Active Timing-Based Correlation of Perturbed Traffic Flows with Chaff Packets*

Pai Peng, Peng Ning, Douglas S. Reeves
Department of Computer Science
North Carolina State University

Xinyuan Wang
Department of Information & Software Engineering
George Mason University

Abstract

Network intruders usually launch their attacks through a chain of intermediate stepping stone hosts in order to hide their identities. Detecting such stepping stone attacks is difficult because packet encryption, timing perturbations, and meaningless chaff packets can all be utilized by attackers to evade from detection. In this paper, we propose a method based on packet matching and timing-based active watermarking that can successfully correlate interactive stepping stone connections even if there are chaff packets and limited timing perturbations. We provide several algorithms that have different trade-offs among detection rate, false positive rate and computation cost. Our experimental evaluation with both real world and synthetic data indicates that by integrating packet matching and active watermarking, our approach has overall better performance than existing schemes.

1. Introduction

Network intruders have developed various countermeasures to elude from being discovered. A popular and effective method is to launch attacks through a sequence of intermediate hosts, also known as *stepping stones*. Intruders can connect from one host to another using protocols such as Telnet or SSH, and attack the real victims only on the last host. In this scenario, even if the last host is correctly identified, it could be very difficult to trace back to the real origin. Correlation methods are needed to link the connections between stepping stones together.

Researchers have proposed several approaches to detect stepping stone connections. Early methods are based on comparing the contents of packets [5] [9]. Due to the broad applications of secure protocols such as SSH and IPsec, recent approaches focus on analyzing packet timing characteristics [12] [10] [8] [2] [1] [7] [11].

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

However, existing correlation schemes are still far from being perfect. Attackers may intentionally insert timing perturbations simply through delaying certain packets. Another countermeasure is to insert meaningless padding packets, also called *chaff*. When transmitted through encrypted channels, chaff packets are very difficult to be differentiated from normal packets.

In this paper, we propose an approach that can correlate stepping stone connections when timing perturbations and chaff are introduced simultaneously. Inspired by [7] and [11], we first embed timing-based watermarks into attack flows. We then use packet matching to find all possible corresponding packets in suspicious flows. Correlation results are decided by decoding the watermarks closest to the original ones from all packet combinations. We provide 4 algorithms with different trade-offs among detection rate, false positive rate and computation cost. We experimentally compare our algorithms with the best existing approaches, and show our approach can achieve overall better performance.

In the rest of this paper, section 2 gives out the problem statement. Section 3 describes packet matching process and watermark decoding algorithms. Section 4 provides the experimental evaluations and comparisons. Section 5 reviews related work. Section 6 concludes our paper and points out further research directions.

2. Problem Statement

We use $h_1 \leftrightarrow h_2$ to represent a bi-directional network *connection* between host h_1 and h_2, and $h_1 \rightarrow h_2$ a unidirectional *flow* from h_1 to h_2. A flow is also denoted as f when hosts and directions are not concerned. Given hosts $h_1, h_2, \ldots, h_n$, when a person or a program connects from h_i to h_{i+1}, the sequence of connections $h_1 \leftrightarrow h_2 \leftrightarrow \ldots \leftrightarrow h_n$ is called a *connection chain*. The intermediate hosts in a connection chain are called *stepping stones*. Assuming $j > i$, we call $h_i \rightarrow h_{i+1}$ an *upstream* flow of $h_j \rightarrow h_{j+1}$, and $h_j \rightarrow h_{j+1}$ a *downstream* flow of $h_i \rightarrow h_{i+1}$. Intuitively, information is propagated from an upstream flow to its downstream flows. Timestamp of packet p_i is t_i. Flow f is also represented as the sequence of its packets $\langle p_1, p_2, \ldots, p_n \rangle$. We define the *tracing* problem of a con-

nection chain as given an upstream flow f, to identify its downstream flows.

Currently, the most promising correlation approaches are based on timing analysis. To evade timing analysis, attackers may introduce timing perturbations by delaying some or all packets. Another countermeasure is to insert meaningless chaff packets into a downstream flow. It would be very difficult to distinguish chaff from normal packets when encryption is used.

