Detecting Intermediary Hosts by TCP Latency Measurements

Gurvinder Singh, Martin Eian, Svein Y. Willassen, and Stig Fr. Mjølsnes

Department of Telematics,
Norwegian University of Science and Technology
{gurvinde,eian,sventy,sfm}@item.ntnu.no

Abstract. Use of intermediary hosts as stepping stones to conceal tracks is common in Internet misuse. It is therefore desirable to find a method to detect whether the originating party is using an intermediary host. Such a detection technique would allow the activation of a number of countermeasures that would neutralize the effects of misuse, and make it easier to trace a perpetrator. This work explores a new approach in determining if a host communicating via TCP is the data originator or if it is acting as a mere TCP proxy. The approach is based on measuring the inter packet arrival time at the receiving end of the connection only, and correlating the observed results with the network latency between the receiver and the proxy. The results presented here indicate that determining the use of a proxy host is possible, if the network latency between the originator and proxy is larger than the network latency between the proxy and the receiver. We show that this technique has potential to be used to detect connections were data is sent through a TCP proxy, such as remote login through TCP proxies, or rejecting spam sent through a bot network.

Key words: TCP, Latency, Intermediary Host, Proxy Server, Botnet, Intrusion Detection, Cyber Security

1 Introduction

The use of intermediary hosts as stepping stones to conceal tracks is common in Internet misuse. By using intermediary hosts, the misuser will make it significantly more difficult to trace and detect his origins. When intermediaries are used as stepping stones, the investigator has to identify the communicating host was a stepping stone, and take steps to secure evidence on that host. This process may be cumbersome, especially if the stepping stones are located in different jurisdictions [7].

Further, there might be no evidence in the intermediary host that can be used for further tracing, either because the investigation has taken too long, or because the intermediary host has been specifically configured to avoid recording anything about the originator. Previous research has identified stepping stones to be particularly common during computer intrusions. In these instances, remote

attackers have been found to log in by remote shell through several intermediaries, effectively using them as TCP proxies [2].

Recent research has also revealed that spread of spam by the use of intermediary hosts is common. These intermediaries have been found to be computers infected by malicious programs and are used as SOCKS proxies without the owners' knowledge or consent [13]. It is desirable to find a method to detect if the communicating party is using an intermediary. This would allow activation of a number of countermeasures that would neutralize the effects of the misuse, and make it easier to trace the perpetrator. It would for example be possible for a mail server to stop receiving email messages from such SMTP requests or login server to deny entry for connections via an intermediary, or perhaps to activate more verbose logging and notify a system administrator.

In this work, we propose a novel approach for detecting whether the host initiating a TCP connection is an intermediary or not. The approach is based on the observation that data in certain situations arrive in bursts, and the interval between the bursts depends on the network latency of the different steps between the receiver and the originator of the data. If the network latency between the client and the proxy is larger than the observed latency between the proxy and the server, it can be inferred that the traffic is passing through one or more intermediaries.

Several researchers have studied the problem of detecting stepping stones in the past. Staniford-Chen and Heberlein proposed using traffic content thumb prints to find correlation between traffic on both sides of an intermediary [15]. This approach has been extended by several authors to also allow detection for encrypted traffic by recognizing the timing information on the connection [11, 5, 12]. Others have studied the limitations of these approaches under active countermeasures [18]. Coskun and Memon propose steppingstone detection based on the timing and correlation of ingress-egress packet flows at the network border [4].

These approaches all require measurement points in the network distributed in such a way that there is at least one measurement point on each side of the intermediary. In this work, no measurement point in the network path is assumed. Instead, we investigate the possibility to infer the existence of a TCP proxy from observations at the receiving host end alone.

The experiments in this work have been conducted in a controlled lab environment where the delay conditions in the lab shows similar behavior as for the Internet environment. We focus here on investigating the feasibility and the potential for this new technique of detecting intermediary hosts, which we answer positively. The next step will be to consider the network congestion, effects of application layer protocols, and other effects of observed network behavior.

2 Background

Transmission Control Protocol (TCP) is the prevalent transmission protocol on the Internet. TCP implements mechanisms to control the data rate to the

network conditions, including bandwidth, latency of the connection and speed of the receiving host. Network latency between two hosts on a network can be defined as the amount of time elapsed from data has been sent from the sending host until it has been received at the destination host. On the Internet, network latency between two communicating hosts is the sum of the latencies in all the routers and network links between the communicating hosts. Network latency between two hosts on the Internet can be measured by sending icmp echo request packets from one of the hosts to the other. Unless configured to do otherwise, the other host will then reply with icmp echo reply packets, and the time elapsed between sending echo request and receiving the reply is equal to the round trip time (Δ) of the connection.

$$\Delta = 2 * (\delta_{prop} + \delta_{proc} + \delta_{queue}) \tag{1}$$

where δ_{prop} is the propagation delay, δ_{proc} is the delay caused by routers and end hosts while processing the packets and δ_{queue} is the delay caused by waiting time in the router and end hosts's queue.

