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 based intrusions grow rapidly in recent years, despite the huge amount of resources that have been put into network and information security. To hide their identities, intruders have developed various countermeasures to elude from being discovered. For example, they could spoof the source IP addresses in their attacking flows to conceal the hosts where the attacks are launched. Another 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 pass the malicious instructions without disclosing the sources of attacks. Only on the last host, they will issue the attack actions toward the real victims. In this scenario, even if the last host can be correctly identified, it could be very difficult to trace back to the real origin behind these stepping stones. Correlation methods are needed to link the connections between stepping stones together.

Researchers have proposed many approaches to detect attacks through stepping stone connections. Early work includes correlation methods based on comparing the packet contents in suspicious network flows [7] [11]. Due to the broad applications of secure protocols such as SSH and IPsec, recent approaches focus on analyzing packet timing characteristics in interactive stepping stone connections [14] [12] [10] [4] [1] [9] [13].

However, existing correlation schemes are still far from being perfect. To defeat packet timing based approaches, attackers could intentionally insert timing perturbations simply through delaying certain packets. It may severely destroy the timing characteristics among corresponding flows. Another countermeasure is to insert meaningless padding packets, also called *chaff*. Such chaff packets are very difficult to be differentiated from normal packets, especially when they are transmitted through encrypted channels.

In this paper, we propose a practical solution that can correlate stepping stone connections even if both timing perturbations and chaff packets are introduced at the same time. Inspired by [9] and [13], we first actively embed timing-based watermarks into upstream flows. Then packet matching is used to find all possible corresponding packets in suspicious flows. Correlation results are determined by computing 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 that can deal with timing perturbation or chaff. Our results indicate that by integrating the ideas in [9] and [13] we can achieve better performance.

The rest of this paper is organized as follows. 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 on stepping-stone correlation. Section 6 concludes our paper and points out further research directions.

[*]This work is supported by Advanced Research & Development Agency (ARDA) under contract # NBCHC030142

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 $h_1 \rightarrow h_2$ as a unidirectional *flow* from h_1 to h_2. A flow is also denoted as f when hosts and directions are not concerned. Given a series of hosts $h_1, h_2, \ldots, h_n$, when a person or a program sequentially 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 flow $h_i \rightarrow h_{i+1}$ 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. We use t_i to represent the timestamp of packet p_i. The sequence of packets in a flow f is represented as $\langle p_1, p_2, \ldots, p_n \rangle$ ($t_i \leq t_j$ for $1 \leq i < j \leq n$). We define the *tracing* problem of a connection chain as given an upstream flow f, to identify its downstream flows.

Solutions to the tracing problem are particularly useful to pin down attackers who launch their attacks through stepping stones. Currently, the most promising approaches are correlation methods based on timing analysis. To evade timing analysis, attackers may introduce timing perturbations by delaying some or all packets to change the timing characteristics of downstream flows. Another countermeasure is to insert meaningless chaff packets into a downstream flow. For example, adding chaff packets c_i into flow $\langle p_1, p_2, \ldots, p_n \rangle$ may lead to $\langle p_1, c_1, p_2, c_2, c_3, \ldots, p_n, c_m \rangle$. Because it is very difficult to distinguish chaff from normal packets when encryption is used, upstream and downstream flows may look very differently.

We propose to investigate tracing techniques that can deal with both timing perturbations and chaff packets. Similar to previous work [4] [9] [1] [13], we focus on interactive connections and assume the maximum timing perturbation attackers can introduce is bounded. Delays may also come from the networks or intermediate hosts that the connection chain passes through. It is difficult to guarantee the synchronization of the clocks of hosts where packets are captured; this prevents us from comparing packet timestamps directly. To simplify the situation, we assume the skews between different clocks are known so that we can adjust the timestamps of packets from different hosts 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

In our approaches, we adopt the inter-packet-delay (IPD) based watermarking scheme [9], which was originally proposed to defeat timing perturbations. The basic idea is to first embed a unique timing-based watermark into an upstream flow f. If later we can detect the same watermark in another flow f', it is very likely that f' is a downstream flow of f. By choosing different watermark parameters, the detection rate and false positive rate are controllable. However, this watermark scheme cannot be directly used when there are chaff packets. It is because watermark detection requires that we know exactly which packets in f' should be used to decode. When extra chaff packets are present, current detection mechanism fails to find the correct packets. In other words, suppose an upstream flow $f = \langle p_1, \ldots, p_n \rangle$, and $f' = \langle p'_1, c_1, c_2, \ldots, p'_n, c_m \rangle$ is the chaffed downstream flow of f, the corresponding packets $\langle p'_1, \ldots, p'_n \rangle$ form a *subsequence* of f'. Current scheme cannot dig out the correct subsequence from all possible subsequences with n packets. In fact, due to the difficulty of distinguishing chaff from normal packets, it is very unlikely, if possible, to identify the correct subsequence exclusively.

To defeat chaff, our idea is to find all possible subsequences of f in f', and decode a watermark from each of them. From all these watermarks, we choose the "*best*" one, which is the closest to the original watermark in terms of hamming distance. The rationale is that by using all possible subsequences, it is guaranteed that the right subsequence will be chosen sometime so that we can get the desired watermark. So by using the "best" watermark, if a downstream flow can be identified before chaff packets are added, we can still identify that flow afterward. Actually, it even enables us to detect certain flows missed by the basic watermark scheme. On the other hand, since the "best" watermark may be obtained from another (incorrect) subsequence, the false positive rate may also increase. This is the trade-off between detection rate and false positive rate in our approaches.

3.1 Inter-Packet-Delay Based Watermark Scheme

Before discussing our approaches, we first briefly introduce the IPD based watermarking scheme [9]. This scheme was proposed to provide a robust correlation method under timing perturbations. The idea is to actively embed a watermark w into an upstream flow by slightly delaying certain packets. Such changes of timing will then propagate to all of its downstream flows. If w is unique enough, we should detect w in all the downstream flows, but nowhere else, with a high probability.