We propose to investigate tracing techniques that can deal with both timing perturbations and chaff packets. Similar to previous work [2] [7] [1] [11], we focus on interactive connections and assume the maximum timing perturbation attackers can introduce is bounded. Normally packet timestamps captured from different hosts cannot be compared directly because time clocks may not be fully synchronized. To simplify the situation, we assume the skews between different clocks are known so that timestamps can be adjusted for comparison. The timing errors from timestamp adjustment, the maximum perturbations added by attackers, and delays from other sources are collectively represented by a single *maximum delay* Δ. In summary, we have following assumptions in our solution:

1. Every packet in an upstream flow will go to its downstream flow as a single packet.
2. The delay between a packet in an upstream flow and its corresponding packet in a downstream flow is bounded by $[0, \Delta]$. We also call this *timing constraint*.
3. The order of the packets in an upstream flow is kept the same in a downstream flow. We also call this *order constraint*.

3. Proposed Approaches

We adopt the inter-packet-delay (IPD) based watermarking scheme [7], which was originally proposed to defeat timing perturbations. The idea is to embed a unique timing-based watermark into an upstream flow f. If later the same watermark can be detected in another flow f', it is very likely that f' is a downstream flow of f. However, this scheme cannot be applied to chaff directly. Extra chaff packets will make current detection mechanism fail to find the correct location to decode the watermark. Suppose upstream flow f is $\langle p_1, \ldots, p_n \rangle$, and $f' = \langle p'_1, c_1, \ldots, p'_n, c_m \rangle$ is its chaffed downstream flow, the correct corresponding packets $\langle p'_1, \ldots, p'_n \rangle$ form a *subsequence* of f'. Current scheme cannot dig out this subsequence from all other subsequences of n packets. In fact, due to the difficulty of distinguishing chaff from normal packets, it is very unlikely to identify the correct subsequence exclusively.

To defeat chaff, our idea is to find all possible subsequences of f in f', decode a watermark from each of them, and choose the "*best*" one that has the smallest hamming distance to the original watermark. The right subsequence must be chosen sometime so that we can get the correct watermark. By using the "best" watermark, if a downstream flow can be identified before chaff is added, it can still be identified afterward. Actually, it enables us to detect certain flows missed by the basic watermark scheme. On the other hand, since the "best" watermark may be obtained from an incorrect subsequence, the false positive rate may also increase. This is the trade-off in our approach.

3.1. Inter-Packet-Delay Based Watermarking

We briefly introduce the IPD based watermark scheme [7]. A watermark w is embedded into an upstream flow through slightly delaying certain packets. Such changes of timing will then propagate to all of its downstream flows. If w is unique enough, it should be detected in all the downstream flows, but nowhere else, with high probability.

Given a flow $\langle p_1, \ldots, p_n \rangle$, to embed a single watermark bit, we randomly choose $2r$ distinct packets $\langle p_{e_1}, \ldots, p_{e_{2r}} \rangle$, called *embedding packets*, and construct $2r$ packet pairs: $\langle p_{e_i}, p_{e_i+d} \rangle$. d (≥ 1) is a user-selected value. The IPD of pair $\langle p_{e_i}, p_{e_i+d} \rangle$ is defined as:

$$ipd_{e_i} = t_{e_i+d} - t_{e_i}.$$

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

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

Then we should have $E(D) = 0$. r is called *redundancy number*. The bigger r is, the more likely D is equal to 0.

If we increase or decrease D by a value $a > 0$, we can skew the distribution of D. The probability that D is positive or negative is increased. This gives out a way to embed a single watermark bit probabilistically. To embed 0, we decrease D by a so that it is more likely $D < 0$. To embed 1, we increase D by a so that it is more likely $D > 0$. The decrease (increase) of D is achieved by decreasing (increasing) every ipd^1_i and increasing (decreasing) every ipd^2_i by a. The decrease or increase of a single IPD is achieved by delaying the $1st$ or $2nd$ packet in that IPD, respectively. The watermark bit can be decoded by checking the sign of D. Bit 0 (1) is decoded when $D \leq 0$ (> 0). There is a slight probability that a watermark bit cannot be correctly embedded. This probability can be reduced by increasing r.