The network latency can be approximated as half the value of round trip time. This measurement can be done with the ping command. However, some network service provider have special QoS class for echo packets to show the better service. In the paper, we used the value of Δ computed during the TCP 3-way handshake to avoid the special QoS class case for the ping packets.

The TCP control mechanisms include various algorithms designed to transmit the data from the sender to the receiver as efficiently as possible, while adjusting to the current conditions of the network and avoiding overloading the network. One of these mechanisms is the TCP flow control [1] and another is the Nagle [10] Algorithm. These algorithms are of special interest, since their application result in a possibility to observe network latency in a TCP data flow. In flow control mechanism, TCP window size limits the amount of unacked data in to the network and results in a possibility to observe the network latency.

TCP uses the Nagle Algorithm to adjust the sending rate of traffic where small amounts of data arrive at the TCP-layer frequently. The Nagle Algorithm works by inhibiting the sender from sending new TCP segments if any previously transmitted data on a connection remains unacknowledged [10]. New TCP data can be sent immediately only in two cases: First, if the TCP connection is just established or it was idle for some time. Second, if the size of the data to be sent is larger than the Maximum Segment Size (MSS) and the available window size of the receiver is also larger than MSS. If none of these conditions are satisfied, TCP will wait until an ACK has been received for the previously transmitted data until data is transmitted. As a result, data will be sent in bursts with intervals in between. The length of the interval is equal to the time it takes for the ack for the previously sent data to arrive. In other words, the burst interval depends on the value of Δ in the connection between the two hosts.

3 Latency Propagation Theory

The underlying observation motivating this research is that the TCP flow control causes network traffic to be sent in bursts where the interval between each burst depends on the network latency. This property holds for example in the case of a large file transfer, whereas in typical remote login protocols this property only holds if the Nagle algorithm is in use.

The idea is that if these bursts are detectable when data traffic is sent via a TCP-level *proxy*, then analysis of bursts and their intervals can be used at the receiving end to determine if the observed sender is the original sender or a TCP proxy.

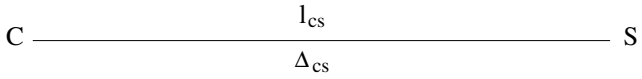

Fig. 1. A TCP connection between client C and server S on link l_{cs} with Δ_{cs} round trip time

Consider a connection as shown in Fig 1 on a link l_{cs} between a client C and a server S having round trip time value equal to Δ_{cs}. On the link l_{cs}, the TCP traffic can be either interactive type with small size segments or traffic due to a large file transfer with large size segments. When the transmitted data is *interactive traffic*, such as remote login and remote desktop session, then the Nagle Algorithm is applied to the traffic. Exceeding the Maximum Segment Size will enable the Nagle Algorithm.

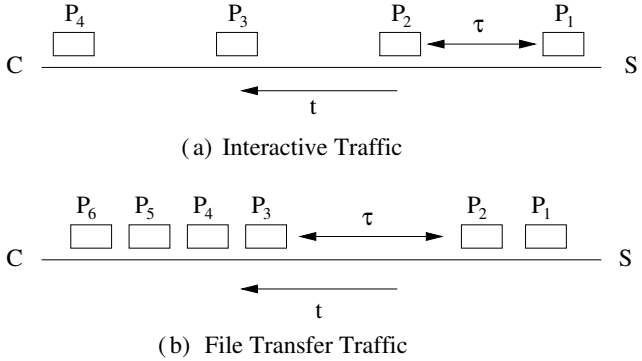

Fig. 2. Behavior of TCP burst interarrival time (τ) with (a) interactive and (b) file transfer traffic type

With a TCP connection between client C and server S with round trip time Δ_{cs}, an observer at the server S can now observe the traffic pattern of the incoming data from the client C. The behavior observed by the observer will

look similar to the pattern as shown in *part (a)* of Fig 2. Since the sender C will have to wait for unacked data to be acked before it can send more data, there will be an interval τ of value at least equal to Δ_{cs} between the data bursts from C as observed from S, as shown in Fig 2.

The time interval τ is called as *Burst Interarrival Time(BIT)*, which is defined as the *time difference between the arrival time of the received burst and the arrival time of the previous burst.* The burst length in case of interactive traffic is equal to single packet, due to the small data size.

