Probabilistic Proof of an Algorithm to Compute TCP Packet Round-Trip Time for Intrusion Detection

Jianhua Yang[1] and Yongzhong Zhang[2]

[1] Department of Computer Science, University of Houston
4800 Calhoun Rd. Houston, TX, 77204 USA
jhyang@cs.uh.edu
[2] Department of Computer Science, Shanghai TV University
288 Guoshun Rd, Shanghai, 200433 China
yzhang@shtvu.edu.cn

Abstract. Most network intruders tend to use stepping-stones to attack or invade other hosts to reduce the risks of being discovered. One typical approach for detecting stepping-stone intrusion is to estimate the number of connections of an interactive session by using the round-trip times (RTTs) of all Send packets. The key of this approach is to match TCP packets, or compute the RTT of each Send packet. Previous methods, which focus on matching each Send packet with its corresponding Echo packet to compute RTTs, have tradeoff between packet matching-rate and matching-accuracy. In this paper, we first propose and prove a clustering algorithm to compute the RTTs of the Send packets of a TCP interactive session, and show that this approach can compute RTTs with both high matching-rate and high matching-accuracy.

Keywords: Network security, intrusion detection, stepping-stone, round-trip time, TCP packet-matching.

1 Introduction

Computer and network security has been becoming more and more important as people depend on the Internet to conduct business, and the number of the Internet attacks has increased greatly [1], [2], [3]. To detect and traceback intruders on the Internet have become more and more difficult than before because most intruders are using some sophisticated technologies and usually launching their attacks indirectly to reduce the risks of being discovered. One prevalent way used by intruders is to take advantage of stepping-stones [4], which are computer hosts compromised by intruders to hide themselves deeply, to launch their attacks. Bunch of techniques have been proposed and developed to detect such kind of attacks, called stepping-stone intrusion.

One representative of the techniques is to estimate the downstream length (in number of connections) of a connection chain from the monitor host where a monitor program resides to the destination host to detect the existence of a stepping-stone intrusion. Yung [5] firstly published the idea to do it in 2002. In that paper, Yung proposed to use the RTT between one Send packet and its corresponding Echo packet to measure the length of a connection chain. The problem is that Yung did not

J. Zhou, M. Yung, and F. Bao (Eds.): ACNS 2006, LNCS 3989, pp. 18 – 32, 2006.
© Springer-Verlag Berlin Heidelberg 2006

propose a way to match each Send and Echo packet exactly. Instead he used statistical method to estimate the RTT of a Send packet, which is not accurate, especially when send-echo pairs are overlapped deeply, which happens often on the Internet. Yang and Huang [6] published an idea to estimate the RTT of a Send packet by matching a TCP Send packet with its corresponding Echo packet; it results in the Conservative and the Greedy algorithms. Yang [6] makes use of TCP Send and Echo packet sequence number and takes advantage of the gap between two consecutive Send packets to match TCP packets. However, even though Yang claimed that the Conservative algorithm can estimate the RTT accurately, but only few packets are matched especially under the scenario that send-echo pairs are overlapped deeply. The Greedy algorithm can cover most of the Send packets, but with some incorrectly matches. Neither of them can obtain both high packet matching-rate and high packet matching-accuracy. The problem is that they always search for a 'candidate' Echo packet locally, rather than globally, when they try to match a Send packet.

In this paper, we propose a clustering algorithm that matches most of Send packets, and computes the RTTs of Send packets more accurate. This algorithm is based on a result that is a cluster with smallest standard deviation has the highest probability to represent the true RTTs, which can be proved by using Chebyshev inequality. The clustering algorithm can get both high packet matching-rate and high packet matching-accuracy in computing packet RTTs because it looks for a 'candidate' Echo packet globally when it tries to match a Send packet. The way used in the Conservative and the Greedy algorithms is that once an Echo packet is captured, we must determine its matched Send packet immediately even though sometimes we could not. Unlike this way, the clustering algorithm takes the approach that once we catch an Echo packet, we do not determine its matched Send packet immediately even though occasionally we are pretty sure the matched Send packet.

The contributions of this paper are the two points. 1) We prove a result that is the cluster, which is generated from the Send and Echo packets of a TCP interactive session, with smallest standard deviation has the highest probability to represent the true RTTs of the Send packets. 2) We propose a clustering algorithm based on the proved result to compute the RTTs by matching each TCP Send and Echo packets globally.

The rest of this paper is arranged as following. In Section 2, we talk about the motivations of proposing the clustering algorithm to compute RTTs. Section 3 presents the clustering algorithm and its probabilistic proof. In section 4, some experimental results and comparisons are presented. Section 5 presents some related work. Finally, in Section 6, the whole work is summarized, and the future work is presented.

