed to ‘$u$’. Next, we perform the string partition algorithm with watermark length from 20 to 40 bits. We find out that all the watermarks can be correctly identified as 24-bit long. The percentage of watermark bits that can be correctly determined is shown in Table 6.3. It only requires repetitively embedding the watermark about 15 times before all the bits can be correctly identified.

<table>
<thead>
<tr>
<th>repetition</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM bits correctly recovered</td>
<td>74.4%</td>
<td>97.9%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

The situation becomes more complicated when the real interval and offset values are not
known. In this case, we could use the heuristic algorithm to generate a sequence of variable-length intervals $I_1, I_2, \ldots$, and label the intervals with ‘e’, ‘u’ or ‘n’. Note we can further infer some of the ‘n’ intervals by using the fact that there is no 3 or more consecutive ‘e’ and ‘u’ intervals. It is tempting to use the above string partition algorithm for the variable length intervals. However, since the heuristic algorithm is developed to provide an approximation of each interval for watermark duplication, it is very possible that the number of estimated intervals is different from the real intervals. Therefore we usually cannot get a good match and determine the watermark bits.

### 6.4 Watermark Removal and Duplication

The objective for attackers to recover the important watermark parameters is to efficiently and effectively defeat the watermark scheme. In this subsection, we discuss how attackers can achieve this goal by removing the watermarks from the attacking flows to reduce the true positive rate, and/or duplicating the embedded watermarks to other normal flows to fool the watermark decoders and increase the false positive rate.

#### 6.4.1 Watermark Removal

To remove the watermark from the attacking flow, an attacker can further delay the non-embedding packets that are identified previously. By doing so, the attacker tries to negate the effect of watermark encoding, and partially restore the original flow without watermark embedding. Therefore the success of watermark removal only depends on how well the attacker can detect the watermark and distinguish between embedding and non-embedding packets.

#### 6.4.2 Watermark Duplication

Now we discuss how to duplicate the embedding watermarks on other normal flows. Duplication is straightforward if $I$ and $O$ are recovered. So here we only consider duplication by using the interval information obtained from the heuristic algorithm in Section 6.3.3. Basically, we have a sequence of intervals $I_1, I_2, \ldots$ with different lengths, and a sequence of labels $l_1, l_2, \ldots$ which could be “embedding” “non-embedding” and “unknown”. For each packet in a normal flow, we first determine which interval it falls into. If it falls into a “embedding” interval, we then delay this packet to duplicate the watermark.
There are two factors that limit the successful duplication rate. First, for the stepping stone connection embedded with watermark, it is possible that many intervals contain no packets. Therefore a large proportion of intervals may have the label of “unknown”, for which we cannot decide whether we should delay or not. Moreover, for the corresponding embedding intervals in a normal flow, it is also likely that some of them do not have any packets. Therefore the attacker may only have very limited number of packets to duplicate the watermark. For example, Figure 6.8 shows a part of an attacking flow and a normal flow on which the attacker tries to duplicate the watermark. We can see that for those packets in the normal flow, none of them can be delayed since they all fall into “unknown” intervals.

![Figure 6.8: Difficulty in watermark duplication](image)

The good news for the attackers is that they can delay packets much more than a watermark encoder can, since the encoder must make sure that the watermark delays are not too noticeable to the attackers. Therefore the attackers can always choose large delay values to compensate a little for the two undesirable factors described in the previous paragraph. Figure 6.9 shows the watermark duplication rate for different delay values used by attackers. Unsurprisingly, the duplication rates increase with both the number of packets in the flows and the delay values.

![Figure 6.9: Watermark duplication result](image)
6.5 Security Improvements over the Original Scheme

In the previous section, the attackers’ threats on the interval based scheme are carefully investigated. We have shown that intelligent attackers can significantly impair the performance of the watermark scheme. Therefore it is very important for researchers to include these attack scenarios into consideration when they design any active correlation schemes. In this section, we propose multiple security improvements in order to deter the attacks used in the previous section.