Now assume the received traffic at server S is of type file transfer and thus the segment size is large. Now the Nagle algorithm will not be involved in limiting the client C, however the TCP flow control mechanism limits the client C by *CWND* window size in sending large numbers of segments. In the initial phase of the connection, the window size is equal to 2-3 segments. Subsequently, the client C waits for an ack to arrive from server S. This will result in a time interval τ which will be of value comparable to Δ_{cs}. This pattern is shown in the *pat b)* of Fig 2. The value of delay between packet P_1 and P_2 is very small as both segments are sent together by client C. But after sending 2 segments client C waits for an ack to arrive and therefore the next burst of packets will arrive after interval τ of value comparable to Δ_{cs}.

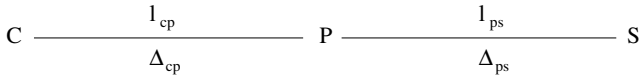

Fig. 3. TCP connections between client C and proxy P, and proxy P and server S on link l_{cp} with Δ_{cp} and link l_{ps} with Δ_{ps} correspondingly

Now consider the connection setup of Fig 3, where P is acting as a TCP proxy. In this case, an observer at server S may not know that P is acting as a proxy, and may erroneously believe that the host P is the originator of the communication.

Data received at P from C is immediately forwarded to S by the proxy program running at P. Thus, when segments arrive at P, the data will already be in bursts, with burst interval depending on Δ_{cp}. The resulting interval between data bursts perceived by the observer at S will then depend on both Δ_{cp} and Δ_{ps}.

We hypothesize that the burst interval observed at the server S will depend on the largest of Δ_{cp} or Δ_{ps}. If Δ_{cp} is larger than Δ_{ps}, the burst interval as observed at S will depend on Δ_{cp} and not on Δ_{ps}. If this is the case, an observer at S can use the burst interval to determine if P is the real originator of the connection, by measuring the Δ_{ps} and comparing with the observed burst intervals τ. See Fig. 4

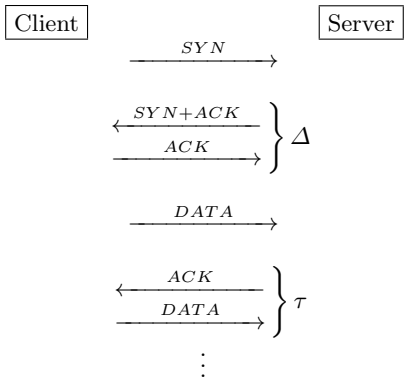

Fig. 4. The TCP threeway handshake and data segments measurements of the round trip time Δ, and the burst interarrival time τ.

4 Statistical Model of Burst Inter-arrival Time

The purpose of our statistical model is to distinguish between the l_{cs} connection and the l_{ps} connection. We construct a model of the l_{cs} connection to achieve this purpose. As explained in Section 3, we expect Δ_{cs} and τ to have similar values. Thus, we assume that there is a linear relationship between Δ and τ. We model the round trip time as $\Delta + \epsilon_\Delta$ and the burst inter-arrival time as $\tau + \epsilon_\tau$, where ϵ_Δ and ϵ_τ represent the random errors in δ_{prop}, δ_{proc} and δ_{queue}.

We have to make several assumptions to be able to construct a hypothesis test. We assume that ϵ_Δ and ϵ_τ are independent and identically distributed. This assumption implies that the network is not congested, and that none of the nodes have strained computational or memory resources. Furthermore, we assume that ϵ_Δ and ϵ_τ follow a normal distribution. The normality assumption is investigated empirically in Section 5.

We use simple linear regression with Δ as the independent variable and τ as the dependent variable to model the linear relationship between Δ and τ . The slope b and intersection a can be estimated based on an experiment with n samples, where each of the samples is a tuple (Δ_i, τ_i). The simple linear regression model is based on the assumption that the independent variable is exact, which is not the case in our experiments. However, the model still provides an unbiased estimate of the slope b and intersection a of the linear relationship. We also used an orthogonal regression model to verify that the results of the simple linear regression were unbiased. The orthogonal regression model resulted in the same parameters a and b.

We use the linear regression model to construct a prediction interval for the τ values to be observed. (Walpole et al. [17, p.410] or other basic statistics textbook will explain how to construct a prediction interval for a simple linear regression model.) Under the assumptions presented above, a $100(1-\alpha)\%$ one-

sided prediction interval is:

$$\{(\Delta_0, \tau_0) | \tau_0 < a + b\Delta_0 + t_\alpha s \sqrt{1 + \frac{1}{n} + \frac{(\Delta_0 - \overline{\Delta})^2}{\sum_{i=1}^{n} (\Delta_i - \overline{\Delta})^2}}\} \tag{2}$$

where

$$s = \sqrt{\frac{\sum_{i=1}^{n} (\tau_i - \overline{\tau})^2 - b \sum_{i=1}^{n} (\Delta_i - \overline{\Delta})(\tau_i - \overline{\tau})}{n - 2}}$$

t_α is a value of the Student-T distribution with $n - 2$ degrees of freedom. $\overline{\Delta}$ represents the mean of the Δ_i values, and $\overline{\tau}$ represents the mean of the τ_i values from the linear regression.