A l-bit watermark w is embedded by repeating the above procedure l times. Each time a different set of embedding packets should be used. In watermark detection, another

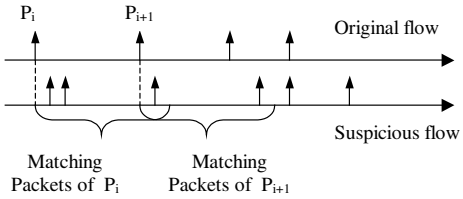

Figure 1. Determining matching packets

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 found. Because watermark location is kept secret from attackers, this scheme is robust against random timing perturbations. However, extra chaff packets will destroy the decoding mechanism.

3.2. Determining Matching Packets

Suppose upstream flow f is $\langle p_1, \ldots, p_n \rangle$, which has watermark w embedded, and suspicious downstream flow f' is $\langle p'_1, \ldots, p'_m \rangle$ ($m \geq n$). For every packet in f, we determine which packet(s) in f' could be its corresponding packet. Based on our assumption that every packet in the upstream flow will go into the downstream flow, if f and f' are in the same connection chain, we can find corresponding packets for all packets. Otherwise, we report they are not in the same connection chain. Because of the difficulty to distinguish normal packets from chaff, we may only obtain some possible corresponding packets through certain packet matching criteria. We call these possible corresponding packets *matching packets*, or simply *matches*.

Timing constraint in our assumptions is used to determine matching packets. It requires that any matching packet p'_j of packet p_i must satisfy: $0 \leq t'_j - t_i \leq \Delta$. All the matching packets of p_i form a set:

$$M(p_i) = \{p'_j \quad | \quad 0 \leq t'_j - t_i \leq \Delta\}.$$

$M(p_i)$ is called the *matching set* of packet p_i in flow f'. This procedure is shown in figure 1.

Matching sets of all packets can be computed quickly. To determine $M(p_{i+1})$, we only need to scan from the first packet in $M(p_i)$. Heuristics can also be used. Suppose p'_j and p'_k are the first and last packet in $M(p_i)$ (i.e., have the smallest and largest timestamp). To find the first packet p'_x in $M(p_{i+1})$, if $t_{i+1} - t_i \leq \frac{\Delta}{2}$, we scan forward from p'_j since p'_x might be closer to it than p'_k. If $\frac{\Delta}{2} < t_{i+1} - t_i \leq \Delta$, we scan backward from p'_k. Otherwise, we scan from p'_{k+1} since $M(p_i)$ and $M(p_{i+1})$ will not overlap. Each packet in f' will be scanned at most twice in the worse case.

For better performance, we want the matching sets to be as smaller as possible. Packet size might be used as an extra constraint for packet matching. For example, when block ciphers in SSH only pad a packet to 16-byte boundary, we may use *quantized* packet size as a constraint, such as multiple of 16 bytes, to determine matching sets. However, using this constraint is inappropriate if attackers can actively add inner-packet paddings.

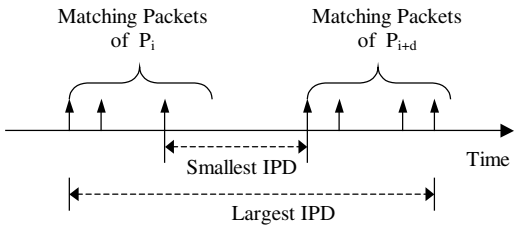

Figure 2. The largest and smallest IPDs

3.3. Computing the "Best" Watermark

After matching sets have been determined, we find the "best" watermark from all possible subsequences in f'. In the following, we discuss several algorithms with different emphases on detection rate, false positive rate or computation cost. We are trying to achieve the best trade-off among these three aspects.

3.4. Algorithm 1: Brute Force Algorithm

The idea can be directly converted into a brute-force algorithm, which forms all subsequences by trying all combinations of matching packets. To satisfy the order constraint, packets $p'_j \in M(p_i)$ and $p'_k \in M(p_{i+1})$ can be in the same subsequence only if $j < k$.

This algorithm obviously suffers from its high computation cost. If $M(p_i)$ has $|M(p_i)|$ packets, the computation cost is approximately: $cost \approx \prod_i^n |M(p_i)|$. We need algorithms with better efficiency.