In the following, we first consider the improvements that make the watermark less detectable to attackers but can still achieve similar performance. Second, we investigate different enhancements to obscure the important watermark parameters.

6.5.1 Enhancement 1: Security Improvements for Watermark Detection

From previous analysis, watermark detection by attackers depends on the total amount of delays introduced by watermark encoders. For a watermark scheme to be less detectable, we need to reduce the the number of delayed packets and the delay value of each packet.

We find out that the original interval based watermark scheme can be further improved to reduce the amount of delays. In the original scheme, all packets fall into a watermark embedding interval are simply delayed by $\delta$ to move certain packets to the next interval. However, if a packet is too far away from the next interval (i.e., the distance is greater than $\delta$), delaying that packet by $\delta$ has no gain\(^1\). It only makes the watermark more noticeable. So the first improvement is to not delay such a packet at all. The watermark encoder will first compute the distance between any packet in an embedding interval and the next interval. A packet is only delayed when the distance is smaller than $\delta$.

Since the watermark scheme only wants to move some packets to their next intervals, always delaying by $\delta$ is not necessary even for those packets that are close enough to the next interval. Instead, we could only delay them by the smallest values that can make them fall into the next interval. As shown in Figure 6.10, the first packet in the embedding interval is not delayed since it is too far away from the next interval. The second and third packets are delayed by two different amount $\delta_1$ and $\delta_2$ ($\delta_1, \delta_2 < \delta$) so that they just make to the next interval (the packet order is also maintained).

However, there is a problem in the above improvement. Since packets are delayed so that

\(^1\)Such delays may help watermark detection under timing perturbation.
they just make to the next interval, attackers can have a very good estimation about where the interval boundaries are. The interval information is very useful for the attackers to duplicate watermark to normal flows. Therefore we further obscure the boundary information by using randomization.

Suppose packet $p_i$ is the first packet in an embedding interval that can be delayed to the next interval. Suppose the distance between $p_i$ and the next interval is $\delta_i (\delta_i < \delta)$. We first decide the seed $s_\delta$ for a pseudo-random function. Each time we generate a random value between $\delta_i$ and $\delta$ and use that value to delay $p_i$. Then it is much harder for attackers to know the interval boundaries. The randomly generated delay value is only used for the first packet to be delayed in an interval. For other packets after it, they are simply pushed back to maintain the correct packet order, just as what we do for non-embedding packets. So the attackers cannot distinguish whether a packet pushed back by an embedding packet is still an embedding packet or a non-embedding packet. This process is illustrated in Figure 6.11.

We have tested the performance of the improved scheme under the SPRT algorithm. The watermark is embedded with $\delta = 150$ms and the attacker uses $\delta' = 400$ms. The SPRT detection rate changing with the number of packets in the flow is shown in Table 6.4. For comparison, we also list the detection rate for the original scheme as shown in Table 6.1. The new scheme performs much better than the original one in terms of secrecy.
Table 6.4: SPRT detection rate

<table>
<thead>
<tr>
<th>Packet #</th>
<th>Original(%)</th>
<th>Improved(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>24.4</td>
<td>3.4</td>
</tr>
<tr>
<td>2000</td>
<td>36.5</td>
<td>4.1</td>
</tr>
<tr>
<td>3000</td>
<td>59.1</td>
<td>3.1</td>
</tr>
<tr>
<td>4000</td>
<td>61.1</td>
<td>3.3</td>
</tr>
<tr>
<td>5000</td>
<td>60.7</td>
<td>3.9</td>
</tr>
</tbody>
</table>

6.5.2 Security Improvements to Resist Watermark Recovery

As we discuss in the previous section, it is possible for attackers to recover the watermark offset $O$ and interval $I$ using a brute-force algorithm despite the high computation complexity, or to create variable-length intervals to partition the flow. The actual watermark bits can also be recovered knowing $I$ and $O$. In this subsection, we propose several enhancements to defeat these possible attacks.