The following hypotheses are defined to use the model for hypothesis testing:

$$H_0 : \Delta_0 = \tau_0$$

$$H_1 : \Delta_0 < \tau_0$$

H_0 represents the l_{cs} connection and H_1 represents the l_{ps} connection. A tuple (Δ_0, τ_0) can then be measured for a single connection. If the measured τ_0 value falls outside the prediction interval, H_0 is rejected and we assume that a proxy is being used.

5 Experimental Setup

Three experiments are performed to test the proposed hypothesis. The first experiment is a RTT experiment, which tests the latency behavior of the lab setup against the real world Internet latency behavior. This will show how much the setup resembles with real world and applicability of the results. The second experiment is conducted using the connection setup as shown in Fig 1. The purpose of this experiment is to see the variation in τ values under different Δ values. We made our model based on setup shown in Fig 1. The last experiment is conducted to test validity of the model to detect intermediary host under the presence of a proxy host in the connection. The setup for third experiment is shown in Fig 3. The result from all these experimets are described in section 6.

The experiments are done under controlled laboratory conditions to eliminate unknown factors and to ensure repeatability. Three computers are configured with Linux distribution Ubuntu 9.10 running kernel 2.6.31-5. These are attached to the same 100 Mb Ethernet LAN and given roles as server (S), client (C) and proxy (P). The round trip time between any of the three computers on the LAN is approximately 1 ms.

In the experimental setup data is generated at the client side and sent to the server with varying latency on the connection in between as shown in Fig 2.

The arriving data bursts can then be observed at the receiving server S and the results can be checked for consistency with the hypothesis. If the results from this experiment do not refute the hypothesis, the experiment can proceed with introducing a proxy P between the client and the server as shown in Fig 3.

To generate the traffic, a different traffic generating program named Traffic Generator (TG), tool developed at SRI International and University of Southern California [9] is used. In the experiments, TG is used to generate TCP traffic on the client C and to sink the traffic at the server S. The experiments are conducted with different packet length and time distributions as described in the next section.

GNU Netcat was used as proxy [6]. Netcat is a simple utility, which reads and writes data across the network connections using TCP and UDP transport protocols. In the experiments, netcat was setup as a proxy in the following way:

```
$ mknod backpipe p
$ nc -l -v -p 1234 0<backpipe | nc 129.241.209.XXX
  1234 1>backpipe
```

When the client connects to this port and sends data to it, a new TCP connection is opened to the server (129.241.209.XXX) at port 1234 and received data is forwarded to it. Any data received in the opposite direction is forwarded back to the client through the pipe.

To observe the results on the server S, the packet sniffer Deamonlogger [14] was used. It was necessary to increase the amount of memory used for the capturing engine to avoid packets being dropped. Further, the Wireshark packet analyzer [3] was used to analyze the captured packets.

To create network latencies resembling Internet latencies in the laboratory, Netem coupled with the Traffic Control (tc) tool was used. Netem and the traffic control tool are parts of the *iproute2* package of the Linux kernel. The traffic control tool uses Netem to emulate Internet behavior. The queuing architecture of Linux kernel is shown in Figure 5. The queuing discipline sits between the protocol output and the network device. The queuing discipline is an object with two interfaces. One interface receives packet from IP protocol and another interface forwards these packets to the network device. The queuing discipline makes the decision of forwarding packet based upon the defined policies.

6 Burst Interarrival Time Results

6.1 RTT Experiment Results

To make sure that the lab setup resembles with the real world behavior, the behavior of induced delay via netem is tested against the real world round trip delay results. An experiment has been performed to ping the Univ. of Tromsø web server from the lab system. Thus results obtained from pinging the web server are compared with the result from pinging the lab server from the same system with similar Δ values.