3.5. Algorithm 2: Greedy Algorithm

Since only the "best" watermark is wanted, we propose a very fast Greedy algorithm, which guarantees to return a watermark whose hamming distance is no bigger than that of the Brute Force algorithm. For each watermark bit, this algorithm only selects the matching packets that are most likely to decode the same bit. If bit 1 is wanted, we use the largest IPDs in the $1st$ group ipd^1, and the smallest IPDs in the $2nd$ group ipd^2. Similarly, to decode 0, we use the smallest IPDs in ipd^1 and the largest IPDs in ipd^2. We then selects the first or the last matching packets to get the desired IPDs as shown in figure 2.

This algorithm has very low computation cost because it only form a single subsequence. It also has very good detection rate and can identify every flow that can be identified by the Brute Force algorithm. However, by simply selecting the most appropriate matches, the subsequence constructed

may be conflict with the order constraint, which leads to potential high false positive rate.

3.6. Algorithm 3: Greedy$^+$ Algorithm

In this algorithm, we use the order constraint to decrease the false positive rate of the Greedy algorithm, while still keeping high detection rate and low computation cost. By only constructing a very small portion of all possible subsequences that will most likely give the best result, watermarks can still be computed efficiently. Compared with Greedy, Greedy$^+$ reduces the false positive rate at the cost of slightly decreased detection rate.

The Greedy$^+$ algorithm has four phases. First, matching sets are further simplified by removing duplicate first or last packets. For example, suppose both $M(p_1)$ and $M(p_2)$ are $\{p'_1, p'_2\}$. p'_1 can never be used as a matching packet for p_2 because then p_1 will have no match to choose. Similarly, p'_2 cannot be used for p_1 either. Such packets can be safely remove without affecting final result.

Second, we use the Greedy algorithm to compute a watermark w_g and compare to the original watermark w. If the hamming distance is larger than the threshold, we report it is not a stepping-stone connection. Obviously, the unmatched bits of w_g will not match in the Greedy$^+$ algorithm either. So we will only focus on the rest of watermark bits.

Third, we adjust the matching packets selected by the Greedy algorithm to eliminate the conflict with the order constraint. We always allow a packet to choose its first match, and re-select for others that use their last matches. This process starts from the last embedding packet, for which we can alway stick to its current selection. If the current match has no conflict or is the first in the matching set, we stick to it. Otherwise, we switch to the last match that has no conflict with packets later than it. For example, suppose $M(p_1) = \{p'_1, p'_2, p'_3, p'_4\}$, $M(p_2) = \{p'_3, p'_4, p'_5\}$, $M(p_3) = \{p'_4, p'_5, p'_6\}$, p_1 and p_3 are embedding packets and their current matches are both p'_4. We begin with p_3 and stick to p'_4. For p_2 we select the last non-conflict match p'_3. Then p_1 has to select the last non-conflict match p'_2. We then decode a new watermark w_b. If w_b has a hamming distance less than or equal to the threshold, we report this is a stepping-stone flow. Otherwise, we go to the final phase to further adjust the selections of matching packets.

To speed up later computation, we record all the IPD differences D when computing w_b, except for those bits that will never match. Recall that we decode 1 if $D > 0$, and decode 0 if $D \leq 0$. These IPD differences are divided into two groups D^+ and D^- based on whether their corresponding watermark bits match the original watermark bits. If the ith bits of w_b and w are the same, we put D_i into D^+. Otherwise, we put D_i in D^- (whether D_i is positive or negative is not related with the group it belongs to). Apparently, we can get a better watermark by making a $D \in D^-$ change its sign. Among all $D \in D^-$, the one with the smallest absolute value is the easiest to be changed.

In the final phase, we focus on the embedding packets used for those $D \in D^-$. We start with D_j, which has the smallest absolute value in D^-. Suppose p_k is its last embedding packet.

1. If the current selection of matching packet of p_k is the same as in the Greedy algorithm, we stick to it, and continue for the previous embedding packet.

2. Otherwise, we select the next matching packet of p_k, if any. Since other packets will be affected, we have to re-select their matches too. If this can improve D_j and does not make any $D \in D^+$ change its sign, we then switch to the new matching packet.