Given a flow $\langle p_1, \ldots, p_n \rangle$, to embed a single watermark bit, we first randomly choose $2r$ distinct packets $\langle p_{e_1}, \ldots, p_{e_{2r}} \rangle$, and construct $2r$ packet pairs: $\langle p_{e_i}, p_{e_i+d} \rangle$, where $1 \leq e_1 < ... < e_{2r} \leq n - d$. The packets chosen to embed the watermark are called *embedding packets*. Here $d \geq 1$ is a user-selected offset value. The IPD of packet pair $\langle p_{e_i}, p_{e_i+d} \rangle$ is defined as:

$$ipd_{e_i} = t_{e_i+d} - t_{e_i}, (i = 1, \ldots, 2r). \tag{1}$$

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

$$D = \frac{1}{2r} \sum_{i=1}^{r} (ipd_i^1 - ipd_i^2) \tag{2}$$

Then we should have $E(D) = 0$. Here r is called *redundancy number*. The bigger r is, the more likely D is equal to 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 so that it is more likely to have $D < 0$. To embed bit 1, we increase D by a so that it is more likely to have $D > 0$. The decrease of D is achieved by decreasing every ipd_i^1 and increasing every ipd_i^2 by a; the increase of D is achieved by increasing every ipd_i^1 and decreasing every ipd_i^2 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. This probability can be reduced by increasing r.

A l-bit watermark w can be embedded by repeating the above procedure of embedding one watermark bit l times. Each time a different set of embedding packets should be used. All these embedding packets form a subsequence of f. In watermark detection, another l-bit watermark w' is decoded from a suspicious flow and compared with w. If the hamming distance h between w and w' is less than or equal to a pre-defined threshold, we report a stepping-stone flow is found. Because the packets used to embed the watermark are kept secret from attackers, this watermark scheme is robust against random timing perturbations. However, extra chaff packets will destroy the decoding mechanism. In this paper, we use packet matching to find possible corresponding packets and use the "best" watermark to determine the correlation result.

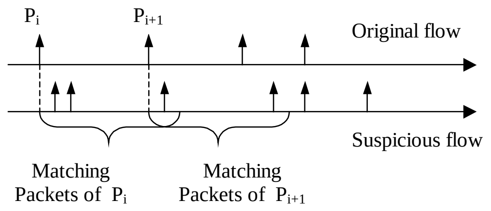

Figure 1. Determining matching packets

3.2 Determining Matching Packets

Suppose the upstream flow is $f = \langle p_1, p_2, \ldots, p_n \rangle$, which has been embedded with watermark w, and the suspicious downstream flow is $f' = \langle p'_1, p'_2, \ldots, p'_m \rangle$ $(m \geq n)$. Before we can form all possible subsequences of f', for every packet in f, we first determine which packet(s) in f' could be its corresponding packet. According to our assumption that every packet in the upstream flow will go into the downstream flow, if f and f' are actually in the same connection chain, we must be able to find the corresponding packets for all packets in f. 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*. If for any packet no matching packets can be found, we directly return that f and f' are not in the same connection chain.

We use the timing constraint in our assumptions to determine the matching packets. Since the timing constraint requires that the delays between corresponding packets is bounded by $[0, \Delta]$, any matching packet p'_j of packet p_i must satisfy: $0 \leq t'_j - t_i \leq \Delta$. All matching packets of p_i will form a set, which is defined by:

$$M(p_i) = \{p'_j | 0 \leq t'_j - t_i \leq \Delta\} \tag{3}$$

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

To compute all the matching sets, f and f' only need to be scanned once. First, we scan f' from the beginning to determine $M(p_1)$ using equation 3. Packets in a matching set is sorted so that the first packet has the smallest timestamp and the last one has the largest timestamp. Suppose in $M(p_1)$, the first and last matching packet are p'_i and p'_k respectively $(i < k)$. To compute $M(p_2)$, we do not need to re-scan f' from the beginning. Because $t_1 < t_2$, the first packet to be checked is p'_i, and no packets before this packet can satisfy the timing constraint.

Based on the timestamps of p_1 and p_2, heuristics can be used to further reduce the number of packets to be scanned for $M(p_2)$. Suppose the first matching packet in $M(p_2)$ is p'_j. To determine p'_j, if $t_2 - t_1 \leq \frac{\Delta}{2}$, we should scan forward from p'_i because it is more likely that p'_j is close to p'_i than p'_k. After p'_j is found, all packets from p'_j to p'_k fulfill equation 3. They are directly added to $M(p_2)$. If $\frac{\Delta}{2} < t_2 - t_1 \leq \Delta$, we scan backward from p'_k to determine the first half of $M(p_2)$, then scan forward from p'_{k+1} to determine the second half of $M(p_2)$. If $t_2 - t_1 > \Delta$, we start directly from p'_{k+1} since $M(p_1)$ and $M(p_2)$ will not share common packets. Similarly, we can determine the matching sets for the rest of packets in f. Using such heuristics, each packet in f' will be checked at most twice in the worse case.

It is desirable to further reduce the size of matching sets to improve the correlation performance. Besides the timing constraint, it is possible to use packet size as a constraint in packet matching. For example, in Telnet connections, the sizes of corresponding packets are usually kept the same. Even when SSH protocol is used, block ciphers normally only pad a packet to 8-byte or 16-byte boundary. In such case, we may use *quantized* packet size, such as multiple of 16 bytes. However, if attackers can actively add inner-packet paddings, using packet size constraint is inappropriate because the correlation may be completely removed. Nevertheless, keeping packet size constraint as an option will at least increase the inconvenience of attackers.

3.3 Computing the "Best" Watermark

After the matching sets of all packets in flow f have been determined, we will 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.

3.3.1 Algorithm 1: Brute Force Algorithm

The simplest algorithm is to directly form all subsequences by brute-forcely trying all combinations of matching packets. Because we need to maintain the order constraint, not all combinations can form a legitimate sequence. Different matching sets are likely to share common packets, especially when the maximum delay Δ is large. We need to guarantee that chosen matching packets will not incur conflict with the order constraint. If packet $p'_j \in M(p_i)$ and packet $p'_k \in M(p_{i+1})$, p'_j and p'_k can be selected in the same subsequence only if $j < k$. For example, if $M(p_1) = \{p'_1, p'_2\}$ and $M(p_2) = \{p'_1, p'_2, p'_3\}$, the only

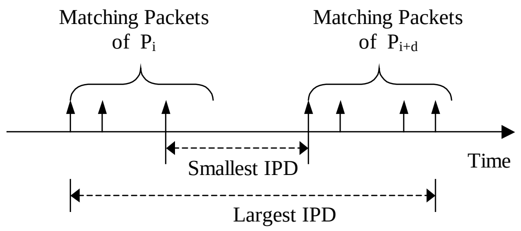

Figure 2. The largest and smallest IPDs

legitimate subsequences are $\langle p'_1, p'_2 \rangle$, $\langle p'_1, p'_3 \rangle$, and $\langle p'_2, p'_3 \rangle$. After all subsequences are created, we can decode every watermark then obtain the "best" one.