2 The Motivation

Detecting a long interactive connection chain is a very important method to detect stepping-stone intrusion because it has no false alarms. The key issue of estimating the length of a connection chain is to match the TCP packets flowing through a connection chain, or to compute the RTTs of TCP Send packets. If each Send packet is followed immediately by one or more Echo packets, such as the sequence $\{s_1, e_1, s_2,$

e_2, e_3, s_3, e_4} in which each element represents the timestamp of the corresponding Send or Echo packet, the gaps e_1-s_1, e_2-s_2, and e_4-s_3 would be the true RTT of each Send packet s_1, s_2, and s_3 respectively. The complexity of matching TCP packets is in the situation that more Send packets are followed by more Echo packets, which is overlap of send-echo pair. For example, if the above case became the sequence {s_1, s_2, e_1, e_2, e_3, s_3, e_4}, there would be several possible packet-matching schemes. 1) Send packet s_1 together with s_2 are matched by e_1, e_2, and e_3; 2) s_1 is matched with e_1, as well as s_2 is matched with e_2, and e_3; 3) s_1 is matched with e_1 and e_2, thus s_2 matches e_3; 4) s_1 matches e_1, e_2, and e_3, therefore, s_2 and s_3 match e_4. If you look at the four schemes, s_1 must match e_1 whatever the matching scheme is. This is just the idea of the Conservative algorithm [4], which has low matching-rate because it ignores to match s_2 in the above case. The Greedy algorithm [4] takes a very rapacious way to match the rest Send packets, which is FIFO. As a result, for the above case, the Greedy algorithm would match s_1 with e_1, in addition, match s_2 with e_2. This is why it is possible that the Greedy algorithm has low matching-accuracy because the matches determined by FIFO policy might not be correct.

However, we are aware of one fact that each Echo packet must correspond to one or more Send packets which timestamps are smaller than that of the Echo packet. When we capture an Echo packet, even though we are not sure its matched Send packet, but we do know there is at least one Send packet matched with it. We simply assume that every Send packet is supposed to match the Echo packet, and compute each gap between each Send packet and the Echo packet. For each Echo packet, we have one gap set in which one of the gaps must be the true RTT of the Echo packet. The problem is that we are not sure which gap is the right one. The interesting thing is if we observe more such gap sets, we found that for most of the gap sets, each gap set has one element that is very close to the ones in other gap sets. The only sound explanation is those tight elements are the true RTTs of the Send packets unless this is a coincidence. The more gap sets we observe, the lower probability that it is a coincidence. After we explore the distribution of true RTTs, we believe the probability of coincidence is extremely small. The feature of the distribution of the true RTTs motivates us a way to extract the RTTs from the gap sets observed. This way is the algorithm to be discussed in Section 3.

The RTT of a Send packet is the sum of processing delay, queuing delay, transmission delay and propagation delay [10] for the packet in a connection chain on the Internet. Further research pointed out that a RTT is mainly determined by the propagation delay and the queuing delay [10]. The propagation delay determines mainly its constant part, and the queue delay determines mainly its varying part, which can be simulated by an exponential distribution. In other words, the variation of the RTTs can be modeled as an exponential distribution, which indicates that most of the true RTTs are scattered in a very small range. The true RTTs can be different because the Internet traffic always fluctuates but they vary slightly. If we use standard deviation to measure the variation degree of RTTs, it should be small. If we combine the elements in the gap sets to form clusters, the cluster with smallest standard deviation should have the highest probability to represent the true RTTs. If we could prove this point, the way to pull out the true RTTs from the gap sets would become to find the cluster with smallest standard deviation.

Table 1 shows the comparison of standard deviation over different clusters in a real world example that can give us some practical sense on the above analysis. In this example, a connection chain, which contains six connections, is established by using OpenSSH. We monitor the connection chain at the start of the session for a period of time, and capture all the Send and Echo packets. First, for each Echo packet, we form one gap set; second, we form all the clusters by combining the elements in all the gap sets (for details to form the clusters, see Section 3). We compute the standard deviation (with unit microsecond) of each cluster and show only part of the results in Table 1. It is apparently that the standard deviation of the RTTs (one of the clusters) is much smaller than that of any other cluster.

Table 1. Comparison of standard deviations of time gap clusters

RTTs	cluster1	cluster2	cluster3	cluster4	cluster5
2.8E3	4.4E7	3.3E6	2.7E5	1.7E7	8.5E6

There are two problems needed to mention. One is to process resend packet. Another is to process the Send packets without reply. Resend packets are easy to handle because they have the same sequence (Seq) and acknowledgement (Ack) number. We do not record the Send packet if it has the same Seq and Ack number as its previous packet. We know that is not every Send packet is replied by the victim site (or the host at the end of a session). There are still few Send packets only acknowledged by the downstream neighbor host or not replied at all, such as ignore packet, keep-alive packet, and key re-exchange packet [7], [8], [9]. These packets are not intended for the target machine, so we cannot capture their Echo packets. The question is if they affect the result of packet-matching or computing RTTs. First, it does not affect our clustering algorithm much because the amount of these packets is very small comparing to the whole Send packets. Second, if a Send packet is not echoed by the final destination host, it does not matter due to the two reasons. 1) Its gap is not involved into the cluster that represents the RTTs because this gap is probably either smaller or larger than a regular RTT. 2) Even though we assume that the gap between this Send packet and the other Echo packet is close to the true RTTs and involved into the RTT cluster accidentally, but the only effect is we have one more packet-matching. It does not affect the estimation of RTTs. To simplify our analysis, we assume every Send packet is replied by the final destination host.