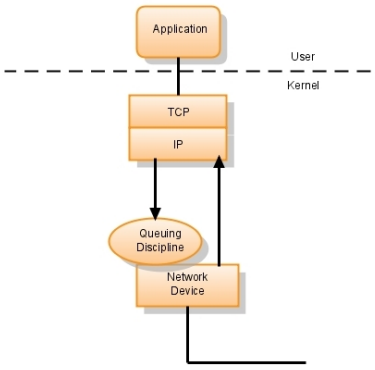

Fig. 5. Linux Queuing Architecture

Four experiments are performed to compare our lab setup to a real world scenario and to investigate how close the Δ values follow a normal distribution. Two experiments are performed on the lab setup, and two experiments are performed on the Internet connection. The Internet connection from the lab to the University of Tromsø has no preferential treatment of ICMP packets. 100,000 ICMP echo requests are issued in each experiment, and the Δ value recorded with microsecond accuracy.

Scenario	# Samples	Median	Mean	Stdev
Lab 1	100,000	$14719\mu s$	$14768\mu s$	$111\mu s$
Lab 2	99,998	$14714\mu s$	$14766\mu s$	$173\mu s$
UiT 1	99,995	$14869\mu s$	$14872\mu s$	$61\mu s$
UiT 2	100,000	$14869\mu s$	$14872\mu s$	$69\mu s$

Table 1. Results from the RTT validation experiment

Table 1 shows the estimated parameters of the sampling distributions for each experiment under the normality assumption. The estimated parameters show that the measurements from the real world scenario have less variance and a median closer to the mean than the measurements from our lab setup. This indicates that the results from our lab could be applicable to the real world scenario we use as a comparison.

Figure 6 shows the normal quantile-quantile plots for each of the experiments. The normal Q-Q plots show that an overwhelming majority of the measured Δ values follow a normal distribution. However, the distributions have a long right tail. This result is expected, as any deviation from the distribution due to excessive processing or queueing delays will result in a higher Δ. The lower part of the distribution is bounded by the propagation delay.

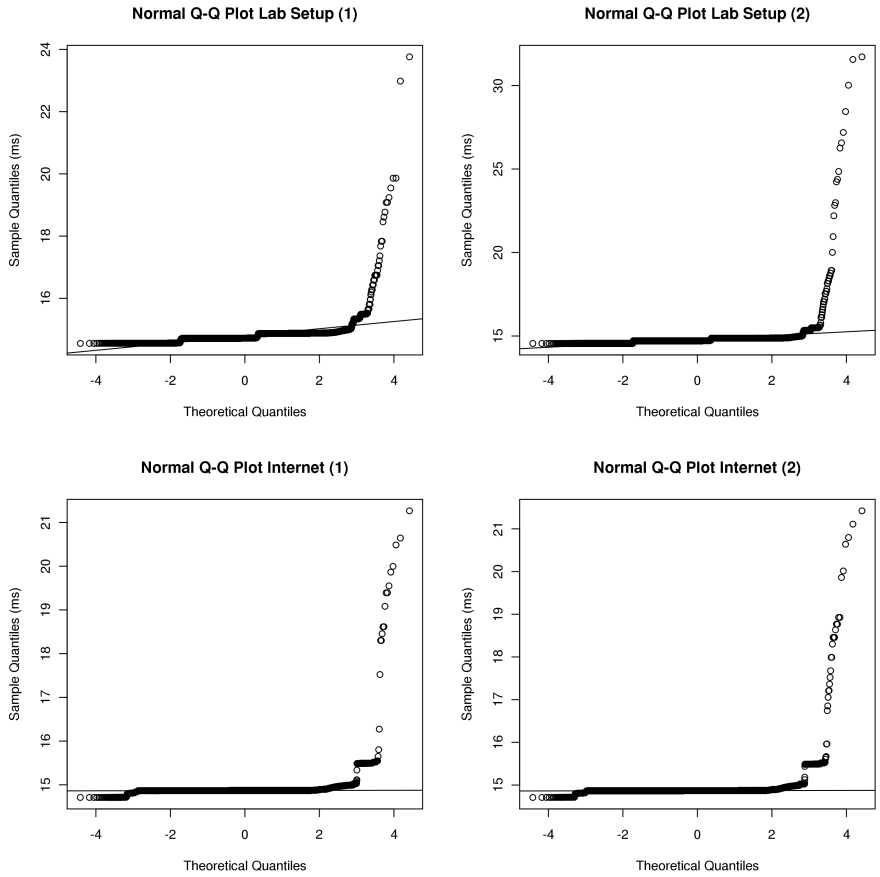

Fig. 6. Normal Q-Q plots for the RTT validation experiment. Most of the measurements follow the normal distribution, but there is a long tail to the right in each of the plots.

We use the estimated parameters from the first sample in each scenario to compute $100(1-\alpha)\%$ two-sided prediction intervals for the distributions, under the normality assumption. Table 2 shows the results. We use the Student-T distribution to compute the prediction intervals. We then use the second sample in each scenario to test the estimated prediction interval. Lost packets are considered to have a Δ value higher than the prediction interval. The goals of this test are to determine how closely the prediction interval of a normal distribution matches the measured Δ values, to quantify the impact of the long right tail, and to compare the results from our lab setup to the real world scenario.