3. Repeat step 2 for p_k until a) D_j changes its sign, b) no more matching packets for p_k, or c) making changes will not necessarily improve the watermark. If watermark bit j is now match, we then forward to the next smallest $D \in D^-$. Otherwise, we repeat this procedure for the previous embedding packet of D_j. We terminate whenever the hamming distance is reduced to the same as the threshold.

Unlike the Brute Force algorithm, we avoid time consuming backtracking. The selection of one embedding packet is determined without considering the changes of other packets. Because we use such heuristics to always adjust those packets that are most likely to generate a better result, usually the watermark obtained is very close to the "best" one.

3.7. Algorithm 4: Greedy*

We can obtain the actual "best" watermark by only changing the last phase of the Greedy$^+$ algorithm. After the embedding packets for those $D \in D^-$ are determined, we can enumerate all possible combinations of their matching packets and decode the "best" watermark. Although this is similar to the Brute Force algorithm, the previous phases of the algorithm will reduce the size of searching space significantly so that we can expect a much better efficiency. To bound the worst case execution time, we allow users to set up a maximum bound. If it cannot finish within the bound, it returns the best watermark obtained so far.

3.8. Complexity Analysis

In this section, we investigate the time complexity of different algorithms. The complexity of all algorithms are affected by the number of packets in the flow pairs, the number of chaff packets added, and the maximum delay Δ.

In the packet matching process, each packet in the suspicious flow f' is checked at most twice. Thus, the worst case

complexity is $O(m)$, where m is the number of packets in f'. If f' is in the same connection chain as f, m is equal to the sum of packet number n in f and the number of chaff c: $m = n + c$. The average packet number in a matching set can be approximated by the product of the average packet arrival rate $\lambda_{f'}$ in f', and the maximum delay Δ: $\lambda_{f'} \cdot \Delta$. When f' is actually the chaffed flow, $\lambda_{f'}$ is equal to the average packet arrival rate of f plus the average arrival rate of chaff: $\lambda_{f'} = \lambda_f + \lambda_c$.

The Greedy algorithm only checks every embedding packet once, so its complexity is $O(n)$. For the Greedy$^+$ algorithm, the most time consuming part is the last phase. Suppose the embedding packets left to be adjusted are $p_{e_1}, \ldots, p_{e_k}$. At most $\sum_{i=1}^{k} |M(p_{e_i})|$ packets have to be scanned. This is bounded by $O(n \cdot \lambda_{f'} \cdot \Delta)$. Similarly, the last phase of Greedy* algorithm needs to check $\prod_{i=1}^{k} |M(p_{e_i})|$ packets. However, both algorithms normally have much better performance than their worst case scenarios.

4. Experiments

We evaluate the performance of our algorithms through *detection rate*, *false positive rate* and *computation cost*. Both real world and synthetic flows are used. We compare our algorithms with the basic watermark scheme [7] and scheme S-IV [11], because they are the best existing active and passive schemes, respectively. Only the timing constraint is used in packet matching process. We expect the false positive rate and computation cost to decrease dramatically if quantized packet size constraint can also be used.

4.1. Real World Data Set

We use 91 real SSH/Telnet traces derived from Bell Labs-1 Traces of NLANR [3]. All traces have more than 1,000 packets. For each trace, we first embed a randomly generated watermark[1]. We then add 9 different timing perturbations, which is uniformly distributed with a maximum delay from 0 (i.e., no perturbation) to 8 seconds. For each timing perturbation, we add 11 different kinds of Poisson distributed chaff packets. The arrival rate of chaff packets λ_c is from 0 (i.e., no chaff) to 5. The maximum delay Δ is set the same as maximum timing perturbation. The parameters are shown in table 1. In Greedy* algorithm, we also set the bound of computation cost to 10^6.

Detection Rate is evaluated by calculating the correlation between each original flow and its perturbed and chaffed flows. Figure 3 shows the detection rate changing with λ_c when Δ is 7 seconds. It clearly shows that chaff will destroy basic watermark scheme. The Greedy algorithm has the best detection rate. Greedy$^+$ and Greedy* outperform

[1]No watermark is embedded when scheme S-IV is evaluated.