3 Clustering Algorithm and Its Proof

Given two sequences $S=\{s_1, s_2, \ldots, s_n\}$ and $E=\{e_1, e_2, \ldots, e_m\}$, where s_i is a Send packet, as well as its timestamp, and so is e_j. We assume that the packets in these two sequences are captured from the monitor host in a connection chain at the same period of time. We can use S and E to generate different data sets, which are actually aggregations of gaps between each Send packet in S and each Echo packet in E. There are two ways to create the data sets: one is to compute the gaps based on each Echo packet in E, while another is based on each Send packet in S. Obviously the data sets created by the two ways are fundamentally equivalent.

If we create each data set based on each Send packet in S, we have the following n data sets in which the negative elements are not taken into consideration:

$S_1=\{s_1e_1, s_1e_2,\ldots, s_1e_m\}$,
$S_2=\{s_2e_1, s_2e_2,\ldots, s_2e_m\}$,
…
$S_n=\{s_ne_1, s_ne_2,\ldots, s_ne_m\}$.

Here, S_i represents i^{th} data set based on the Send packet s_i; $s_ie_j=e_j-s_i$ represents the time gap between the timestamp of ith Send packet and the jth Echo packet. There is one and only one gap which represents the true RTT in each data set because we have assumed that each Send must be replied by the victim site (final destination host).

If we create the data sets based on each Echo packet in E, we have the following m data sets in which the negative elements are also not taken into consideration:

$E_1=\{s_1e_1, s_2e_1,\ldots, s_ne_1\}$,
$E_2=\{s_1e_2, s_2e_2,\ldots, s_ne_2\}$,
...
$E_m=\{s_1e_m, s_2e_m,\ldots, s_ne_m\}$.

Similarly, E_j represents the jth data set based on the Echo packet e_j in E. We are not sure if we have and only have one gap to represent the RTT in each data set E_i. The reason is that one Send is possibly replied by one or more Echo packets. We need to define which one represents the true RTT of the Send exactly. Under this situation, we define the gap between the Send and the first Echo to represent the true RTT. A similar situation is that more Send packets are perhaps responded by only one Echo, under which we define the gap between the last Send and the Echo to represent the true RTT. Anyway, we prefer to define the smallest gap to represent the true RTT.

For convenience, we first consider the data sets based on each Send in S. We already knew that each data set must contain one and only one true RTT, but we are not sure which one in a data set is the right one. We simply assume that each gap in each data set S_i has the same probability to represent the RTT. We make a combination by picking up one element from each data set and call each combination a cluster, so we have m^n clusters altogether. The true RTTs must be one of the m^n clusters because all the possibilities of combination are enumerated. We can prove that the cluster with the smallest standard deviation has the highest probability to represent the true RTTs. The following clustering algorithm to compute the true RTTs of TCP Send packets is rooted in this statement.

3.1 A Clustering Algorithm

We monitor an interactive TCP session established by using OpenSSH for a period of time, capture all the Send and Echo packets, and put them in two sequences S with n packets and E with m packets, respectively. The following clustering algorithm with inputs S and E can compute the true RTTs for all the Send packets in S.

A Clustering Algorithm (S, E):
Begin

1. Create data sets S_i, $1\leqslant i\leqslant n$, and $S_i=\{t(i,j) \mid t(i,j)=t(e_j)-t(s_i), 1\leqslant j\leqslant m\}$;
2. Generate clusters C_k ($1\leqslant k\leqslant m^n$) from data sets S_i ($1\leqslant i\leqslant n$), and $C_k=\{t(i, j_i) \in S_i \mid \forall 1\leqslant i\leqslant n \;\&\; j_i\in[1, m] \;\&\; j_1\leqslant j_2\leqslant\cdots\leqslant j_n\}$;

3. Filter out each cluster C. For any cluster C_k: (a) if t(i, u), t(i, v) $\in C_k$ & u<v, then delete t(i, u), and (b) if t(u, j), t(v, j) $\in C_k$ & u<v, then delete t(v, j)
4. Compute the standard deviation σ of each cluster C;
5. Output the cluster C_u to represent the true RTTs of the Send packets in S, and $C_u=C_q \mid \sigma_q \leqslant \sigma_v$ for all $1 \leqslant v \leqslant m^n$.

End

Here we use t(i,j) to represent the time gap between i[th] Send packet and j[th] Echo packet, $t(e_j)$, and $t(s_i)$ to represent the timestamp of jth Echo and i[th] Send packet, respectively.

In Step 1, we create *n* data sets, one of which has at most *m* elements because the negative elements are not considered. In Step 2, we take one element from each data set and combine them into one cluster, thus at most form m^n clusters because each data set has at most *m* elements. For any two clusters C_u and C_v, they must have at least one element different. The condition $j_1 \leqslant j_2 \leqslant \cdots \leqslant j_n$ can compress largely the space of the clusters. This condition is reasonable because once a Send packet, such as s_i, is assumed to match an Echo packet, such as e_j, it is impossible that any Send packet after s_i will match an Echo packet before e_j. In Step 3, we focus on handling the case that is either more Send packets are responded by one Echo packet or one Send packet is responded by more Echo packets. In Step 5, we select the cluster with the smallest standard deviation to represent the true RTTs of the Send packets in S. Step 5 is guaranteed by the following *Theorem 1.*

Theorem 1*. If given two sequences S (n Sends) and E (m Echoes) from the same session at same period of time, and generate clusters C_1, C_2, …, C_k from S and E according to the clustering algorithm, then the cluster with the smallest standard deviation has the highest probability to represent the true RTTs of the packets in S.*

Proof

Given any cluster C of clusters C_1, C_2,…, C_k with distribution Z which has standard deviation σ_1 and mean μ_1. We assume that the Echo packets inter-arrival distribution is Y with mean μ_2, standard deviation σ_2, and the smallest inter-arrival is L. We first compute the probability of selecting an incorrect gap to represent the true RTT.