The effect of the long right tail can clearly be seen for $\alpha/2 \leq 0.001$. For these α values, none of the measured Δ values are lower than the prediction interval, but approximately 200 of the Δ values exceed the prediction interval.

$\alpha/2$	$E(\Delta)$	$\Delta_{Lab} <$	$\Delta_{Lab} >$	$\Delta_{UiT} <$	$\Delta_{UiT} >$
0.1	10,000	4,216	615	51	885
0.05	5,000	4,181	443	51	606
0.01	1,000	0	237	49	299
0.005	500	0	237	49	202
0.001	100	0	236	0	199
0.0005	50	0	236	0	199
0.0001	10	0	222	0	199

Table 2. Number of values lower and higher than the $100(1-\alpha)\%$ prediction intervals

These are the values on the long right tail. The right tail thus puts a lower bound on the α value, where decreasing it further has little effect. The optimal value is $\alpha = 0.001$, where we only have to consider the one-sided prediction interval larger than the estimated mean. α represents the probability of a type I error (false positive). The lowest practical α value we are able to achieve, given the distribution of Δ, is between 0.002 and 0.003. For the rest of this paper, we will use the conservative assumption that $\alpha = 0.001$ gives a 0.5% probability of type I errors.

Finally, Table 2 also shows that for $\alpha \leq 0.001$, the results from our lab setup and the real world scenario are very similar. In both cases, a prediction interval computed under a normality assumption gives less than a 0.5% probability of type I errors for $\alpha = 0.001$. The results in Section 6 are thus applicable to the real world scenario as well.

6.2 Burst Interarrival Time Results from No Proxy Case

To study the TCP behavior on a connection l_{cs}, the data traffic is generated by TG on the connection using an exponentially distributed packet size with a mean of size equal to MSS bytes. The experiment is performed to transfer 10 Kbytes of data from a client (sender) to a server (receiver). The τ values were measured at the server side from the incoming bursts.

The delay (Δ) values chosen for experiments are varies from 1ms - 50ms with an increment of 1ms. Initially, the tests are conducted using these values to study the IPT behavior without introducing a proxy system in the network path. The values of delays are chosen so to observe the effect of variation of Δ values on the observed pattern at the server side. The connection setup in this case is similar to shown in the Fig 1.

We measure 20 tuples (Δ_i, τ_i) for each of the increments, for a total of $n = 1000$ samples. We then use the samples to perform a simple linear regression. Figure 7 shows the measurements and the estimated regression line. The estimated intercept is $a = 144\mu s$ and the estimated slope is $b = 0.999$, which are very close to the expected values $a = 0\mu s$ and $b = 1$ for a perfect linear relationship. The R^2 value for the regression is 0.9998.

Figure 8 shows a normal Q-Q plot of the residuals. Most of the residuals follow a normal distribution, but as in the previous experiment, the distribution has a