Table 1. Experiment parameters

Δ	$0, 1, 2, 3, 4, 5, 6, 7, 8$ (second)
λ_c	$0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5$
Watermark	24 bits
Redundancy r	4
WM threshold	7
WM delay a	600ms
S-IV threshold	3 seconds

scheme S-IV when there is no chaff. Although theoretically Greedy* should have better detection rate than Greedy$^+$, it performs slightly worse under the bound of computation cost. Figure 4 shows the detection rate changing with Δ when $\lambda_c = 3$. S-IV shows significant lower detection rate than our algorithms when there is no chaff, and fails to reach 100%. By using the "best" watermark, the increase of chaff helps the detection rate. However, it also increases the false positive rate, as shown in the following.

False Positive Rate is evaluated by correlating each original flow with the perturbed and chaffed flows of other 90 flows. Figure 5 shows the false positive rate changing with λ_c when $\Delta = 7s$. Figure 6 shows the false positive rate changing with Δ when $\lambda_c = 3$. Unsurprisingly, the Greedy algorithm shows the worst false positive rate. Except for the basic watermark scheme, the false positive rates of other algorithms increase with λ_c and Δ. Both Greedy$^+$ and Greedy* show better performance than scheme S-IV. The false positive rates of Greedy$^+$ and Greedy* are up to about 40% lower than that of S-IV, as shown in figure 6.

Computation Costs. To eliminate the bias of implementation details, we define *computation cost* as the number of packets had to be accessed to compute the "best" watermark or the smallest deviation for scheme S-IV. We also include the packet matching process since it is a time consuming step in our approaches and S-IV. We distinguish the computation costs between correlated and uncorrelated flows.

For correlated flows, figure 7 and 8 show computation costs changing with λ_c and Δ, respectively. The Greedy algorithm has constant and the smallest cost. Greedy* has a bump in its curve. It is because when more chaff packets are added, matching sets grow bigger and more packets need to be checked. However, if enough chaff is added, our optimization techniques can give the results quickly and make the cost decrease. Greedy$^+$ also shows a smaller bump for the same reason. There exist certain cases that Greedy* fails to finish within its bound of computation cost. Both Greedy$^+$ and Greedy* have up to about 40 times lower costs than S-IV.

Figure 9 and 10 show the costs for uncorrelated flows. It is worth noticing that we may have 0 as the cost[2]. It is

[2]In order to draw figures in logarithm scale, we change 0 to 1.