Suppose c_i, which is any element in cluster C, is selected from $S_i=\{s_ie_1, s_ie_2, \ldots, s_ie_{k-1}, s_ie_k, s_ie_{k+1}, \ldots, s_ie_m\}$, we assume the correct selection should be s_ie_k, but other element in S_i is selected. To satisfy the condition that C has the smallest standard deviation, the element in S_i selected incorrectly must be closer to μ_1 than s_ie_k. The reason is that for any distribution, if we add one more element, the closer to the mean the one is, the smaller the standard deviation. Only one of the two elements s_ie_{k-1}, s_ie_{k+1} has the highest probability to be selected incorrectly because the elements in S_i are in ascending order. Here, we assume s_ie_{k+1} is closer to μ_1 than s_ie_{k-1}, so we have the inequality (1) which indicates that s_ie_{k+1} is selected incorrectly to represent the true RTT,

$$\left|s_ie_{k+1} - \mu_1\right| < \left|s_ie_k - \mu_1\right| \tag{1}$$

We have assumed that L is the smallest interval in distribution Y, so we have

$$t(e_{k+1}) - t(e_k) \geq L = 2q\sigma_1 \tag{2}$$

Here q is a real number. From inequality (2), for any Send packet s_i, we have

$$t(e_{k+1}) - t(s_i) - (t(e_k) - t(s_i)) \geq 2q\sigma_1$$
$$s_ie_{k+1} - s_ie_k \geq 2q\sigma_1$$
$$s_ie_{k+1} - \mu_1 + \mu_1 - s_ie_k \geq 2q\sigma_1$$
$$|s_ie_{k+1} - \mu_1| + |s_ie_k - \mu_1| \geq 2q\sigma_1 \quad (3)$$

From inequality (1) and (3), we derive

$$|s_ie_k - \mu_1| \geq q\sigma_1$$

The probability that c_i is selected incorrectly can be estimated by using Chebyshev inequality [11], [12],

p(c_i is selected incorrectly)=p(s_ie_{k+1} is selected)

$$= p(|s_ie_{k+1} - \mu_1| < |s_ie_k - \mu_1|)$$
$$= p(|s_ie_k - \mu_1| > q\sigma_1) < \frac{1}{q^2}$$

In other words, the probability to make a correct selection of a Send packet's RTT can be estimated by the following inequality,

p(c_i) = p(c_i is selected correctly)

= 1-p(c_i is selected incorrectly)

$$\geq 1 - \frac{1}{q^2} \quad (4)$$

Given any two clusters C_i and C_j with standard deviation σ_i and σ_j respectively, we know that:

$$\sigma_i < \sigma_j \quad (5)$$

and

$$q_i\sigma_i = q_j\sigma_j = L \quad (6)$$

Here, q_i and q_j are two real numbers. From Step 2 of the clustering algorithm, we know that C_i, and C_j have n elements respectively,

$C_i = \{c_{i1}, c_{i2}, \ldots, c_{in}\}$
$C_j = \{c_{j1}, c_{j2}, \ldots, c_{jn}\}$

Each Send packet is independent from the others, and from inequality (4) we have

p(C_i is the RTTs)=p(c_{i1} is the RTT of s_1, c_{i2} is the RTT of s_2, …c_{in} is the RTT of s_n)

=p(c_{i1})*p(c_{i2})….*p(c_{in})

$$\geq \left(1 - \frac{1}{q_i^2}\right)^n$$

p(C_j is the RTTs)=p(c_{j1} is the RTT of s_1, c_{j2} is the RTT of s_2, …c_{jn} is the RTT of s_n)

=p(c_{j1})*p(c_{j2})….*p(c_{jn})

$$\geq \left(1 - \frac{1}{q_j^2}\right)^n$$

From (5), (6) we know that

$$\left(1-\frac{1}{q_i^2}\right)^n \geq \left(1-\frac{1}{q_j^2}\right)^n$$

This indicates that each cluster C has a probability to represent the true RTTs, but the one with the smallest standard deviation has the highest probability to represent the true RTTs. Therefore, we select the cluster with smallest standard deviation among all the clusters created to represent the true RTTs of the Send packets in S.
End Proof.

We prove that the cluster with smallest standard deviation has the highest probability to represent the true RTTs. Even though the algorithm can give us the best answer when we find the RTTs of Send packets in S, but it is not efficient because of the time complexity which is $O(m^n)$ in worst case. This algorithm cannot be used in real time. In the following section, we propose an efficient clustering algorithm that can be used in real time. Unfortunately, we cannot prove if we can get the best answer with the efficient algorithm, but we can justify it by comparing its result with that of the clustering algorithm in the same context in Section 4.