Enhancement 2: Pseudo-Randomized Interval Lengths

The fact that the original scheme uses fixed size intervals makes the recovery of interval value possible. To fix this problem, we propose to use intervals with different lengths by using randomization. We first determine a minimal and a maximal interval lengths $I_{\text{min}}$ and $I_{\text{max}}$, and choose the seed $s_I$ for a pseudo-random function. Each time, we generate a random interval length between $[I_{\text{min}}, I_{\text{max}}]$ and use that for watermark encoding. In watermark decoding, the decoder use the same seed to reproduce the interval lengths and decode the watermark. To compensate for different interval lengths, the packet numbers in the intervals need to be normalized on their interval lengths.

Enhancement 3: Pseudo-Randomized Watermark Encoding

We have also shown earlier that watermark bits used in the original scheme can be recovered by attackers. This is because in each round of watermark embedding, the encoder always uses the same order of watermark bits. Then attackers can compare the embedding/non-embedding intervals between different rounds to figure out what is embedded exactly. To defeat this attack, we propose to use randomized bit order between different round. The encoder first choose a seed $s_e$ for a pseudo-random function. Each round, the encoder generates a random permutation of the watermark bits using the Knuth Shuffle [21] algorithm. Then the bit orders between different rounds
can be totally different. The attackers cannot decide what is embedded anymore by still using the attack techniques proposed in Section 6.3.4.

6.5.3 The Improved Encoding and Decoding Algorithm

In summary, three optional security improvements are proposed. The encoder can use randomly generated delay value to reduce watermark detectability. Random-length intervals and random watermark encoding orders can also be used to increase the difficulty of watermark recovery and watermark duplication. Each improvement can be applied independently from others. Table 6.5 lists the parameters used in the improved watermark scheme.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>The number of binary watermark bits</td>
</tr>
<tr>
<td>$w_1 \ldots w_L$</td>
<td>The watermark, in which each bit $w_i$ is either 0 or 1</td>
</tr>
<tr>
<td>$O$</td>
<td>Offset value</td>
</tr>
<tr>
<td>$I_{\text{min}}$</td>
<td>The minimal interval length</td>
</tr>
<tr>
<td>$I_{\text{max}}$</td>
<td>The maximal interval length</td>
</tr>
<tr>
<td>$s_I$</td>
<td>The seed to generate random interval length</td>
</tr>
<tr>
<td>$\delta$</td>
<td>Extra watermark delay (maximum delay)</td>
</tr>
<tr>
<td>$s_\delta$</td>
<td>The seed to generate random delay value (only for encoding)</td>
</tr>
<tr>
<td>$s_e$</td>
<td>The seed to generate random bit order</td>
</tr>
</tbody>
</table>

6.6 Experiment

In this section, we experimentally evaluate the effectiveness of our attacking technique to both the original and the improved interval based watermark schemes. Then we compare the performance of both schemes in terms of true positive and false positive rates under random timing perturbation.

6.6.1 Attacking the Interval Based Watermark Schemes

We first evaluate the impact of our attacks to both watermark schemes. We use network flows collected from hosts in PlanetLab. Totally 41 traces of 1000 packets 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 6.1 and estimate the variance $\sigma^2$. All other traces from the same
host will directly use $\sigma$ without estimating for their own.

The flows are embedded with 24-bits watermarks. Different delay values $\delta = 100, 200, 300, 400, 500$ms have been used in the watermark schemes. For the original scheme, we choose
$O = 2500$ms and $I = 1000$ms. For the improved scheme, we may use random intervals within
$[I_{\min} = 700$ms, $I_{\max} = 1300$ms] and $[I_{\min} = 500$ms, $I_{\max} = 1500$ms]. For the improved scheme,
we compare the results when enhancements 1, 2, and 3 are applied independently and all together.

Watermark Detection

Here we test the effectiveness of watermark detection using the SPRT algorithm, with
$\theta_0 = 0.02$ and $\theta_1 = 0.33$. The false positive and false negative rate of the SPRT algorithm are
chosen as $\alpha = \beta = 0.05$. Since attackers do not know the real delay value $\delta$, a guessed value
$\delta' = 400$ms is used in SPRT.

