2010 Fourth International Conference on Network and System Security

Getting the Real-Time Precise Round-Trip Time for Stepping Stone Detection

Ping Li, Wanlei Zhou, Yini Wang
School Of Information Technology
Deakin University
Burwood, VIC, 3125, Australia
{Pingli, Wanlei.Zhou, Yiniwang}@deakin.edu.au

Abstract

Stepping stone attacks are often used by network intruders to hide their identities. The Round Trip Times (RTT) between the send packets and corresponding echo packets for the connection chains of stepping stones are critical for detecting such attacks. In this paper, we propose a novel real-time RTT getting algorithm for stepping stones which is based on the estimation of the current RTT value. Our experiments show that it is far more precise than the previous real-time RTT getting algorithms. We also present the probability analysis which shows that our algorithm has a high matching rate and a high accurate rate.

1. Introduction

The Internet has become more crucial these days but at the same time, the Internet attacks have increased dramatically in size and in scale. Instead of attacking a computer directly, most attackers launch their attacks through intermediary hosts they have previously compromised to hide themselves. These compromised computers are called stepping stones [1]. In this technique, attackers construct a chain of interactive connections on stepping stones using protocols such as Telnet or SSH. Attack commands or programs are sent from an attacker's machine, transferred by stepping stones, and then to the target machine. The final victim only sees traffic from the last hop in the chain of stepping stones, making it difficult for the victim to learn any information about the true origin of the attack.

The stepping stone detection approach is responsible for identifying the interactive connections which are in the chain of the stepping stones, which means stepping stone attacks can be blocked or traced back. Since the problem of detecting stepping stones was first discovered by Staniford-Chen and Heberlein [1], many approaches have been suggested in detecting stepping stones in encrypted traffic. They can be classified into three types: timing based, packet number based and Round Trip Time (RTT) based. Unlike other types of approaches that only use Send or Echo packets, RTT based approaches use Send and Echo packets together in order to detect stepping stones. As a result, RTT based approaches can filter unsymmetrical Internet packets and chaff packets, and can also be more resistant to network imperfections and intruder evasion than any other type of approach. Yung [2] was the first to propose a method to detect stepping stones by using RTT. The basic idea is to estimate the length of the downstream connection chain by computing the ration between packet Ack-delay and Echo delay (i.e. RTT). Yung [2] claims there is no reason to access a host through a long chain instead of a direct connection unless in some very special applications. In Yung's approach if the length of downstream connection chain is more than a specified number, the connection may be considered a stepping stone connection. However, Yung's estimating approach for connection chain length can only give good results when network traffic is relatively uniform. On the other hand, Yang and Huang [3] proposed a "Step-Function" approach to detect stepping stones using the RTT feature that RTT changes small for normal connections, but this change, proportionally increases with the number of stepping stones in the chain. The number of steps for RTT changes reflects the number of hosts in the connections. If the number of steps for RTTs changes on an interactive connection is more than a specified number, this connection may be considered a stepping stone connection. This approach can detect stepping stone correctly if the RTTs can be obtained correctly.

However it is not easy to obtain the RTT with high accuracy, as Echo packets do not have an obvious characteristic to identify correlated Send packets. Yung [2] used a statistical method to match TCP Send and Echo packets. It resulted in a correct match only when the Echo packet was received before the next Send packet was sent. The other issue is that it cannot be used in real-time. Yang and Huang [3] proposed Conservative

978-0-7695-4159-4/10 $26.00 © 2010 IEEE
DOI 10.1109/NSS.2010.36

and Greedy algorithms to obtain RTT. But this proposal is based on the assumption that every Send packet exactly matches one Echo packet. Yang [4] proposed a standard deviation-based clustering approach (SDBA) which calculates time delay between all send packets and echo packets, and finds the cluster with the smallest standard deviation. Although it can achieve high accuracy, it is inefficient and cannot be used in real-time. So accurately obtaining RTTs in real-time remains a challenge.