3.2 The Efficient Clustering Algorithm

The inefficiency of the above clustering algorithm is that the cluster space complexity is $O(m^n)$. Our goal is to shrink the space without losing useful information to make the clustering algorithm efficient. Here we still suppose that we monitor an interactive TCP session for a period of time, and capture n Send packets and m Echo packets, as well as assuming that all the n Send packets are echoed and only echoed by the m Echo packets. We form data sets S_1, S_2, …, S_n upon the n Send packets and m Echo packets. The reason that we have huge combination space in the clustering algorithm is that we combine the elements in data sets S_1, S_2, …, S_n freely, without any restrictions, and enumerate all the possibilities. Some combinations that are apparently impossible to represent the true RTTs are still involved into the final cluster space.

We take some measures in the efficient clustering algorithm to reduce the size of the final cluster space. We have n data sets, and know that each element of the true RTTs is hidden in different one of the data sets as well. For all the m elements in S_1, we simply assume that each one is possible to represent the true RTT of the Send packet s_1, even though we know that actually only one element in S_1 is qualified to represent the true RTT of s_1. We take any element in S_1, such as the ith element s_1e_i, to be the first element of cluster C_i, and look at all the elements in S_2 to find the one that makes C_i more possible to represent the true RTTs and add it to C_i. Similarly, we check all the elements in S_3, S_4, …, S_n respectively, and find one suitable element in each data set and add them to C_i respectively, which finally has n elements. We eventually have m clusters because S_1 has m elements each of which can be used to form one cluster. From *Theorem 1*, it is obvious to take the cluster that has the smallest standard deviation among the m clusters to represent the true RTTs.

However, we still have two problems here. First, when we check a data set to find one element to make the current cluster to be more possible to represent the true RTTs, we have the question that is how to make the current cluster to be more possible to represent the true RTTs. Second, from the whole process to generate the *m* clusters, we cannot guarantee that the cluster that represents the true RTTs is involved in the final *m* clusters. For the first problem, the way we take is to select an element that makes the current cluster have the smallest standard deviation. Upon each element in data set S_1, we could have m^{n-1} combinations in the worst case. The cluster generated by ensuring the smallest standard deviation every time to select one element from a data set could not guarantee the smallest standard deviation among its whole m^{n-1} combinations. This is why the second problem is. To be more understandable, we explain the second problem in details through an example.

Suppose we have four data sets S_1= {20, X}, S_2= {15, 18}, S_3= {18, 19}, S_4= {7, 8}, it does not matter whatever the second element in S_1 is because we only check the clusters formed upon the first element of S_1. If we traverse all the possibilities upon the first element in S_1, we have eight clusters, which are C_1={20,15,18,7}, C_2={20,15,18,8}, C_3={20,15,19,7}, C_4={20,15,19,8}, C_5={20,18,18,7}, C_6= {20,18,18,8}, C_7={20,18,19,7}, C_8={20,18,19,8} with standard deviation 5.71, 5.25, 5.91, 5.45, 5.91, 5.42, 6.06, 5.56, respectively. From the *Theorem 1*, the good answer should be C_2={20,15,18,8}. However, from the efficient algorithm, cluster C only has one element at first, that is C= {20}. Then we check the elements in S_2, we find that the second element is our best choice because it makes C have the smallest standard deviation, so C= {20, 18}. For similar reason, we check the elements in S_3, and S_4 respectively, finally find the cluster C should be {20, 18, 19, 8} which is different from the one obtained from the first algorithm. This is the second problem that cannot be solved in theory so far.

Fortunately, even though there is possibility theoretically that the efficient clustering algorithm could get an incorrect answer, but that possibility is very low when we apply this algorithm to a real world example. We have justified hundreds of real world examples, the above case happened in a very small chance. In Section 4, we give some real world experimental examples to justify the efficient clustering algorithm. Here we give the detailed efficient clustering algorithm.

The Efficient Clustering Algorithm (S, E)
Begin

```
1.  Do i=1, n
            Si=ϕ;
            Do j=1, m
                    t(i, j) = t(ej)-t(si);
                    Si=Si ∪ t(i, j);
            End Do
      End Do
2.  For each t(1, i)∈S1, form cluster Ci:
    Do k=2, n
         σ = stdev(Ci ∪ t(k,1));
         ts = t(k,1);
         Do u=1, m
```

```
                    If stdev(Ci∪ t(k,u)) ≤σ
                              σ = stdev(Ci∪ t(k,1));
                              ts = t(k,u);
                    EndIf
          End Do
          Ci= Ci∪ t(k,1)
     End Do
3.   Filter out each cluster C. For any cluster Ck (1≤k≤n) : (a) if t(i, u), t(i, v) ∈ Ck,
     u<v, then delete t(i, u), and (b) if t(u, j), t(v, j) ∈ Ck, u<v, then delete t(v, j)
4.   σ = stdev(C1);
        Cs = C1;
     Do k=1, n
            If stdev(Ck) ≤ σ
                  σ =stdev(Ck);
                  Cs=Ck;
            End If
        End Do
5.   Output cluster Cs as the RTTs of the Send packets in S.
End
```

In Step 1, at first, we suppose that each data set is empty which is denoted ϕ. We use sign '$\cup$' to express adding one more element to a data set. In Step 2, we use σ to denote the standard deviation of the cluster, and t_s to denote the element in each data set that makes the current cluster get the smallest standard deviation. We use 'stdev' as a function to compute the standard deviation of a given cluster. In Step 3, C_s stands for a cluster which has the smallest standard deviation among all the clusters considered.