Figure 6.12 shows the watermark detection rates of the SPRT algorithm changing with
different values of watermark delay $\delta$. When the real $\delta$ is much smaller than guessed $\delta'$, SPRT will
miss almost all the watermarks. Enhancements 2 and 3 do not significantly decrease the watermark
detection rate. Enhancement 1 alone causes a much lower detection rate, especially when the real
$\delta$ is close to the guessed $\delta'$ by attackers. Please note that when a small $\delta$ is chosen by the encoder,
more packets are needed for the watermark schemes to achieve the same true positive rate than
when a larger $\delta$ is used. In other words, the true positive rate decreases when a smaller delay is used
for any flow. Figure 6.13 shows the number of packets processed by SPRT before it terminates.
The SPRT algorithm only terminates when it detects a watermark or it reaches the end of the flow.
When watermarks are detected in both schemes, the improved scheme with enhancement 1 forces
attackers to check more packets, thus can reduce the effect of any countermeasures may be used.

Watermark Recovery

After the watermarks are detected, the attacker will first estimate the delay value $\delta$ using
the EM algorithm. They can only use the packets received so far. In the EM algorithm, we set the
initial value $\delta^{(0)} = 100$ms and $\theta^{(0)} = 0.5$. The estimation error for $\delta$ is shown in Figure 6.14. We
can see that for the original scheme, the estimation result is very accurate. Enhancement 2 and 3
do not affect the EM result very much. However, using enhancement 1, the improved scheme has a much larger estimation error than that of the original scheme. Therefore the reduced and randomized delays can increase the resistance of the watermark parameters against parameter recovery.

Next, we show how the brute-force algorithm in Section 6.3.3 works for real data to recover the correct values of interval $I$ and offset $O$. Since the brute-force algorithm requires to try every possible values of $I$ and $O$, here we only shown the number of different kinds of packets $N_{conf}$ that fall into the same intervals for the actual $I$ and $O$. Figures 6.15 and 6.16 show the results, which are consistent with the results in Section 6.3.3. We can see that despite the high computation overhead, $N_{conf}$ is a good indicator to recover the correct value of $O$ and $I$. Therefore using fixed-length intervals leaks very important information to attackers and is definitely not desired for
When the correct values of $O$ and $I$ are known, we further test how many of the watermark bits can we correctly recover. For the flows with 1000 packets and 24-bit watermarks, our algorithm can correctly identify the length of all the watermarks. On average there are about 21.8% of the watermark bits cannot be recovered, since we cannot identify all the “unknown” intervals by only using 1000 packets in the flows.

In comparison, when enhancement 3 are used in the improved watermark scheme, we can recover neither the watermark length nor the actual watermark bits.

### Watermark Removal and Duplication

Now we test how the watermark removal and duplication algorithms perform on both schemes. To remove the watermark, the results of the watermark detection and the identification of the embedding packets are used. After a watermark is detected and the watermark delay $\delta$ is estimated by the EM algorithm, each packet in the rest of the flow will be labeled as either non-embedding or embedding packets, based on the algorithm in Section 6.3.2. Then watermark removal is performed by further delaying the identified non-embedding packets by the amount of estimated $\delta$. The flows manipulated by the attackers are then sent to the watermark decoders and the true positive rate of the watermark schemes are calculated. Suppose the true positive rates before and after watermark removal are $T_1$ and $T_2$, the watermark removal rate is simply $\frac{T_1 - T_2}{T_1}$. The removal result is shown in Figure 6.17. Apparently, the watermarks embedded by the original scheme can be easily removed for large $\delta$. Enhancements 2 and 3 still cannot improve the performance too much.
On the other hand, enhancement 1 can significantly reduce the watermark removal rate. When $\delta$ is very small, the watermarks cannot be removed since they cannot be detected by the attackers.