In this paper, we propose an Estimation-Based Algorithm (EBA) to achieve RTT in real-time. The EBA algorithm can be used together with the "Step-Function" [3] stepping stone detection approach to detect a stepping stone. It is different from previous real-time RTT getting proposals in that it calculates RTT estimation (ERTT) value to begin with, instead of finding a corresponding echo packet directly. Our experiments show that our algorithm is far more precise than other real-time RTT getting algorithms. We also present the theory analysis from the probability point, which shows that our algorithm has a high matching rate and a high accurate rate compared to the complicated non real-time SDBA [4] approach.

The rest of the paper is organized as follows. The detail of our Estimation-Based Algorithm is presented in Section 2. Section 3 gives the probability analysis. Some experimental application results are given in section 4, and finally we summarize this paper in Section 5.

2. Estimation-Based Algorithm (EBA)

Before presenting the algorithm, we present some definitions to begin with.

RTT: The packets sent in interactive connections from attacker (client) to target (server) are called Send packets; and the packets sent in the reverse direction are called Echo packets. The time delay between the Send packet and the corresponding Echo packet on a connection is called Round-Trip Time (RTT) for this interactive connection.

RTT sequence: A RTT sequence $\{RTT_1, RTT_2,\ldots, RTT_n\}$ is a series of real RTTs in chronological order calculated by the time delay between the Send packet and corresponding Echo packet on a interactive connection, where RTT_n is the RTT for the n^{th} Send packet.

ERTT: The estimation value for RTT.

ERTT sequence: A ERTT sequence $\{ERTT_1, ERTT_2,\ldots, ERTT_n\}$ is a series of ERTTs in chronological order calculated by the EBA algorithm, where $ERTT_n$ is the ERTT for the n^{th} Send packet.

ΔRTT: the deviation that RTT from ERTT.

ΔRTT sequence: A ΔRTT sequence $\{\Delta RTT_1, \Delta RTT_2,\ldots, \Delta RTT_n\}$ is a series of ΔRTTs in chronological order, $\Delta RTT_i = RTT_i - ERTT_i$.

FR (fluctuate range): The maximum value that RTT_i can deviate from $ERTT_i$.

Our algorithm is composed of two modules: the estimating module and the matching module. Next we will present the detailed algorithm description for each module and include some improvements.

2.1. The Estimating Module

The Estimating Module is responsible for calculating the ERTT. We use the first-order linear recursive filter to estimate the RTT, which is also used in the current TCP RTT estimation mechanism. For the RTT sequence $\{RTT_1, RTT_2,\ldots, RTT_n\}$ and ERTT sequence $\{ERTT_1, ERTT_2,\ldots, ERTT_n\}$ on a interactive connection, ERTT can be calculated by the last ERTT and RTT, as shown in equations (1) and (2)

$$ERTT_i = a * ERTT_{i-1} + (1-a)\, RTT_{i-1} \qquad (1)$$

$$ERTT_1 = RTT_1 \qquad (2)$$

In (1), a is the weighting factor used to adjust how quickly the estimation value responds to the real value. The weighting factor in the TCP RTT estimation mechanism by current TCP/IP implementation is normally set to 0.875, which has been used for many years and has been seen as reasonable up to now over the Internet [5]. We also tested parameter a by different values in our algorithm, and found that we can achieve the smallest standard deviations for ΔRTT, when a equals 0.875. The smaller the ΔRTT, the more precise the estimation. Therefore, we set parameter a to 0.875 in our algorithm.

To calculate ERTT, the key is how to find the first real RTT (i.e. RTT_1). From the previous analysis in this section, we know that it is inevitable that there are some time intervals between two consecutive Send packets which are considerably larger than the RTT of the network during an interactive terminal session. So it is reasonable to start or resume our estimation from these large time intervals. If two consecutive Send packets have timestamp differences of more than TI (a predefined time interval threshold), we will assume the existence of a large gap and then achieve the RTT_1.

Normally, we can consider the first Echo packet is matched with the first Send packet after the large gap. So we calculate RTT_1 as the time delay T_1 between the first Echo packet and the first Send packet.