This Brute Force algorithm obviously suffers from its high computation cost. If the number of packets in each matching sets is represents as $|M(p_i)|$, the computation cost of the brute force algorithm can be approximated by: $cost \approx \prod_i^n |M(p_i)|$, which is unacceptable when Δ or the number of chaff packets added is large. In the following, we propose several algorithms with better computation efficiency.

3.3.2 Algorithm 2: Greedy Algorithm

Since only the "best" watermark is wanted, we propose a very fast Greedy algorithm that always uses the most "appropriate" packets to decode. This algorithm guarantees to return a watermark whose hamming distance is no bigger than that computed from the Brute Force algorithm.

Instead of trying all possible subsequences, the Greedy algorithm only selects those matching packets that are most likely to generate the "best" watermark. In the IPD watermark scheme, suppose $ipd_i = t_{i+d} - t_i$ is used to embed the watermark. To decode watermark bit 1, it is desirable that the IPDs in the first group (ipd^1) is bigger, and the IPDs in the second group (ipd^2) is smaller. In this case, we want all IPDs in the first group is the largest, and all IPDs in the second group is the smallest. Similarly, to decode watermark bit 0, we want all IPDs in the first group to be the smallest, and all IPDs in the second group to be the largest. If the largest IPD is wanted, we should use the first matching packet of packet p_i, and the last matching packet of packet p_{i+d}. If the smallest IPD is wanted, we would choose the last match of p_i and the first match of p_{i+d} to get the smallest IPD. The largest and smallest IPDs are shown in figure 2.

Based on this idea, we label the embedding packets with a *triple* according to how they are used to embed the watermark. The first element (0 or 1) indicates which watermark bit is embedded. The second element (1 or 2) indicates whether this packet is used in the first or second IPD group. The last element (1 or 2) indicates whether this packet is the first or the second packet in the IPD. From the previous discussion, it is easy to see that for packets labeled with

$(0,1,2)$, $(0,2,1)$, $(1,1,1)$, and $(1,2,2)$, we should choose the first packets in the matching sets. For packets labeled with $(0,1,1)$, $(0,2,2)$, $(1,1,2)$, and $(1,2,1)$, we should choose the last packet.

According to these labels, the Greedy algorithm then selects the first or the last matching packets to form one subsequence, and computes the watermark. Unlike the Brute Force algorithm, the order constraint is not enforced here. The major advantage of this algorithm is its extreme low computation cost, because it only needs to form a single subsequence. This algorithm also has very good detection rate. Since the watermark it computes will not have larger hamming distance than that computed by the brute force algorithm, it can identify every flow that can be identified by the Brute Force algorithm. However, the false positive rate of this algorithm is high, because the packets selected may not satisfy the order constraint.

3.3.3 Algorithm 3: Greedy$^+$ Algorithm

Although the Greedy algorithm can provide very good detection rate and is very efficient in computation, the potential high false positive rate makes it not so desirable, especially when the number of chaff packets is large. However, the idea to always choose the most desired matching packets is very useful to compute the "best" watermark quickly. In this algorithm, we decrease the false positive rate by utilizing the order constraint, while still keeping the high detection rate and low computation cost. The Greedy algorithm is revised so that not one but multiple subsequences will be created. We make sure that every subsequence satisfies the order constraint. We call this algorithm *Greedy*$^+$. Unlike the Brute Force algorithm, Greedy$^+$ only forms a very small portion of all possible subsequences, so that the watermark can still be computed very efficiently. We also ensure the embedding packets that can potentially give us a better watermark will always be checked first. Although only a small portion of all subsequences is formed, the watermark obtained should be very close to the one obtained by the Brute Force algorithm. Compared with Greedy, Greedy$^+$ reduces the false positive rate at the cost of slightly decreased detection rate.

Greedy$^+$ algorithm has five phases. First, the matching sets obtained from equation 3 is further simplified to improve performance. We have observed that although certain packets are included in a matching set, they may never be used in any subsequence because doing so will invalidate the order constraint. For example, if $M(p_1) = \{p'_1, p'_2\}$ and $M(p_2) = \{p'_1, p'_2, p'_3\}$, the only legitimate subsequences of $\langle p_1, p_2\rangle$ are $\{\langle p'_1, p'_2\rangle,\ \langle p'_1, p'_3\rangle,\ \langle p'_2, p'_3\rangle\}$. Although $p'_1 \in M(p_2)$, it cannot be selected as a matching packet of p_2, because p'_1 is also the first match in $M(p_1)$. Similarly, suppose now $M(p_1) = \{p'_1, p'_2, p'_3\}$. p'_3 cannot be used for p_1 either, because it is the last match in $M(p_2)$. We can remove all such packets to get smaller matching sets. For the above example, the matching sets will be changed to $M(p_1) = \{p'_1, p'_2\}$ and $M(p_2) = \{p'_2, p'_3\}$. The elimination of extra matching packets can be combined with the packet matching process to reduce extra overhead. To remove duplicated first packets, instead of starting from the first matching packet p'_j of p_i, we begin with p'_{j+1}. To remove duplicated last packets, we only need one extra comparison between the last packets in $M(p_{i+1})$ and $M(p_i)$.

Second, we use the Greedy algorithm to compute a watermark w_g. If the hamming distance between w_g and the original watermark w is larger than the threshold, we quit immediately and return that it is not in the same chain of stepping-stone connections. From w_g, it is also known which watermark bits will never match no matter how we choose the matching packets. Such watermark bits will be omitted from later computation. We only focus on the watermark bits that match with w.