![Figure 6.17: Watermark removal for the original and the improved schemes](image)

![Figure 6.18: WM duplication using the heuristic algorithm on both schemes](image)

To duplicate the watermark, one way is to recover the parameters $I$ and $O$ by using the brute-force algorithm and then recover the watermark bits. If attackers do not care the overhead of the brute-force algorithm, it is easy to see from the results in Section 6.6.1 that the watermarks will be recovered and then successfully duplicated to other flows. Therefore here we only focus on the performance of the heuristic algorithm in Section 6.3.3. The heuristic algorithm will divide the attacking flow into variable length intervals and label each interval as “embedding”, “non-embedding” or “unknown” based on what kinds of packets it has. To duplicate the watermark, a normal flow will first be divided into multiple intervals using the above interval information. For each interval, if it is labeled as “embedding”, all the packets will then be delayed by 500ms. The result is shown in Figure 6.18. As we can see, the duplication for small delays is close to 0, partly because of the difficulties in watermark detection or parameter estimation. The duplication rates for the improved scheme and enhancement 3 are the highest. Enhancement 2 can significantly reduce the duplication rate. The results when enhancement 1 is used are the lowest.

### 6.6.2 Computation Overhead of Timing Analysis Attacks

Now we discuss the computation overhead of the timing analysis attacks. To detect the watermark existence with SPRT algorithm, the most time consuming part for each packet $p_j$ is to compute the value of $f(d_j, \theta_1) / f(d_j, \theta_0)$, which can still be calculate very quickly. If we considering the entire flow with $n$ packets, this algorithm is $O(n)$. 
In the EM algorithm, suppose we estimate $\delta$ from totally $n$ packet delays. In each round of E-step, we need to compute $z = (z_1, \ldots, z_n)^T$ from Equation 6.6. In each round of M-step, we update the value of $\delta$ from $z$. In our experiment, the EM algorithm terminates after around 10 to 20 rounds. Therefore the EM algorithm has a computation complexity of $O(n)$. The time required by the EM algorithm to estimate $\delta$ for 1000 packet delays is roughly 60ms without any optimization, by using a Pentium-4 2.40GHz desktop with 512MB memory. Therefore EM algorithm can be implemented as real-time algorithm.

The brute-force algorithm to recover $I$ and $O$ usually requires to check thousands of possible values of $I$ and $O$ each. Totally we need to compute $N_{\text{conf}}$ millions of times. The computation of $N_{\text{conf}}$ is roughly $O(n)$, where $n$ is the number of packets in the flow. However, since $I$ and $O$ are usually independent with $n$, the brute-force algorithm can still be $O(n)$.

It is easy to see that watermark removal is an $O(n)$ algorithm, since it only further delays embedding packets. Both the heuristic algorithm in Section 6.3.3 and the watermark duplication algorithm require to process the original flow or the flow to duplicate the watermark once. Therefore they also have $O(n)$ computation complexity.

### 6.6.3 The Performance of Improved Interval Based Watermark Scheme

In this subsection we test the performance of the improved watermark scheme in terms of true positive and false positive rates.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>24</td>
</tr>
<tr>
<td>$h$</td>
<td>5</td>
</tr>
<tr>
<td>$w_1 \ldots w_L$</td>
<td>randomly generated</td>
</tr>
<tr>
<td>$I_{\min}$</td>
<td>700ms and 500ms</td>
</tr>
<tr>
<td>$I_{\max}$</td>
<td>1300ms and 1500ms</td>
</tr>
<tr>
<td>$\delta$</td>
<td>400ms</td>
</tr>
<tr>
<td>$P_{\max}$</td>
<td>0, 200, 400, 600ms</td>
</tr>
</tbody>
</table>

We first compare the true positive rates of the original and improved watermark schemes under certain amount of timing perturbation. The parameters used are in Table 6.6, in which the perturbations are uniformly distributed between 0 to the maximum perturbation $P_{\max}$.