To evaluate the accuracy of our estimating algorithm, we built a connection chain with 3 connections. We input simple characters with big intervals so the Send packets with Echo packets are one-to-one mapping and there is no overlap of RTT. We do this in order to get the real RTTs by one-to-one matching easily. And we found that the RTT distribution is more-or-less a Poisson

distribution with a relatively narrow range. At the same time, we calculated ERTT by equation (1) and (2) with the real RTT data we achieved. Then we compared the ERTT with the real RTT, and obtained the ΔRTT distribution which is near normal distribution, with more than 97% of the |ΔRTTs| smaller than 17 ms.

We also found that the standard deviation for the ΔRTT distribution is nearly the same as the standard deviation for the ΔRTT distribution. Table 1 shows standard deviation examples we experimented with in our tests.

Table 1. Standard deviation compare for RTT and ΔRTT distribution.

Examples	Standard deviation for RTT	Standard deviation for ΔRTT (ms)
1	1.735	1.771
2	2.841	2.827
3	3.663	3.722
4	5.312	5.538
5	6.469	6.651
6	9.016	9.043

2.2. The Matching Module

Because most of RTT_i fluctuate around $ERTT_i$ with a relatively narrow range, if the time delay between an Echo packet and the Send packet is in the range of $ERTT_i$ - FR and $ERTT_i$ + FR, we will consider this time delay as the RTT_i. This is the basic idea of the matching process.

We found that the ΔRTT distribution is near normal distribution. So the maximum ΔRTT (i.e. FR) is infinite in theory. But our destination is to get the real RTTs which are used to detect stepping stones by the "Step-Function" stepping stone detection approach [3]. The few too small or too big real RTTs are of no benefit to us, so we can filter these abnormal RTTs by selecting an appropriate FR. When the value of FR becomes bigger, more packets will be in the range of $ERTT_i$ - FR and $ERTT_i$ + FR, then the probability to find matched packets will be higher, but at the same time the incorrect probability will be higher as well. So the value of FR is critical for our algorithm. We will discuss the value of FR further in Section 3.

In our algorithm, we have a queue called SendQ, which stores the send packets in time order. When the time Interval between two consecutive Send packets is bigger than the TI, we will reset the SendQ. If we find the corresponding Echo packet for one Send packet, or if we are sure there is no corresponding Echo packet for that Send packet, we will delete that send packet from the SendQ queue.

By the estimating algorithm we can find the ERTT. Now whenever we capture an echo packet, we will get the first Send packet from SendQ and calculate the time delay T_{delay} between the Echo packet and the Send packet. If the T_{delay} is smaller than ERTT – FR, we consider there is no Send packet to match this Echo packet; if the T_{delay} is in the range between (ERTT – FR) and (ERTT + FR), we consider these to be matched with each other, and the RTT_i is T_{delay}; If the T_{delay} is larger than ERTT + FR, we consider there is no Echo packet to match this Send packet, and we will get the next Send packet to repeat the above process. Figure 1 describes the matching process.

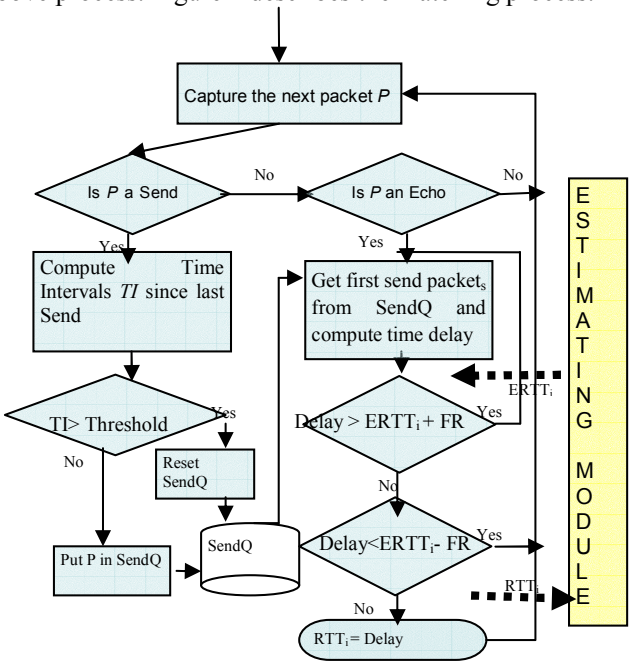

Figure 1. Matching module processing.