Third, we check whether the matching packets chosen by the Greedy algorithm incur any conflict with the order constraint. For example, if $M(p_1) = \{p'_1, p'_2\}$ and $M(p_2) = \{p'_2, p'_3\}$, the Greedy algorithm may choose p'_2 as the matching packet for both p_1 and p_2. If a matching packet does not have conflict with other matching packets, then it is safe to stick to this selection, and we do not need to worry about this packet later. Otherwise, we will have to adjust the existing selection to satisfy the order constraint. Suppose p'_k is the matching packets of p_i selected by the Greedy algorithm, and $I(p_i) = k$ is a function returning the index of matching packets. Obviously, for two consecutive embedding packets p_i and p_j $(i < j)$, if $I(p_i) \geq I(p_j)$, there will be conflict with the order constraint. Even if $I(p_i) < I(p_j)$, there may still exist conflict because the non-embedding packets between p_i and p_j also need to have exclusive matches. If there are x non-embedding packets in between, it is required that $I(p_i) < I(p_j) - x$ to ensure every non-embedding packet has at least one match.

Fourth, we form the basic subsequence by adjusting the matches selected by the Greedy algorithm to maintain the order constraint. We make sure that when there are conflicts, we always allow a packet to choose its first match, and adjust those using their last matches. This process starts from the last embedding packet, for which we can alway stick to the current selection. If the matching packet of embedding packet p_i has no conflict or is the first one in the matching set, we stick to it. Otherwise, for a embedding packet with conflict or a non-embedding packet, we choose the last match that will not incur any conflict with packets later than it. For example, suppose $M(p_1) = \{p'_1, p'_2, p'_3, p'_4\}$, $M(p_2) = \{p'_3, p'_4, p'_5\}$, and $M(p_3) = \{p'_4, p'_5, p'_6\}$. p_1 and p_3 are embedding packets and they both choose p'_4 as the match. We begin with p_3, and it can select its best match p'_4. p_2 is non-embedding packet, so it selects the last non-conflict packet p'_3. Then p_1 has to select the last non-conflict packet p'_2. From the basic subsequence, we decode a watermark w_b. If it has a hamming distance less than the threshold, we can immediately quit and return that this is a stepping-stone flow. Otherwise, we adjust the selections of matching packets for a better match of the watermark.

To speed up later computation, we record all the IPD differences D when we compute w_b. Recall that watermark bit is decoded as 1 if $D > 0$, and 0 if $D \leq 0$. We divide D in two groups D^+ and D^- based on whether their corresponding watermark bits match the original watermark bit. If the ith watermark bit of w_b is the same as that of w, we put the corresponding IPD difference D_i in group D^+. Otherwise, we put D_i in group D^-. We then sort all $D_i \in D^-$ in the ascending order according to their absolute value $|D_i|$[1]. Apparently, we can get a better watermark by choosing different matching packets to make some $D_i \in D^-$ fall into D^+. Among all $D_i \in D^-$, the one with the smallest absolute value is most likely to be changed by using different matching packets.

In the final phase, we focus on those embedding packets of $D_i \in D^-$, whose most desirable matching packets haven't been chosen in the basic subsequence. For these packets, the current choice of their matching packets are not optimized. This process starts from the D with the smallest absolute value. Suppose it is D_j. To adjust D_j, we begin with its last embedding packet, p_k, because adjusting it will not force other embedding packets of D_j to change their selection of matching packets.

1. For packet p_k, if the matching packet currently used is already the most desirable one, we stick to this selection, and continue for the previous embedding packet.

2. Otherwise, we select the next matching packet of p_k, if any. Since it also affects other packets, we have to re-select their matching packets too. Then we compute the new D_j to see if it moves closer to 0. We also make sure such a change will not make any D_i previously in D^+ fall into D^-. If both conditions hold, it means choosing the new matching packet will improve the watermark we can get. So we make this change permanent and update D_j. Since we have recorded all the IPD differences and labeled each packet with a triple, the computation of new D_j can be executed quickly.

3. Repeat step 2 for packet p_k until 1) D_j falls into D^+, 2) there is no more matching packets, or 3) making changes will not necessarily improve the watermark. If D_j falls into D^+, that means we have found a better watermark. We then adjust the next smallest $D \in D^-$. Otherwise, we will repeat this procedure for the previous embedding packet of D_j. Whenever we obtain a watermark whose hamming distance is less than the threshold, we can terminate and report a stepping stone flow is found.

[1]Whether D_i is positive or negative is not related with the group it belongs to.

In the final phase, each time we determine the best option of one embedding packet without considering the possible change of other packets. In other words, we fix the selection of other packets, and check what is the best selection of the current packet. Unlike the Brute Force algorithm, there will be no backtracking. Because we use the heuristics to always adjust those packets that are most likely to generate a better result, usually we should get a watermark very close to the "best" one.

3.3.4 Algorithm 4: Greedy*

In case a user wants to find the actual "best" watermark, and the computation cost is not much a concern, we provide another algorithm *Greedy**. The only difference between this algorithm and the Greedy$^+$ algorithm is in the last phase. In Greedy*, after we determine which embedding packets are non-optimized in the basic subsequence, we enumerate all the possible combinations of these embedding packets to find the "best" watermark. Whenever a watermark that has a hamming distance less than the threshold is obtained, this algorithm will terminate.

The last phase of Greedy* is similar to the Brute Force algorithm. However, after the previous several phases, usually we can reduce the size of searching space significantly. For example, suppose the watermark has 24 bits, the redundancy number r is 4, the threshold is 5, the hamming distance h_g for watermark w_g is 4, and the hamming distance h_b for the watermark w_b is 7. Half of the embedding packets use their first matching packets and will be omitted. So the number of embedding packets we need to check will be at most $(h_b - h_g) \times r \times 2 = 24$. Compared with the number of packets we need to check in the Brute Force algorithm, $24 \times r \times 4 = 384$, this is a huge improvement.

To bound the worst case execution time, this algorithm also allows users to set up a maximum bound of computation cost. If it cannot finish within the specific cost, it returns the best watermark obtained so far.

3.4 Complexity Analysis

Due to the possible large number of chaff packets added and suspicious flows to be correlated, computation complexity is a big concern in our solution. In this section, we investigate the time complexity of different algorithms. Obviously, the complexity of all algorithms are affected by the

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	4
WM threshold	7
WM delay	600ms
S-IV threshold	3 seconds

number of packets in the flow pairs, the number of chaff packets added, and the maximum delay Δ.