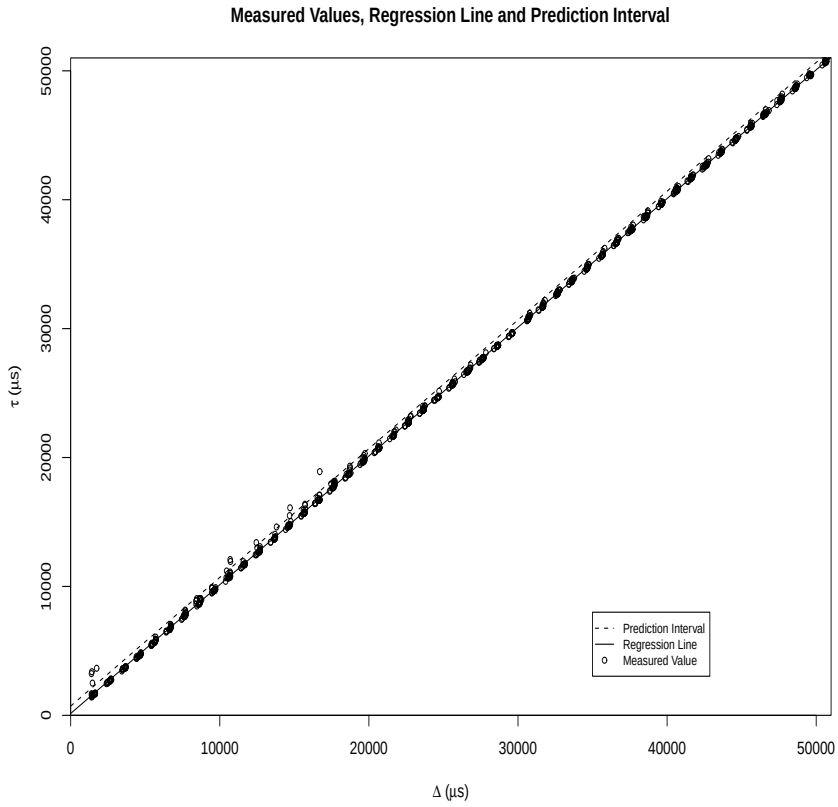

Fig. 7. The measured values, regression line and prediction interval with $\alpha = 0.001$. The number of samples outside the prediction interval is 12 (1.2% of the samples).

long tail. However, the long tail starts earlier in the residuals than in the RTT experiment. Thus, the expected number of observations outside the prediction interval will be higher than in the RTT experiment. We use Formula 2 to compute a 99.9% one-sided prediction interval ($\alpha = 0.001$), which is shown in Figure 7 together with the regression line and measurements. 12 of the samples (1.2%) in the linear regression model are outside the prediction interval, so we cannot make the same conclusion as in the RTT experiment. Based on the results, a conservative assumption is that 95% of τ values to be observed fall within the 99.9% prediction interval. Thus, when we use $\alpha = 0.001$, the actual false positive rate is 5%, rather than 0.1%. The residual standard error is $179\mu s$, which is similar to the results from the previous experiments.

From the result, we have seen that the bursts arrived at the server side after corresponding delay Δ_{cs} of the l_{cs} connection. The similar behavior is seen in all

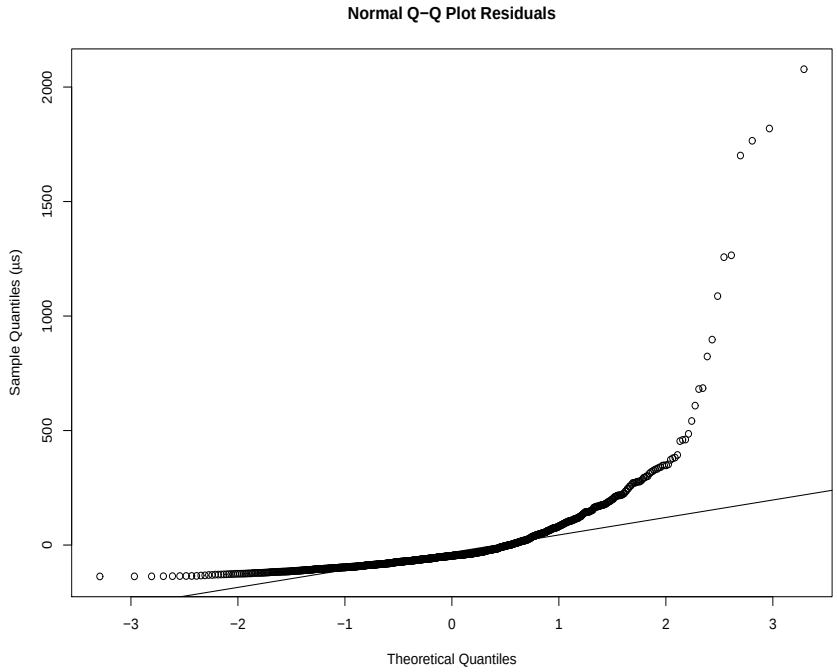

Fig. 8. Normal Q-Q plot of the residuals. The right tail is significantly longer for the residuals of the linear regression than for the RTT experiments.

the test cases. The observed behavior can be explained by the TCP flow control algorithm, which limits the amount of data to the sender by window size. The client must then wait for ACK to arrive before sending more data.

The client sends the data bursts to the server and upon receiving the data, the server acknowledges it and this acknowledgement arrives at the client side after completing journey from server to the client. This result in arrival of ACK at the client side after one complete RTT delay of the connection and the next data packet at the server end arrives after this delay. The observed data is thus consistent with the hypothesis that burst interarrival time is related to the network delay.

6.3 Burst Interarrival Time Results from Proxy Case

Now to study the TCP behavior under the presence of the proxy system, we introduce a proxy system between the server and the client. The connection setup of this test case is shown in Fig 3. The delay between server and the proxy system is set to 20ms and delay between the proxy and the client system is varying from 10 ms, 20 ms, 25ms, 30ms and 50ms respectively. The reason

behind choosing the specific delay conditions is as, the given settings address the case when the delay on client-proxy connection is less, equal and higher than server-proxy connection. Therefore we will see the IPT behavior in all five cases.