Through this matching process, we can achieve RTT, and store every RTT. At the same time as we input the RTT into the estimating process, we get the new ERTT for the continuous processing. The stored RTTs can be used to judge if the monitored host is a stepping stone by the "Step-Function" stepping stone detection approach [3].

3. Evaluation

It is impossible to know every real RTT in practical application, so therefore, we can't achieve the exact matching rate nor the accurate rate. But we can still evaluate them from the point of probability.

3.1. Matching Rate

Matching rate is defined as the ratio between the number of matched packet pairs and the number of Send packets having corresponding number of Echo packets. According to our algorithm, only the RTT whose difference with ERTT is smaller than FR can be matched. So FR is critical for our algorithm. The bigger the FR, the matching rate will also be higher; but the incorrect probability will be higher as well. In addition, our

intention is to achieve the real RTTs which are used to detect stepping stones by using the "Step-Function". The few too small or too big real RTTs can not benefit us, so our algorithm also has the filter's function.

Assume Echo packet Pe_i is the corresponding Echo packet to Send Packet Ps_i, the timestamps for Pe_i and Ps_i are Te_i and Ts_i, respectively. If Pe_i is selected to match Ps_i, the time delay between them is RTT_i. And we assume we also knew $ERTT_i$. Then we can get

$Ts_i + ERTT_i - FR < Te_i < Ts_i + ERTT_i + FR$

$ERTT_i - FR < Te_i - Ts_i < ERTT_i + FR$

$ERTT_i - FR < RTT_i < ERTT_i + FR$

$|RTT_i - ERTT_i| < FR$

We assume ΔRTT has standard deviation δ, and $u = \frac{FR}{\delta}$. We evaluate the matching rate, which is the probability that Ps_i has a corresponding packet to be found, i.e., the probability that Pe_i is selected to match Ps_i by using Chebyshev inequality as the following:

Matching rate = P (Ps_i has corresponding packet being found) = $P(|RTT_i - ERTT_i| < FR) > 1 - \frac{1}{u^2}$

So the matching rate is related to the value of u which is the ratio between FR and standard deviation of ΔRTT. In our experiments, FR was set to 30ms, which works well. We calculate with the previous standard deviation examples for ΔRTT we had achieved, and found the u and matching rate as shown in Table 2. We know that matching rates for all the standard deviation examples are higher than 90% which is high enough to detect a stepping stone.

Table 2. Matching rate examples for EBA.

Examples	Standard deviation forΔRTT (ms)	u	Matching Rate (%)
1	1.771	16.940	99.651
2	2.827	10.612	99.112
3	3.722	8.060	98.461
4	5.538	5.417	96.592
5	6.651	4.510	95.086
6	9.043	3.317	90.802

3.2. Accurate Rate

To begin with, we initially estimated the probability of making an incorrect choice of Echo packet Pe_i for Send packet Ps_i. There are two reasons that Pe_i is incorrectly selected to match Ps_i:

- Pe_i should be the corresponding packet for previous Send packets, but is not selected to match previous Send packets because the real RTT_{i-1} is more than ERTT + FR. In this case, the most probability is that Pe_i is the corresponding packet for last Send packet Ps_{i-1}. We assume the timestamps for Ps_{i-1}, Ps_i, Pe_i are Ts_{i-1}, Ts_i, Te_i, respectively, and the time delay between Te_i and Ts_{i-1} is RTT_{i-1}. So we can get

$Te_i > Ts_i + ERTT_i - FR > Ts_{i-1} + ERTT_{i-1} + FR$

$Ts_{i-1} + RTT_{i-1} > Ts_i + ERTT_i - FR > Ts_{i-1} + ERTT_{i-1} + FR$

$RTT_{i-1} - ERTT_i > Ts_i - Ts_{i-1} - FR > ERTT_{i-1} - ERTT_i + FR$

Since Pe_i is not selected to match Ps_{i-1}, ERTT is not calculated again. So $ERTT_i$ is equal to $ERTT_{i-1}$. Then