Figure 6.19 shows the comparison between true positive rates of the original scheme and
the improved scheme only using enhancement 2 when we choose $I_{\text{min}} = 700\text{ms}$ and $I_{\text{max}} = 1300\text{ms}$. The results of the original scheme are labeled with solid lines while those of the improved scheme are labeled with dotted lines. Figure 6.20 shows the comparison when we choose $I_{\text{min}} = 500\text{ms}$ and $I_{\text{max}} = 1500\text{ms}$. We can see that by using randomized intervals, the true positive rate is slightly decreased under timing perturbation. The result for $I \in [500, 1500]$ is also slightly lower than that for $I \in [700, 1300]$.

Figure 6.19: True positive rate with enhancement 2 ($I_{\text{min}} = 700\text{ms}, I_{\text{max}} = 1300\text{ms}$)

Figure 6.20: True positive rate with enhancement 2 ($I_{\text{min}} = 500\text{ms}, I_{\text{max}} = 1500\text{ms}$)

Figure 6.21 compares the true positive rates when only enhancement 1 are used in the improved scheme. In this case, the true positive rate drops for the improved scheme. This is because we only delayed a portion of the packets compared with the original scheme. The total amount of the delays is much smaller so that it is much more difficult to detect the watermark. This is the trade-off for the improved scheme.

Figure 6.21: True positive rate with enhancement 1

Figure 6.22: True positive rate with enhancement 3
Figure 6.22 shows the true positive rates when enhancement 3 is used. The results of the improved and the original schemes are very similar.

Figure 6.23 shows the true positive rates when all 3 enhancements are used. The improved scheme has lower true positive rate than the original one. However, we have significantly improved its secrecy and security.

We have also performed online experiments to test the true positive rates when both timing perturbation and repacketization exist. The system setting is the same as that used in [41]. No timing perturbation and a maximum of 400ms perturbation are tested. Figure 6.24 shows the result. We can see that the true positive rates are consistent with those in the previous experiments.

Figures 6.25 and 6.26 show the comparisons of false positive rates of the original scheme and the improved scheme with all 3 enhancements, under 0 or 1000ms timing perturbations. We can see that the improved scheme has very similar false positive rates as those of the original scheme.

In summary, compared with the original watermark scheme, the proposed security improvements decrease the true positive rate under timing perturbations. The major reason is enhancement 1 (i.e., the reduced and randomized delays), although it does not decrease the performance when no timing perturbation is inserted. Enhancement 2 also slightly decreases the performance. It is also easy to see that enhancement 3 should not affect the performance. The decrease of true positive rate is the trade-off we have to pay in order to achieve better security of the watermark scheme. Also note that the three improvements can be applied independently. Therefore a watermark encoder can choose the most desired improvements to achieve the desired trade-off between secrecy and performance. The false positive rate of the improved scheme is the same as that of the
6.6.4 Parameter Selection for the Improved Watermark Scheme

As shown in the previous results, there exists a trade-off between the watermark secrecy and the watermark capability. Small watermark delay $\delta$ makes detecting the existence of watermarks more difficult. The watermark removal and duplication will also be less successful. The disadvantage of using smaller $\delta$ is that it also reduces true positive rate for the watermark scheme. More number of packets has to be embedded with watermark in order to achieve the same true positive rate. We prefer to use small $\delta$ if watermark secrecy is the major concern and there are enough packets. As shown in Section 6.2, the SPRT detection rate will reach its maximum for a certain number of packets. Therefore embedding watermark on a lot of packets using small $\delta$ will not increase attackers’ ability to detect the watermark. Although attackers may use smaller guessed delay value to increase the watermark detection rate, the false positive rate will also be much higher and cause them a lot of trouble.

6.7 Summary

In this chapter, we have investigated the secrecy of the recently proposed interval based active watermark scheme. Based on the timing analysis between stepping stones, several attacks are constructed to detect the watermark existence using the SPRT algorithm, recover important watermark parameters with the EM algorithm and the string partition algorithm, and remove/duplicate
the watermark. To increase the resistance to the proposed attacks, several security improvements are developed. It has been shown in experiment that the improved scheme can significantly decrease the successfulness of the attacks, which are very effective on the original scheme.