Let us analyze the complexity of this algorithm. Suppose we have *n* Send packets, and *m* Echo packets, from the efficient clustering algorithm, we need to select one cluster from the *m* clusters. The complexity of this algorithm is dominated by Step 2. Considering there are *n* elements in each cluster, and there are *m* elements in each data set, the complexity of this algorithm is $O(m*n*(m-1))=O(n*m^2)$ under the worst case. Comparing with the complexity of the previous algorithm, $O(m^n)$, obviously, this algorithm is largely improved in time and space complexity.

4 Empirical Study

We have proposed and proved a clustering algorithm to compute the true RTTs of the Send packets of a TCP session, as well as an efficient one. *Theorem 1* only means among all the clusters created, the cluster with smallest standard deviation has the highest probability to represent the true RTTs of the Send packets. Therefore, from *Theorem 1*, we get the possibility in what extent that the result of the clustering algorithm could stand for the true RTTs. We are not able to prove the result of the clustering algorithm can be the true RTTs definitely. Hence, we design three experiments to evaluate the performance of the above two algorithms and give readers more practical sense. The first experiment is used to justify the correctness of the

results of the clustering by comparing them with the known correct RTTs. The second experiment is designed to evaluate the performance of the clustering algorithm by comparing it with the best packet-matching algorithm. The third experiment is used to evaluate the performance of the efficient algorithm by comparing it with the clustering algorithm.

We made a program by using Libpcap [13], [14] to capture the Send and Echo packets of an interactive TCP session on the Internet. We set up a connection chain that spanned U.S. and Mexico and was long enough so as to generate the overlap of send-echo pair which makes matching packets harder. The connection chain used in our experiment is: Host 1→ Acl08 → Mex→ Themis → Mex → Bayou, in which Host 1, Acl08, Themis and Bayou are hosts located in Houston, and Mex is a host located in Mexico which we have a legal user to access. Acl08 is our monitor host on which one program was running to capture the Send and Echo packets of a TCP session. The sign '→' represents a connection established by using OpenSSH. We did each experiment hundreds of times but here with only one of the results presented.

4.1 Justifying the Correctness of the Clustering Algorithm

In this experiment, we examine the correctness of the clustering algorithm by comparing its results with the known correct RTTs. The problem is how to obtain the correct RTTs of the Send packets of a TCP session for a real world case. The reason to bother packet-matching is the overlap of send-echo pair as we have discussed.

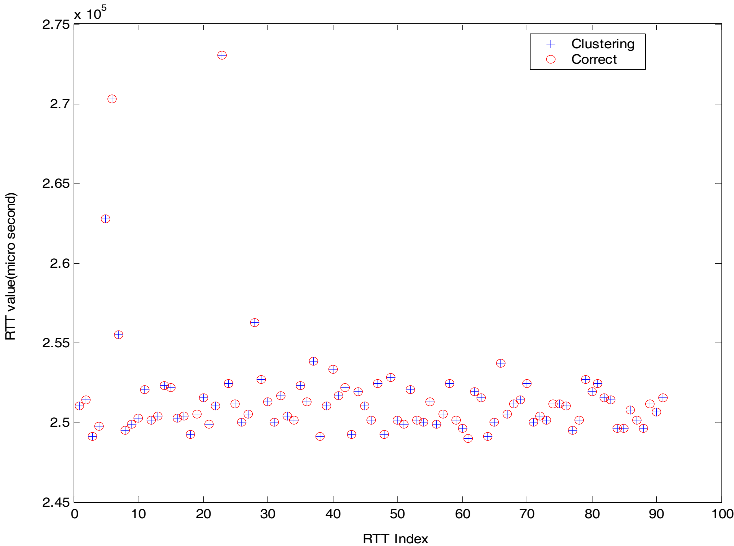

Fig. 1. Justify the correctness of the result of the clustering algorithm

We would avoid such overlap by controlling the keystroke speed to make matching TCP packets easier, and thus obtain the correct RTTs. We did not type each character until we are sure that the previous one (a Send packet) had been replied. It is trivial to

compute the correct RTT for each Send packet by simply matching it with the closest Echo packet following it. Meanwhile, we collect all the Send and Echo packets, and apply the clustering algorithm to them to output the results for the Send packets. Then we can compare these results with the correct RTTs and be sure if they are consistent. The comparison result is showed in Fig. 1, in which X axis represents the Send packet index number, and Y axis represents the RTT value with unit microsecond. For clarity, Fig. 1 only shows us part of the results. It shows obviously that the result of the clustering algorithm is the same as the correct RTTs.

4.2 Comparison Between the Clustering and the Best Packet-Matching Algorithm

In the experiment of Section 4.1, we have justified the correctness of the clustering algorithm, but need to control the keystroke speed to get the correct RTTs. In the real world, it is impossible to do so because intruders control the keystroke speed of an interactive session. Here, we use different way to evaluate the performance of the clustering algorithm under the context very close to the real world. The way is to evaluate the performance of the clustering algorithm by comparing it with the best packet-matching algorithm, the Conservative algorithm [6], which claims to match TCP packet correctly, or computing RTTs correctly, but with few packets matched.