Considering an upstream flow f and a possible downstream flow f'. In the packet matching process, each packet in f' will be checked at most twice in the worse case. Thus, the worst case complexity is $O(m)$, where m is the number of packets in f'. If f' is actually in the same connection chain as f, m is equal to the sum of the number of packets n in f and the number of chaff packets c. So the complexity is also $O(n + c)$. The average number of matching packets in each 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 a chaffed flow of f, $\lambda_{f'}$ is equal to the average packet arrival rate of f plus the average arrival rate of chaff packets: $\lambda_{f'} = \lambda_f + \lambda_c$.

Since the Greedy algorithm only needs to check every embedding packet once, its computation complexity is $O(n)$. For the Greedy$^+$ algorithm, the most time consuming part is the last phase where every matching packet for any non-optimized embedding packet will be re-scanned. Suppose the non-optimized embedding packets are $p_{e_1}, \ldots, p_{e_k}$. It requires to check at most $\sum_{i=1}^{k} |M(p_{e_i})|$ packets. Using the average number of packets of matching sets, it is bounded by $O(n \cdot \lambda_{f'} \cdot \Delta)$. The last phase of Greedy* algorithm needs to check $\prod_{i=1}^{k} |M(p_{e_i})|$ packets. However, both Greedy$^+$ and Greedy* algorithm normally have much better performance than their worst case scenarios.

4 Experiments

In this section, we evaluate the performance of our algorithms using *detection rate*, *false positive rate* and *computation cost*. Both real world and synthetic network flows are used in our experiments. We compare our algorithms with the basic watermarking scheme proposed in [9] and scheme S-IV in [13], because they are the best existing active and passive schemes, respectively. Only timing constraint is used in packet matching process. We expect that false positive rate and computation cost decrease dramatically if quantized packet size constraint is also used.

4.1 Real World Data Set

We first evaluate our algorithms using 91 real SSH/Telnet traces derived from Bell Labs-1 Traces of NLANR [5]. All real traces have more than 1,000 packets. For each trace, we first embed a randomly generated watermark[2], then introduce 9 different kinds of timing perturbations, which is uniformly distributed with a maximum delay from 0 (i.e., no perturbation) to 8 seconds. The increment of maximum delay is 1 second. 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, and the increment is 0.5. The maximum delay Δ in our algorithms is set the same as maximum delay of timing perturbation. For each setting of Δ and λ, we compute the detection rate, false positive rate, and computation cost. The threshold of hamming distance in the watermark scheme is 7. Watermark length is 24, redundancy number is 4, and the delay a is 600 milliseconds. The threshold for scheme *S-IV* is set to be 3 seconds. The parameters is shown in table 1. In Greedy* algorithm, we also set the maximum cost to 1,000,000 to bound its execution time. The meaning of computation cost is explained in section 4.1.

Detection Rate. To evaluate the detection rate, for each setting of parameter Δ and λ_c, we calculate the correlation between each original flow and its perturbed and chaffed flows. For a specific Δ and λ_c, there are 91 flow pairs to be correlated. We then compute the average detection rate for every setting of Δ and λ_c.

Figure 3 shows the detection rate changing with λ_c when $\Delta = 7$ seconds for real flows. The detection rate of the basic watermark scheme falls near 0 when chaff packets are added, which shows chaff destroys watermark detection mechanism completely. The Greedy algorithm can always achieve 100% detection rate. For other algorithms, the detection rates increase rapidly with the number of chaff packets. They will have around 100% detection rate when $\lambda_c \geq 2$. If there is no chaff added, Greedy$^+$ and Greedy* can detect more than 40% of stepping stone connections, while scheme S-IV can only detect less than 10%. Although theoretically Greedy* should have better detection rate than Greedy$^+$, under the bound of computation cost, Greedy$^+$ outperforms Greedy*.

Figure 4 shows the detection rate changing with Δ when $\lambda_c = 3$. In this case, all algorithms except the basic watermark approach have very high detection rate. However, S-IV has significant lower detection rate than our algorithms when there is no chaff, and fails to reach 100% for other cases.

Because we are trying to use the "best" watermark, the increase of chaff packets helps the detection rate. However,

[2]No watermark is embedded when the fast solution of scheme S-IV is evaluated.