$RTT_{i-1} - ERTT_{i-1} > Ts_i - Ts_{i-1} - FR > FR$

In addition we assume L_{i-1} is the time interval between these two consecutive Send packets, i.e. $Ts_i - Ts_{i-1} = L_{i-1}$. And L is the smallest time interval between two consecutive Send packets. Then

$RTT_{i-1} - ERTT_{i-1} > L_{i-1} - FR$ and $L_{i-1} > 2FR$

$$RTT_{i-1} - ERTT_{i-1} > L_{i-1}/2 \quad (3)$$

- Pe_i should be the corresponding packet for Ps_{i+1} -- the next Send packet of Ps_i, but it is matched with Ps_i. Because the difference in timestamps of Ps_i and Pe_i is closer to $ERTT_i$ than the difference of timestamps of Ps_{i+1} and Pe_i, we assume the timestamps for Ps_i, Ps_{i+1}, Pe_i are Ts_i, Ts_{i+1}, Te_i, respectively, and the time delay between Te_i and Ts_{i+1} is RTT_i. Then we can get

$Te_i - Ts_i - ERTT_i < ERTT_i - (Te_i - Ts_{i+1})$

$Te_i - Ts_{i+1} + (Ts_{i+1} - Ts_i) - ERTT_i < ERTT_i - (Te_i - Ts_{i+1})$

$(Te_i - Ts_{i+1}) - ERTTi < -(Ts_{i+1} - Ts_i)/2$

$$RTT_{i-1} - ERTT_{i-1} < -L_{i-1}/2 \quad (4)$$

So we have $|RTT_i - ERTT_i| > L_i/2 > L/2$ from (3) and (4). And we assume ΔRTT has the standard deviation δ, and $v = \frac{L}{2\delta}$ achieves the probability that Pe_i is selected incorrectly to match Ps_i by using Chebyshev inequality as the following:

P (incorrect choice of Pe_i for Ps_i)

$= P(|RTT_{i-1} - ERTT_{i-1}| > L/2) < \frac{1}{v^2}$

Then the accurate rate, i.e. the probability to make a correct selection of a packet RTT can be estimated by using the following inequality:

Accurate rate= P (correct choice of Pe_i for Ps_i) > $1 - \frac{1}{v^2}$

Yang claims that the accurate rate of his SDBA algorithm [4] is higher than $1 - \frac{1}{q^2}$. Where $q = \frac{L}{2\sigma}$, σ is the standard deviation of RTT. We knew that the standard deviation of RTT is close to the standard deviation of ΔRTT, i.e. $\sigma \approx \delta$, then $v \approx q$. So our

algorithm has nearly the same accurate rate as SDBA. Yang [4] claimed that the probability of the accurate rate for his SDBA experiment examples was higher than 97%.

4. Application

As we had mentioned before, the EBA algorithm can be used together with the "Step-Function" [3] stepping stone detection approach to detect a stepping stone. The "Step-Function" approach is responsible for monitoring the steps of the RTT changes on interactive connections which reflect the numbers of connections in its downstream connections chain. When the RTTs change with more than a specified number of steps, the connection will be considered a stepping stone connection. Further action can then be taken such as a block or trace-back. Since the EBA algorithm is responsible for getting a stepping stone RTTs in real-time, we concentrated our experiment on the RTT values that the RTT getting algorithm can achieve and the levels of RTT changes. In here, we apply our EBA Algorithm in a real environment. At the same time we implement other real-time RTT getting algorithms -- the Greedy and Conservative algorithms [3] in the same environment, and present the comparable experimental results.

We estimate our experiments from two points of views: if the RTT getting algorithms can get RTTs with one level for a single connection and if the RTT getting algorithms can get RTTs with the correct number of levels during the establishment of a connection chain. In addition, as we mentioned previously, the typing speed and inputting commands can affect the ordering and mapping of the Send and Echo packets. So we conducted our experiments using two modes as well: slow typing speed and simple inputting commands, quick typing speed and complex inputting commands.