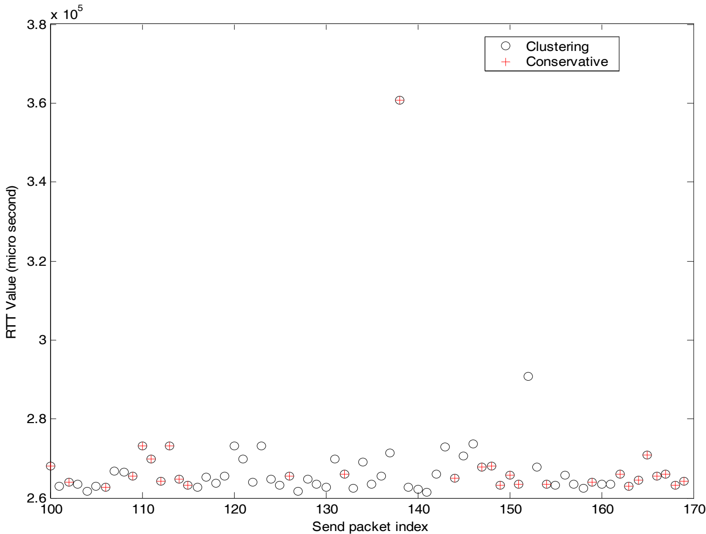

Fig. 2. Comparison between the Conservative and the clustering algorithm

We monitor the TCP connection chain at Acl08 by running the Conservative algorithm to compute the RTTs, while we capture all the Send and Echo packets, and apply the clustering algorithm to them to compute the RTTs. In this experiment, we captured 232 Send packets, but the Conservative algorithm only gave 107 send-echo matches, while the clustering algorithm can obtain 232 RTTs that are equal to 232

send-echo matches. For clarity, we only show part of the RTTs (Send packet index number 100-170) in Fig. 2. From this comparison, we draw the following two points. 1) All the RTTs obtained by the Conservative algorithm are the same as the part of the RTTs found by the clustering algorithm. 2) Even though we cannot judge the correctness of the rest of the RTTs, but from their distribution, we see they are very close to the results of the Conservative algorithm.

4.3 Justifying the Efficient Clustering Algorithm

The efficient clustering algorithm cannot guarantee to compute the best RTTs theoretically just as we have analyzed in Section 3.2. However, this algorithm is still useful to compute the RTTs in practice because the empirical study showed that the efficient clustering algorithm could obtain the RTTs, which are the same as the results of the clustering algorithm for most real world examples. Even though in a very few cases, they are different, but the RTTs from the efficient clustering algorithm are still useful in detecting stepping–stone intrusion because it does not affect to identify one level of a connection chain.

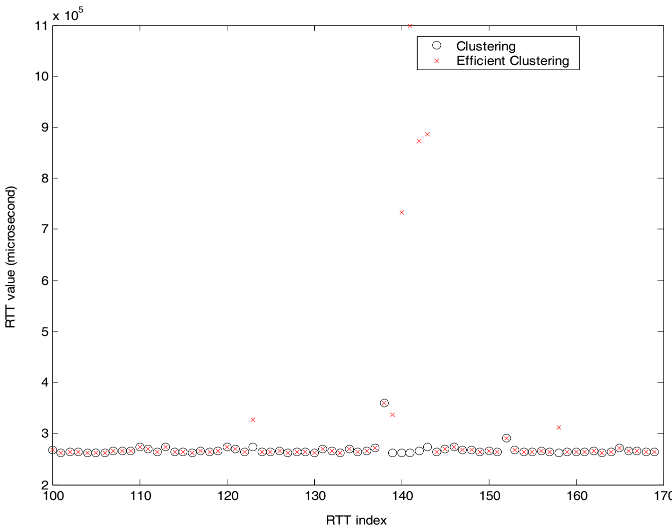

Fig. 3. Verifying the efficient clustering algorithm

In this experiment, we also monitored the connection chain at host Acl08, and collected all the Send and Echo packets in a period of time. We applied the two algorithms to these packets respectively, and compared their results. We did this experiment hundreds of times, and found their results are the same in more than 99% cases. Fig. 3 shows one exception in which their results are different slightly.

In Fig. 3, the circles represent the part of the RTTs found by the clustering algorithm, while the crosses stand for the part of the results obtained from the efficient clustering algorithm. It is obvious that the most of the RTTs computed by the

two algorithms respectively are identical. Only a few RTTs computed by the efficient clustering algorithm are different from the ones obtained from the clustering algorithm. Compare to number of the whole RTTs computed, the number of these RTTs is relatively small (in this experiment, it is 8 out of 232), so we can still figure out one level of a connection chain upon the RTTs found by the efficient algorithm, and thus make this algorithm useful in detecting stepping-stone intrusion.

5 Related Work

The related work can be classified into two categories: a) detecting stepping-stone; b) detecting stepping-stone intrusion. Category a) includes the approaches proposed in papers [4], [15], [16], [17], [18], [19]; Category b) includes the algorithms proposed in papers [5], [6]. The difference between the two categories is that the approaches in category a) can only predict if a host is used as a stepping-stone, the approaches in category b) can predict not only a host is used as a stepping-stone, but also if the host is used by an intruder. Being used as a stepping-stone does not mean being used by an intruder because some legal users also need a host to be used as a stepping-stone. In this paper, we propose an algorithm to compute the RTTs, which eventually can be used to detect stepping-stone intrusion. It is not necessary to compare this algorithm with the approaches in category a).