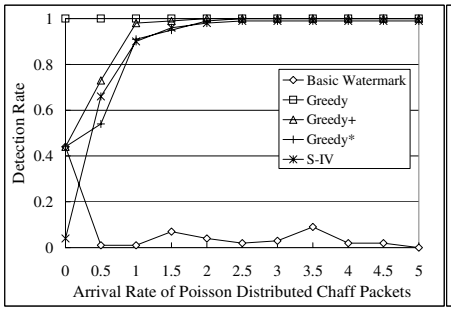

Figure 3. Detection rate changing with λ_c, $\Delta = 7s$

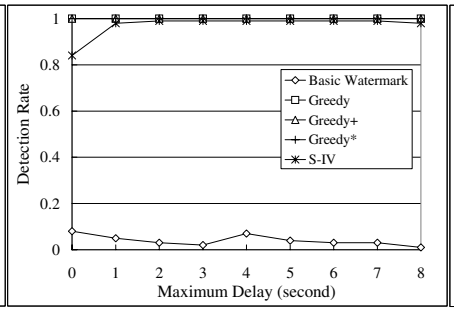

Figure 4. Detection rate changing with Δ, $\lambda_c = 3$

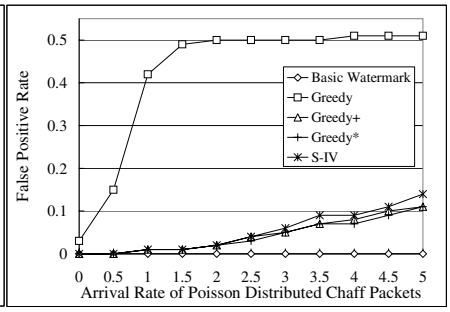

Figure 5. False positive rate changing with λ_c, $\Delta = 7s$

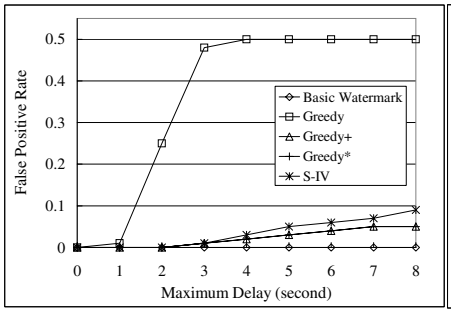

Figure 6. False positive rate changing with Δ, $\lambda_c = 3$

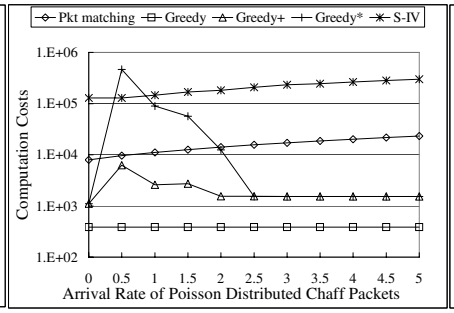

Figure 7. Costs changing with λ_c, $\Delta = 7s$, correlated flows

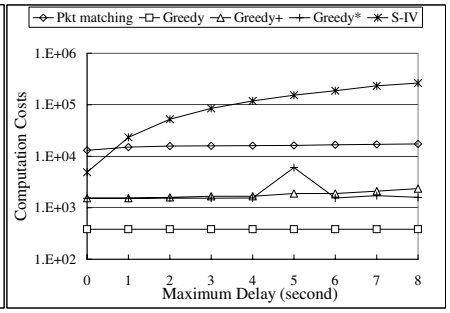

Figure 8. Costs changing with Δ, $\lambda_c = 3$, correlated flows

because if the matching process fails to find any matching packet, we can immediately have negative correlation result. The cost of Greedy* reaches its maximum bound rapidly when chaff or delay is big. Greedy$^+$ is still up to about 2 times faster than S-IV.

4.2. Synthetic Data Set

We have repeated the above experiments using 100 synthetic tcplib traces. The results are consistent with the real world data. Please refer to our full version paper [4] for complete information.

4.3. Overall Performance

The Greedy$^+$ algorithm has shown overall the best tradeoff among detection rate, false positive rate and computation cost. Greedy* suffers from high computation cost, especially when it fails to find correlation. Scheme S-IV has worse false positive rate than both Greedy$^+$ and Greedy*. It also has higher computation cost than Greedy$^+$.

5. Related Work

The problem of detecting interactive stepping stones was first formulated by Staniford and Heberlein [5]. Their content-based approach compared thumbprints created from packet payload. Another content-based scheme was Sleepy Watermark Tracing [9], which injected non-displayable contents into packets. These methods are vulnerable to encrypted traffic such as SSH connections.

More recent schemes focused on timing characteristic of packets. Zhang and Paxson [12] proposed an ON/OFF based approach to correlate encrypted traffic. Yoda and Etoh [10] proposed a deviation-based scheme which calculated deviation between an attacking flow and all other flows appeared around the same time. Wang et al. [8] showed that timing characteristics of IPDs were preserved across multiple stepping stones, and could be used for correlation. These methods are all vulnerable to timing perturbations.

Donoho et al. [2] investigated the theoretical limits of the attackers' ability to disguise their traffic through timing perturbations and chaff packets. Wang and Reeves [6] proposed an active watermarking scheme that was robust to random timing perturbations. They identified the tradeoffs between the correlation effectiveness, timing perturbations, and packet number needed. Blum et al. [1] proposed to correlate stepping stone connections by counting the packet number differences in certain time intervals. Wang et al. also proposed a probabilistic watermarking scheme [7] with even timing adjustments and better true positive rate.