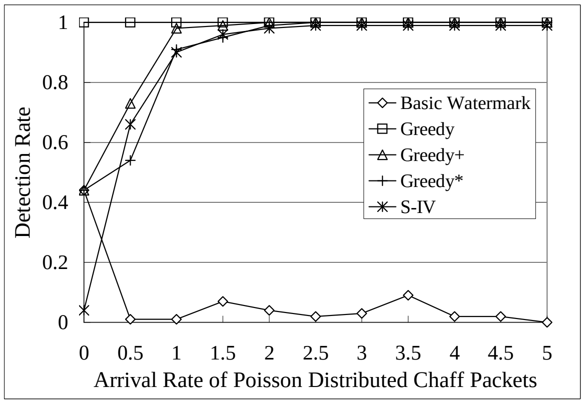

Figure 3. Detection rate changing with λ_c, $\Delta = 7s$

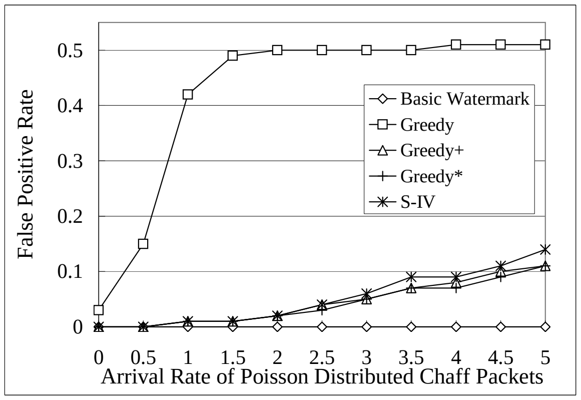

Figure 5. False positive rate changing with λ_c, $\Delta = 7s$

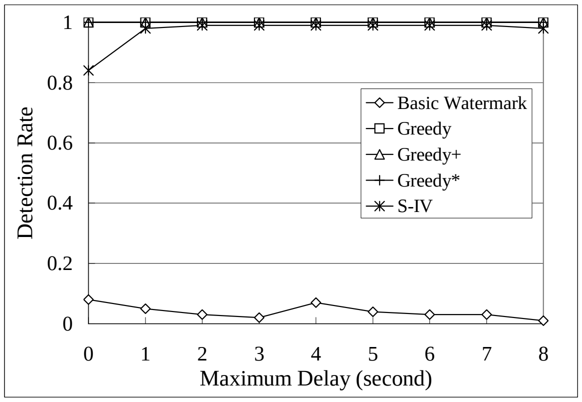

Figure 4. Detection rate changing with Δ, $\lambda_c = 3$

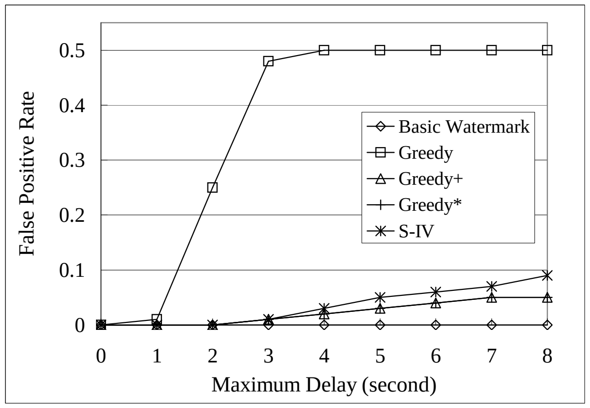

Figure 6. False positive rate changing with Δ, $\lambda_c = 3$

the increase of chaff will also increase the false positive rate, which is shown in the following.

False Positive Rate. To evaluate the false positive rate, for each setting of parameter Δ and λ_c, we calculate the correlation between each original 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 Δ and λ_c. Figure 5 shows the false positive rate changing with λ_c when $\Delta = 7$ seconds. Figure 6 shows the false positive rate changing with Δ when $\lambda_c = 3$.

Unsurprisingly, Greedy algorithm shows the worst performance in terms of false positive rate. Its false positive rate reaches about 50% very quickly after $\lambda_c \geq 2$ for both synthetic and real flows. Except for the basic watermark algorithm, 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. Now let's compare the computation costs of these algorithms. We also include the cost of packet matching process since it is a critical and time consuming step both in our approach and scheme S-IV. To eliminate the bias of different implementation details, we define the *computation cost* as the number of packets in both flows that have to be checked to compute the "best" watermark or the smallest deviation for scheme S-IV. We distinguish the computation costs between correlated and uncorrelated flow pairs. For each algorithm, we compute the average computation cost for each setting of λ_c and Δ.

Figure 7 shows the relation between computation cost and λ_c for correlated flows when $\Delta = 7s$. Figure 8 shows the relation between computation cost and Δ for correlated flows, when $\lambda_c = 3$. Among all algorithms, Greedy algorithm has the smallest computation cost, which is also independent of λ_c and Δ. Greedy* has bumps in its curve especially when there are small number of chaff packets. It is because when more chaff packets are added, matching sets grow bigger. The number of packets that need to be

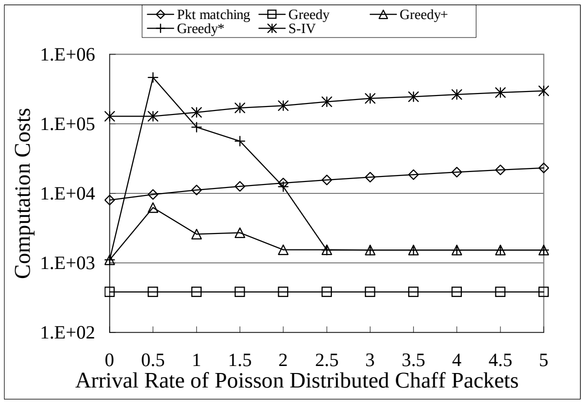

Figure 7. Computation Costs changing with λ_c, $\Delta = 7s$, correlated flow pairs

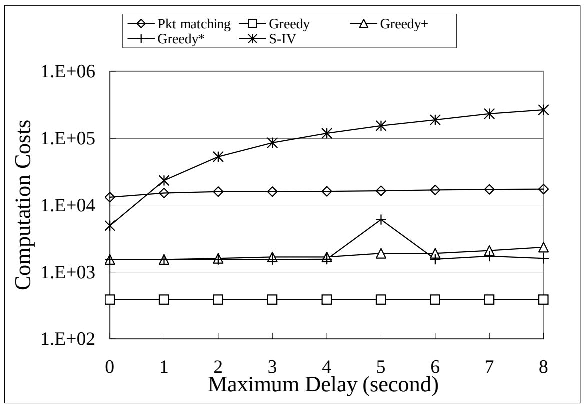

Figure 8. Computation Costs changing with Δ, $\lambda_c = 3$, correlated flow pairs

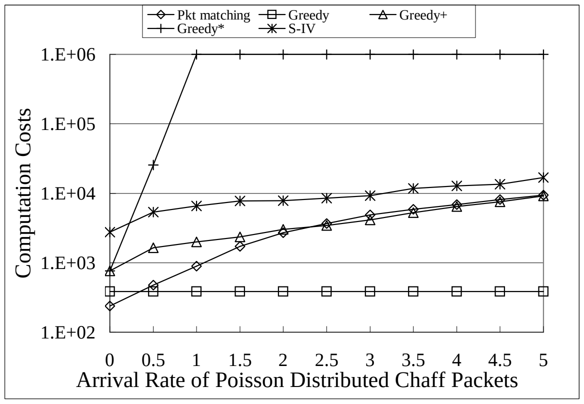

Figure 9. Computation Costs changing with λ_c, $\Delta = 7s$, uncorrelated flow pairs

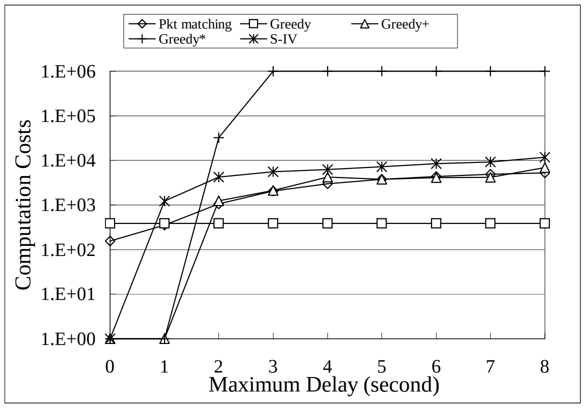

Figure 10. Computation Costs changing with Δ, $\lambda_c = 3$, uncorrelated flow pairs

checked increase rapidly. However, when there are enough chaff packets added, our optimization techniques can give out the results quickly so that the computation cost begin to decrease. Greedy$^+$ also shows smaller bumps in figure 7 for the same reason. There are certain cases that Greedy* cannot finish within its bound of computation cost, which explains why sometimes it has lower detection rate than Greedy$^+$. Both Greedy$^+$ and Greedy* have up to about 40 times lower costs than S-IV.