The disadvantage of the security improvements is that they reduce the performance of the watermark scheme in terms of true positive rate. In the future, we want to investigate the interrelation between secrecy and performance, and develop new schemes with both high secrecy and true positive rate.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, I have investigated multiple research problems related to attack attribution through stepping stone computers using active timing based watermarking. Stepping stone computers are commonly used by attackers to evade from detection. Since only the last host in a stepping stone connection chain will actually attack the victim, it is very difficult to trace back to the origin of the attack when we can only identify the last host using IP source address and IP trace-back techniques.

A lot of research has been done to address this attack attribution problem. The host based approaches may fail to work if the stepping stones are totally controlled by attackers. On the other hand, the network based approaches focus on correlating the connections in the same stepping stone connection chain together. Among all the network based schemes, the active timing based watermarking is one of the most promising schemes that can resist different interference and generally requires less packets than passive timing based schemes.

The interference that affects the correlation of stepping stone connections comes from different places. First, it comes from normal operations of network and stepping stone computers. For example, attackers can use secure transmission protocols such as SSH or IPsec. The packets passing stepping stones may be repacketized to improve network performance. Packets may be lost and retransmitted between stepping stones. Second, the interference also comes directly from
attackers, who may intentionally introduce timing perturbation, meaningless chaff packets or flow split/merge to defeat timing based correlation schemes. Finally, active timing based schemes are also facing direct threats of attackers. The changes on flow timing may leak important information and cause the schemes to be effectively defeated.

In this dissertation, I first propose a trace-back scheme that can correlate stepping stone flows under both chaff packets and timing perturbations. Based on active watermark schemes, this scheme identifies possible corresponding packets between an upstream flow and chaffed downstream flow. Multiple algorithms are developed to efficiently decode watermarks from the chaffed flows.

The emphasis of this dissertation is on the secrecy of the active watermark schemes against timing analysis attacks. Unlike passive timing based correlation schemes, the active watermark schemes need to modify packet timing and are potentially detectable to attackers. We are the first to systematically analyze the attackers’ threats to these schemes. Multiple sophisticated attacking techniques are proposed that can detect watermark existence, recover certain watermark parameters, and defeat the watermark scheme by removing and/or duplicating the watermark. To improve the resistance to such attacks, several security enhancements for the interval based watermark scheme are also developed. It has been shown that such enhancements make the improved scheme much less vulnerable to the constructed attacks, at the cost of slightly decreased performance.

In conclusion, the active timing based watermark schemes are promising trace-back techniques through stepping stones. These schemes are capable of dealing with different interferences such as encrypted transmission, timing perturbation, chaff packets, flow split/merge and repackettization. It also has certain advantages over the passive timing based correlation schemes. However, we must keep in mind that such schemes may be vulnerable to attacks if not used correctly.

7.2 Future Work

The first thing on which we will perform further analysis is the timing characteristics of packet delays with and without watermark. All our results are based on correct modeling of packet delays. In our previous experiments, the packet delays were collected in the application layer, not in the network layer. It is because attackers can easily modify their client program and pad/compare the packet timestamps. However, accessing network interfaces requires much more work for attackers. Also, the CPU and network interface of the computers were usually not in
heavy load in our experiments. So the possible extensive usage of stepping stones in real situation may affect the collected packet delays. For example, in this case the packet delays may have a much larger variance. The attackers may not be able to achieve the same detection rate and false positive rate in watermark detection, while the performance of the watermark scheme may not be significantly affected. We need to demonstrate that the attack methods we constructed can still work in such situation.