Firstly, we built a connection by SSH from host H1 to host H2. We also captured the SSH packets and applied Greedy, Conservative and EBA algorithms concurrently at host H1 from the time that host H2 was initially connected. We input simple commands by slow typing speed and complex commands with quick speed respectively at the connection terminal of H1. We obtained the result by simple inputting commands and slow typing speed as shown in Figure 2, and Figure 3 shows the result by complex inputting commands and quick typing speed. The X-axis represents the Send packet number and the Y-axis represents RTT values in units of ms.

From Figure 2, we know that all three algorithms are concentrated around 1 level if we can ignore the big protuberances. But apparently the EBA algorithm is better than the Greedy and Conservative algorithms as all the resulted RTTs are closely around 47 ms.

In Figure 3, the RTTs obtained by the Greedy algorithm are concentrated around 3 levels, and it will be incorrectly considered as a connection chain with 3 connections by the "Step-Function" stepping stone detection approach. For the Conservative algorithm, there are only 38 RTTs obtained which are far less than the 217 RTTs for the Greedy algorithm and 207 RTTs for the EBA algorithm. It will be hard for the "Step-Function" approach to judge what kind of connection it is due to a small number of RTTs. For the EBA algorithm, all the RTTs it obtained are closely around 49 ms, so the "Step-Function" approach can identify it is a single connection.

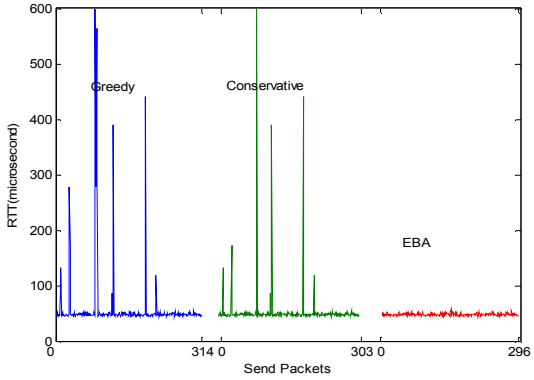

Figure 2. One connection with simple inputting commands by slow typing speed.

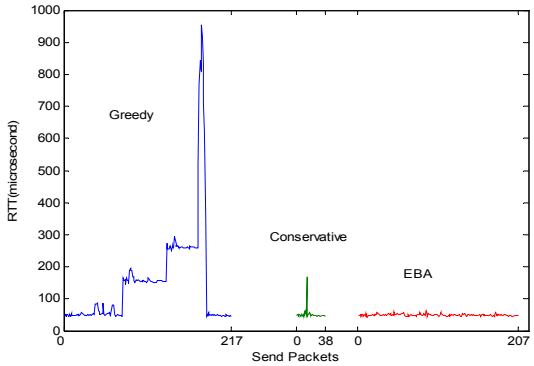

Figure 3. One connection with complex inputting commands by quick typing speed.

We then built a connection chain by SSH that passed through host H1 to host H2, then to hosts H3, and then to H4. We captured the SSH packets and applied the Greedy, Conservative and EBA algorithms concurrently at host H1 from the time that host H2 was initially connected, to the time the whole connection chain was built. We input simple commands using a slow typing speed and complex commands by quick speed respectively at the connection terminal of H1 during the chain building. We obtained the result by the simple inputting commands and slow typing speed as shown in Figure 4 and the result of complex inputting commands

and quick typing speed is shown in Figure 5, where the X-axis represents the Send packet number, and the Y-axis represents RTT values in units of ms.

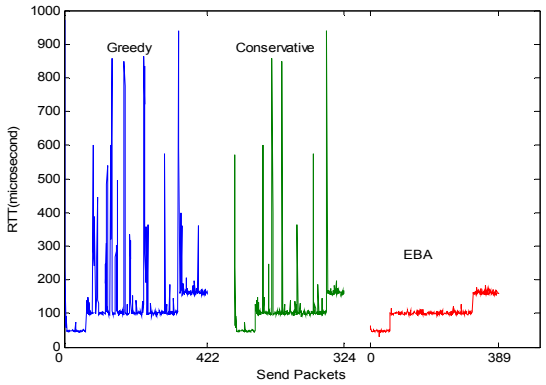

Figure 4. One chain with simple inputting commands by slow typing speed.