Figure 9 shows the relation between computation cost and λ_c for uncorrelated flows, when $\Delta = 7s$. Figure 10 shows the relation between computation cost and Δ for uncorrelated flows, when $\lambda_c = 3$. It is worth noticing that some algorithms will have 0 cost[3]. It is because these algorithms depends on packet matching procedure. If the matching process fails to find any matching packet, we immediately have negative correlation result. The computation cost of Greedy* will reach its 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 also evaluated our algorithms using 100 synthetic tcplib traces [3]. Each synthetic trace has 800 packets. We repeat the above experiments using the same watermarking parameters as in section 4.1.

The results for detection rate are shown in figures 11 and 12. They are similar to those for real flows, except that scheme S-IV has a little improvement. The results for false positive rate are shown in figures 13 and 14. Unlike the real flows, the false positive rates increase more quickly when Δ and λ_c is large. It is because the synthetic flows all have the same number of packets, while the lengths of real flows vary substantially. Both Greedy$^+$ and Greedy* still have up to about 5% and 10% lower false positive rate than that of S-IV, as shown in figure 12.

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

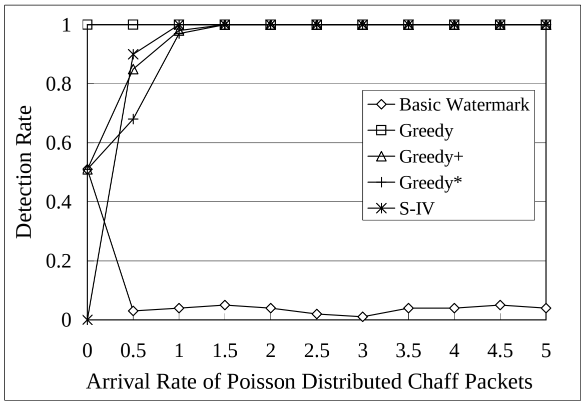

Figure 11. Detection rate changing with λ_c, $\Delta = 7s$

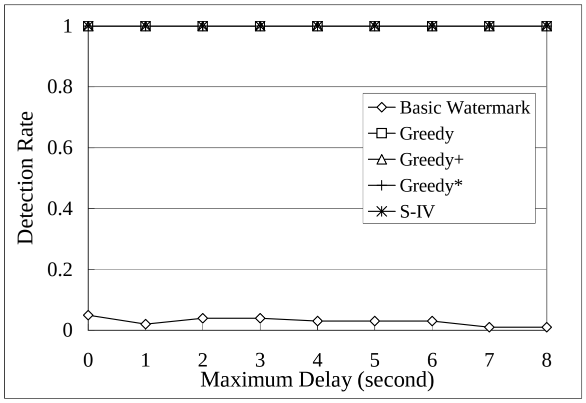

Figure 12. Detection rate changing with Δ, $\lambda_c = 3$

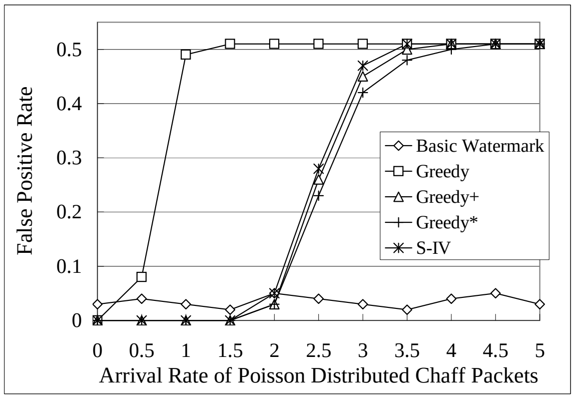

Figure 13. False positive rate changing with λ_c, $\Delta = 7s$

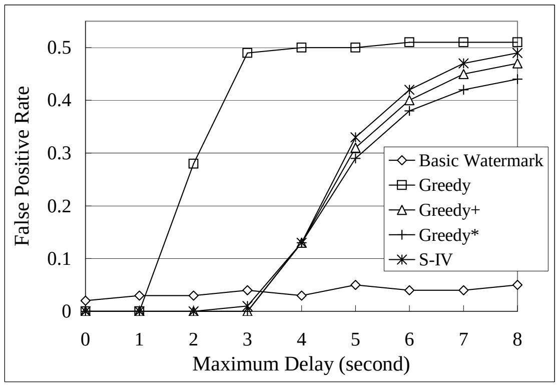

Figure 14. False positive rate changing with Δ, $\lambda_c = 3$

The result of computation costs for synthetic flows are shown in figures 15, 16, 17 and 18. In these figures, we can see similar trends to the results of real flows. Greedy$^+$ has shown much better computation efficiency than both S-IV and Greedy*.

4.3 Overall Performance

From all the experiment results, algorithm Greedy$^+$ has shown overall the best trade-off among detection rate, false positive rate and computation cost. The optimization techniques successfully improve the efficiency while keeping the effectiveness. Greedy* suffers from its high computation cost, especially when it cannot find correlation. Scheme S-IV only shows better detection rate in a small interval of λ_c for synthetic traffic, and has worse false positive rate than both Greedy$^+$ and Greedy*. It also has higher computation cost than Greedy$^+$. The packet matching process shows reasonable computation cost so that the entire correlation procedure can finish in a reasonable time.

5 Related Work

The problem of detecting interactive stepping stones was first formulated by Staniford and Heberlein [7]. They proposed a content-based approach by creating thumbprints from the payload of packets, and comparing them to find extremely good matches. Another content-based scheme was Sleepy Watermark Tracing technique [11], which correlates stepping stone connections by injecting non-displayable contents. Such methods required the contents of packets could not change significantly between different flows, which made them unusable for encrypted traffic such as SSH connections.