Second, knowing the limitation of all existing watermark schemes, we will develop new active timing based correlation schemes that can achieve both better secrecy and performance. We can see that the fundamental problem for the active watermark schemes is to reduce the amount of packet delays while still keeping the robustness under various interference. The simple enhancement we proposed for the interval based scheme can only maintain the watermark true positive rate when no timing perturbation is inserted. Better embedding schemes are needed to tolerate timing perturbations. It is also desirable to derive the mathematical formula between watermark detectability and watermark capability in order to fully understand the trade-off between secrecy and performance. Note in all the existing active watermark schemes, the robustness is introduced by computing arithmetic average to apply central limit theorem. One possible starting point to construct better encoding method is to apply information theory. More specifically, we may be able to use the rate distortion theory to compute the minimal amount of watermark delays needed for certain desired performance of the watermark schemes. Such information will instruct us how the watermark should be embedded into stepping stone flows.

Another thing interests us is that the fact attackers may perform timing analysis could be exploited to construct methods to deter stepping stone attacks. For example, we may try to confuse the timing analysis process by randomly adding delays into lots of flows. The normal network delays attacks observed may be inaccurate and cause wrong decision in watermark detection. Moreover, if the attacker always aborts the connections after he/she suspects being traced, we can simply embedding watermark into all suspicious stepping stone connections. Therefore the attacker may not be able to complete the attack.

Finally, we will investigate how to use the basic ideas of the active watermark schemes to trace through anonymizing networks. A method that uses a watermark scheme to trace peer-to-peer VoIP calls [58] has shown great promise in this direction. However, there are many distinctions between stepping stone attacks and anonymizing networks. For example, the amount of extra delays and timing perturbation an anonymizing network can tolerate could be much smaller than in stepping stone connections. The number of packets in an anonymizing network can be much larger.
Therefore we must conduct further research on the anonymizing networks in order to construct appropriate trace-back approaches.
Bibliography


Appendix A

Frequently Used Symbols

Table A.1: Symbols used in stepping stone connections

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h_i$</td>
<td>a stepping stone host</td>
</tr>
<tr>
<td>$h_1 \leftrightarrow h_2$</td>
<td>a stepping stone connection between $h_1$ and $h_2$</td>
</tr>
<tr>
<td>$h_1 \rightarrow h_2$</td>
<td>a stepping stone flow from $h_1$ to $h_2$</td>
</tr>
<tr>
<td>$h_1 \leftrightarrow \cdots \leftrightarrow h_k$</td>
<td>a stepping stone connection chain from $h_1$ to $h_k$</td>
</tr>
<tr>
<td>$f$</td>
<td>a known stepping stone (upstream) flow</td>
</tr>
<tr>
<td>$f'$</td>
<td>a suspicious downstream flow</td>
</tr>
<tr>
<td>$p_i, t_i$</td>
<td>a packet in flow $f$ and its timestamp</td>
</tr>
<tr>
<td>$p'_i, t'_i$</td>
<td>a packet in flow $f'$ and its timestamp</td>
</tr>
<tr>
<td>$c_i$</td>
<td>a meaningless chaff packet inserted by attacks</td>
</tr>
<tr>
<td>$d_i$</td>
<td>a packet delay between two stepping stones</td>
</tr>
<tr>
<td>$ipd$</td>
<td>an inter-packet delay between two packets in a flow</td>
</tr>
</tbody>
</table>
Table A.2: Symbols used in watermark schemes

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L$</td>
<td>the number of binary watermark bits</td>
</tr>
<tr>
<td>$W = w_1 \ldots w_L$</td>
<td>$L$-bit binary watermark</td>
</tr>
<tr>
<td>$M$</td>
<td>degree of robustness</td>
</tr>
<tr>
<td>$S$</td>
<td>quantization step (quantization scheme)</td>
</tr>
<tr>
<td>$a$</td>
<td>extra watermark delay (probabilistic scheme)</td>
</tr>
<tr>
<td>$\delta$</td>
<td>extra watermark delay (interval scheme)</td>
</tr>
<tr>
<td>$I$</td>
<td>time interval length (interval scheme)</td>
</tr>
<tr>
<td>$O$</td>
<td>offset value (interval scheme)</td>
</tr>
</tbody>
</table>