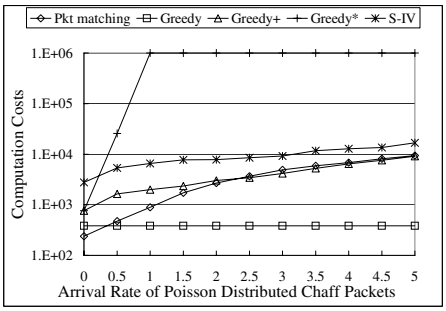

Figure 9. Computation costs changing with λ_c, $\Delta = 7s$, uncorrelated flows

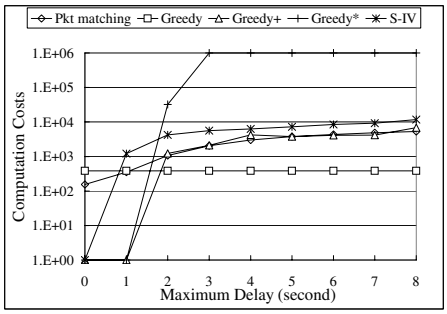

Figure 10. Computation costs changing with Δ, $\lambda_c = 3$, uncorrelated flows

Zhang et al. [11] proposed several algorithms to detect stepping stone connections when there existed timing perturbation or/and chaff. Their algorithms were also based on finding possible corresponding packets. Unlike active schemes, their passive schemes do not require traffic manipulation, thus are less noticeable. However, using active watermarking, our algorithms can achieve better performance with less computation costs.

6. Conclusions

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 introduce our correlation scheme based on packet matching and active timing-based watermarking. We have developed a series of algorithms to compute the "best" watermark. These algorithms have different emphases on detection rate, false positive rate or computation cost. Through experiments, we have demonstrated the effectiveness and efficiency of our algorithms. We have also compared our algorithms with existing best schemes, and shown that overall Greedy$^+$ algorithm has better performance.

Our algorithms (and several previous approaches) reply on the assumption that packets should not be lost or combined together after passing through a stepping stone. However, packet loss or re-packetization are common when packets arrive too closely or system load is high. In this case, our scheme may not always return the desired result. In the future, we will focus on correlation methods that will work under packet loss and re-packetization.

Acknowledgment. We would like to thank Dr. Yong Guan for providing a draft of his work [11], and thank anonymous reviewers for their helpful comments.

References

[1] A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones with maximum delay bound: algorithms and confidence bounds. In *Proceedings of RAID'04*, 2004.

[2] D. Donoho, A. Flesia, U. Shankar, V. Paxson, J. Coit, and S. Staniford. Multiscale stepping-stone detection: detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *Proceedings of RAID'02*, 2002.

[3] NLANR trace archive. http://pma.nlanr.net/traces/long/.

[4] P. Peng, P. Ning, D. S. Reeves, and X. Wang. Active timing-based correlation of perturbed traffic flows with chaff packets. Technical Report TR-2005-11, Department of Computer Science, NC State Univ., 2005.

[5] S. Staniford-Chen and L. T. Heberlein. Holding intruders accountable on the Internet. In *Proceedings of IEEE S&P 95*, pages 39–49, Oakland, CA, 1995.

[6] X. Wang and D. S. Reeves. Robust correlation of encrypted attack traffic through stepping stones by manipulation of inter-packet delays. In *Proceedings of CCS'03*, pages 20–29, 2003.

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

[8] X. Wang, D. S. Reeves, and S. F. Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *D. Gollmann, G. Karjoth and M. Waidner, editors, 7th European Symposium on Research in Computer Security - ESORICS 2002*, 2002.

[9] X. Wang, D. S. Reeves, S. F. Wu, and J. Yuill. Sleepy watermark tracing: An active network-based intrusion response framework. In *Proceedings of 16th International Conference on Information Security (IFIP/Sec'01)*, 2001.

[10] K. Yoda and H. Etoh. Finding a connection chain for tracing intruders. In *F. Guppens, Y. Deswarte, D. Gollmann and M. Waidners, editors, 6th European Symposium on Research in Computer Security - ESORICS 2000*, 2000.

[11] L. Zhang, A. Persaud, A. Johnson, and Y. Guan. Stepping stone attack attribution in non-cooperative IP networks. Technical Report 2005-02-1, Department of Electrical and Computer Engineering, Iowa State University, 2005.

[12] Y. Zhang and V. Paxson. Detecting stepping stones. In *Proceedings of 9th USENIX Security Symposium*, pages 171–184, 2000.