To determine if a host is used as a stepping-stone is easier than to determine if a host is used by an intruder. Most approaches in category a) are to compare an incoming connection with an outgoing connection to determine if a host is used as a stepping-stone, such as content-based method [19], time-based method [4], [18], packet-number-based method [16]. They all suffer from not only a problem of being vulnerable to intruders' evasion except the method in paper [6], but also high false alarm rate in detecting stepping-stone intrusion. The detecting method based on the RTTs from the clustering algorithm can detect intruders' evasion. We discuss this point in another paper. It is obvious that the approaches in category b) have no false alarm problem in detecting stepping-stone intrusion.

6 Conclusions and Future Work

In this paper, we have proved a theory and proposed a clustering algorithm the theory to compute the RTTs of the Send packets of a TCP interactive session, which are useful in detecting stepping-stone intrusion. We also proposed an efficient clustering algorithm to compute the RTTs with less computation cost, which is possible to be used in real-time detection. The empirical study showed: 1) the RTTs found by the clustering algorithm are the same as the correct ones; 2) the clustering algorithm can match packets with both high matching-rate and high matching-accuracy; 3) the efficient clustering algorithm can compute the RTTs almost the same as the results from the clustering algorithm in most cases.

The clustering algorithm can only figure out estimation of single level RTTs. Even though it is useful in detecting stepping-stone intrusion, but computing multi-level RTTs is more useful and challenging to detect intrusion. One future work is to improve the current clustering algorithm to compute multi-level RTTs.

References

1. R. Base: A New Look at Perpetrators of Computer Crime. Proceeding of 16th Department of Energy Computer Security Group Conference, Denver, CO, May (1994).
2. Richard Power: Current and Future Danger. Computer Security Institute, San Francisco, CA, (1995).
3. CERT: CERT/CC Statistics 1988-2005.,http://www.cert.org, Accessed July (2005)
4. Yin Zhang, Vern Paxson: Detecting Stepping-Stones. Proceedings of the 9th USENIX Security Symposium, Denver, CO, August (2000) 67-81.
5. Kwong H. Yung: Detecting Long Connecting Chains of Interactive Terminal Sessions. RAID 2002, Springer Press, Zurich, Switzerland, October (2002) 1-16.
6. Jianhua Yang, Shou-Hsuan Stephen Huang: Matching TCP Packets and Its Application to the Detection of Long Connection Chains, Proceedings (IEEE) of 19th International Conference on Advanced Information Networking and Applications (AINA'05), Taipei, Taiwan, China, March (2005) 1005-1010.
7. University of Southern California: Transmission Control Protocol. RFC 793, September (1981).
8. Ylonen, T.: SSH Protocol Architecture. draft IETF document, http://www.ietf.org/internet-drafts/draft-ietf-secsh-architecture-16.txt, June (2004).
9. Ylonen, T. : SSH Transport Layer Protocol. draft IETF document, http://www.ietf.org/ internet-drafts/ draft-ietf-secsh-transport-18.txt, June (2004).
10. Qiong Li, David L. Mills: On the Long-range Dependence of Packet Round-trip Delays in Internet. Proceedings of International Conference on Communications (ICC'98), Atlanta, USA, June (1998) 1185-1192.
11. E. Kao: An Introduction to Stochastic Processes. Duxbury Press, New York (1996) 47-87.
12. W. Feller: An Introduction to Probability Theory and Its Applications. Volume I, John Wiley & Sons, Inc., New York (1968) 212-237.
13. Lawrence Berkeley National Laboratory (LBNL): The Packet Capture library. ftp://ftp.ee.lbl.gov/libpcap.tar.z, accessed March (2004).
14. Data Nerds Web Site: Winpcap and Windump. http://www.datanerds.net, accessed July (2004).
15. D. L. Donoho (ed.): Detecting Pairs of Jittered Interactive Streams by Exploiting Maximum Tolerable Delay. Proceedings of International Symposium on Recent Advances in Intrusion Detection, Zurich, Switzerland, September (2002) 45-59.
16. A. Blum, D. Song, And S. Venkataraman: Detection of Interactive Stepping-Stones: Algorithms and Confidence Bounds. Proceedings of International Symposium on Recent Advance in Intrusion Detection (RAID), Sophia Antipolis, France, September (2004) 20-35.
17. X. Wang, D.S. Reeves: Robust Correlation of Encrypted Attack Traffic Through Stepping-Stones by Manipulation of Interpacket Delays. Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS 2003), Washington DC, Oct (2003).
18. K. Yoda, H. Etoh: Finding Connection Chain for Tracing Intruders. Proc. 6th European Symposium on Research in Computer Security (LNCS 1985), Toulouse, France, September (2000) 31-42.
19. S. Staniford-Chen, L. Todd Heberlein: Holding Intruders Accountable on the Internet. Proc. IEEE Symposium on Security and Privacy, Oakland, CA, August (1995) 39-49.