More recent schemes focused on timing characteristic of stepping stone connections. Zhang and Paxson [14] 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. The stepping stone connections could be detected by calculating if their OFF periods coincide. Yoda and Etoh [12] pro-

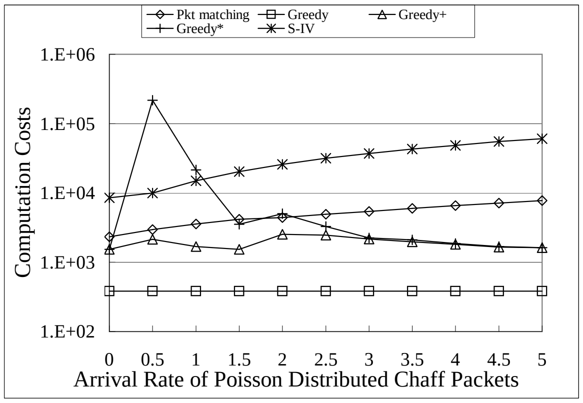

Figure 15. Computation Costs changing with λ_c, $\Delta = 7s$, correlated flow pairs

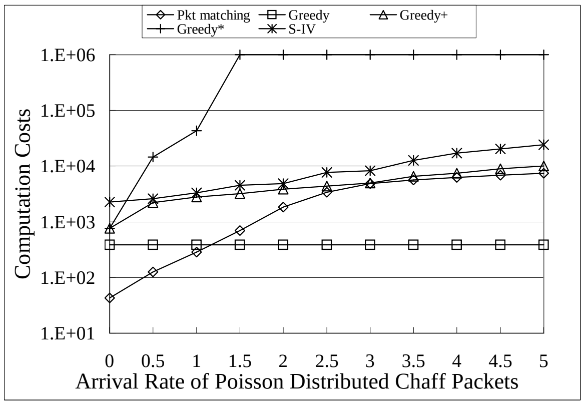

Figure 17. Computation Costs changing with λ_c, $\Delta = 7s$, uncorrelated flow pairs

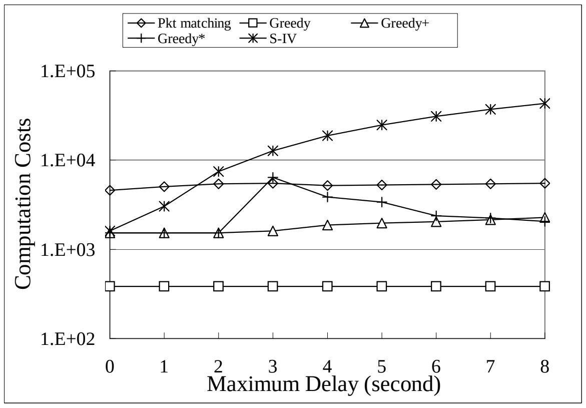

Figure 16. Computation Costs changing with Δ, $\lambda_c = 3$, correlated flow pairs

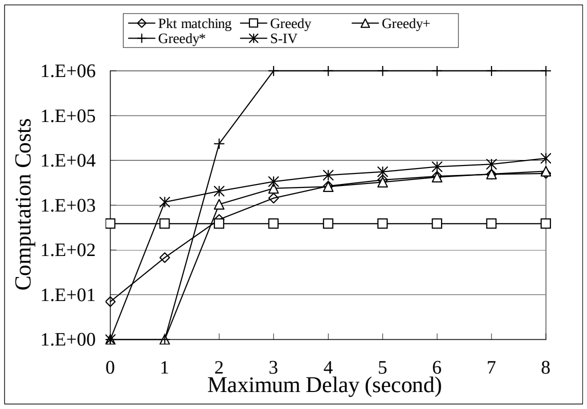

Figure 18. Computation Costs changing with Δ, $\lambda_c = 3$, uncorrelated flow pairs

posed a deviation-based scheme to detect stepping stone connections. This scheme calculated deviation between a known attacking flow and all other flows appeared around the same time. 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. [10] proposed a correlation scheme based on inter-packet delays (IPDs). They showed that timing characteristic of IPDs of corresponding packet pairs were preserved across multiple stepping stones, and could be used for correlation. Although these schemes can be utilized to correlated encrypted connections, they are all vulnerable to timing perturbations intentionally added by attackers.

Donoho et al. [4] investigated the theoretical limits of the attackers' ability to disguise their traffic through timing perturbations and extra chaff packets. Using wavelet and multiscale analysis, they demonstrated that stepping stone correlation is still possible by observing long term behavior. However, they did not discuss the false positive rate of their method, and provided no practical scheme to defeat chaff packets. Wang and Reeves [8] proposed an active watermarking scheme that was robust to random timing perturbations. They first identified the quantitative tradeoffs 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 identify provable bound on the number of packets needed to achieve desired correlation effectiveness. Wang et al. also proposed another watermarking scheme [9] that guaranteed even timing adjustments and had better true positive rates.

Blum et al. [1] proposed to correlate stepping stone connections by counting the packet number differences in certain time intervals. They achieved provable upper bounds on the number of packets needed to guarantee desired false positive rate, under the assumption that packets arrive according to a Poisson distribution. They also provided an algorithm which can deal with small number of chaff packets. However, they did not show any experimental evaluation of their algorithms.

Zhang et al. [13] proposed several algorithms to detect stepping stone connections when timing perturbation or/and chaff packets may be present. Their algorithms were also based on finding possible corresponding packets. They have shown that when deviation was chosen as the correlation criteria, a fast solution could be used to reduce computation time while still maintaining the same correlation results. Unlike active schemes, their passive scheme does not require traffic manipulation, thus is less noticeable to attackers. However, using active watermarking scheme, our algorithms can achieve better performance with less computation costs.

Researchers have also focused on building anonymous systems that can hide the identities of communication participants [2] [6]. Mixes have been used to provide anonymous connections that are resistant to eavesdropping and traffic analysis. A Mix accepts fix-length messages from various sources, performs cryptographic transformations, and forwards them to the next destination in a random order. Random packet delay and traffic padding are also used to improve the privacy.

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, in this paper, we introduce our correlation scheme based on packet matching and active timing-based watermark scheme. 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 work, 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 [13].

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. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, 1981.

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

[4] 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.

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

[6] Onion routing. http://www.onion-router.net.

[7] 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.

[8] 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.

[9] 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 University, 2005.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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