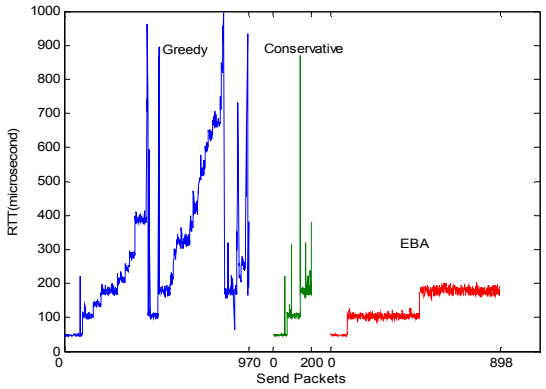

Figure 5. One chain with complex inputting commands by quick typing speed.

In Figure 4, the RTTs obtained by the Greedy and Conservative algorithms are approximately clustered around 3 levels. However both of them have too many large protuberances that may affect the identification of steps for the "Step-Function" approach.

From Figure 5 we know that the RTTs obtained by the Greedy algorithm are clustered around many levels, and the "Step-Function" approach will considered it as a stepping stone connection when it is just a single connection. For the Conservative algorithm, there are only 200 RTTs obtained which are far less than the 970 and 898 RTTs for the Greedy algorithm and the EBA algorithm, respectively.

In both Figure 4 and Figure 5, all the RTTs the EBA algorithm obtained are closely around 3 levels: 47 ms, 102ms and 170 ms. So RTTs achieved by the EBA algorithm can correctly reflect how many connections in its downstream connection chain by any kind of typing speed and inputting commands.

From all of our experimental results, we discovered that the number of Send packets which are matched by an EBA algorithm are all slightly smaller than those achieved by the Greedy algorithm. We also discovered the ratios of EBA and Greedy Send packet numbers for the above figures, which are all higher than 90%. As the Greedy algorithm matches all the Send packets even if they have corresponding echo packets, the real number of Send packets having corresponding echo packets should be smaller than the Greedy number of send packets. So, therefore, we are confident that the real matching rate for the above figures should be higher than 90%.

In addition to this, we also achieved the standard deviations of ΔRTTs for the above figures. These are between 1.771ms and 9.043ms. Although we cannot get the exact accurate rate from the above figure, our algorithm can achieve enough precise RTTs to detect stepping stones for a wide range of standard deviations of ΔRTTs.

5. Conclusion

In this paper, we have proposed a real-time simple precise RTT getting algorithm for stepping stones. This algorithm is different from previous real-time RTT getting algorithms in that it attempts to estimate the RTT initially instead of finding the corresponding echo packet directly. We present the probability analysis in theory which demonstrates that our algorithm has more than a 90% matching rate, and as high an accurate rate as the non real-time complicated RTT getting algorithm, SDBA. This indicates our experimental results using our algorithm are much more precise than previous real-time methods to detect stepping stones.

6. References

[1] Staniford-Chen, S. and Heberlein, L., "Holding Intruders Accountable on the Internet", in *Proceedings IEEE Symposium on Security and Privacy*, Oakland, CA, USA, 1995, pp. 39–49.

[2] Yung, K.H., "Detecting Long Connecting Chains of Interactive Terminal Sessions", in *Proceedings International Symposium on Recent Advance in Intrusion Detection*, Zurich, Switzerland, 2002, pp.1-16.

[3] Yang, J. and Huang, S., "Matching TCP/IP Packets to Detect Stepping-Stone Intrusion", *International Journal of Computer Science and Network Security*, 2006, VOL.6 No 10.

[4] Yang, J. and Huang, S., "Probabilistic Analysis of an Algorithm to Compute TCP Packet Round-Trip Time for Intrusion Detection", *Journal of Computers and Security*, Elsevier Ltd., 2007, pp137-144, Vol. 26.

[5] Mills, D.L.: Internet Delay Experiments, IETF document, http://www.ietf.org/rfc/rfc889.txt (1983)