We measure 100 samples (Δ_i, τ_i) for each of the cases $\Delta_{cp} \in \{10ms, 20ms, 25ms, 30ms, 50ms\}$. Figure 9 shows the results. The approximate prediction interval shown in the figure is based on the maximum Δ value measured to be able to illustrate the results in a single figure. We then use the linear regression model to perform the hypothesis test from Section 4 on each sample. For the cases $\Delta_{cp} \leq 20ms$, we do not detect the proxy in any of the samples. However, for the cases $\Delta_{cp} \geq 25ms$, we detect the proxy in all of the samples.

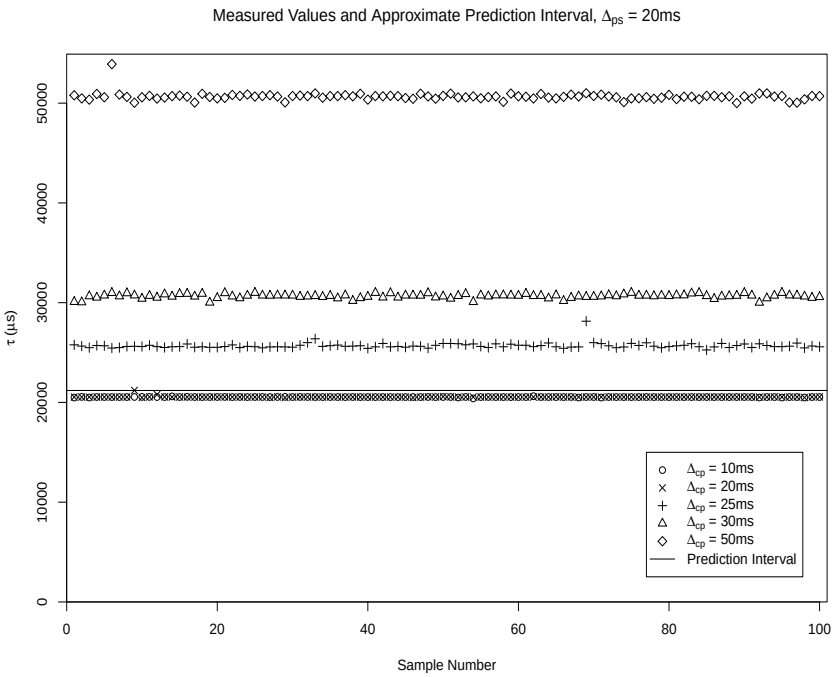

Fig. 9. An illustration of the results for the proxy case. The approximate regression line was based on the maximum Δ value measured. For the experiments where $\Delta_{cp} \leq 20ms$, we are not able to detect the proxy at all. However, for the experiments where $\Delta_{cp} \geq 25ms$, we achieve a 100% detection rate.

When Δ_{cp} is either 10ms or 20ms, the measured τ is approximately 20ms at the server side, which is the value of Δ_{ps}. In the other test cases, when Δ_{cp} is higher than 20ms, the measured τ at the server side is significantly higher than 20 ms. This is caused by the TCP flow control algorithm, which limits the

amount of data to the sender by the window size of the connection. The client must then wait for an ACK to arrive before sending more data. Thus, in this case, the observed τ values at the server is different from what would be expected with the Δ_{ps} between the proxy and the server.

We observe that if Δ_{ps} is higher than Δ_{cp} then it is possible to infer the presence of a proxy system from the observed τ values of the incoming bursts. The proxy system will receive data from the client within the Δ_{cp} delay of its connection with the client. As a result, the data arrive at the server side with τ values comparable to Δ_{cp}, which are higher than Δ_{ps} and make the detection of proxy system possible. The higher the value of Δ_{cp} is compared to Δ_{ps} higher the accuracy is in detection of proxy system.

From the above results, we see that the TCP flow control algorithm behaves differently under different delay conditions. Similar behavior has been shown when the traffic with small segment size was generated and the τ values from the initial stage were compared with the Δ value of the underlying connection. This behavior was, as hypothesized, due to the Nagle algorithm, which inhibits sender from sending small size of data, until the arrival of ack of unacked data.

Thus, by monitoring the τ values of incoming bursts at the receiver end, it is possible to detect the presence of intermediary hosts by comparing the τ values against the incoming connection Δ value. If the τ value is comparable to the Δ delay of the incoming connection, then connection is most likely to be the direct connection, otherwise the connection can be marked as an incoming connection through intermediary hosts.

7 Discussion and Future Work

The result from RTT experiment has shown that the lab setup conditions are similar to the Internet round trip time behavior. Thus the result obtained from the experiments performed in lab under varying Δ values are applicable to internet with high accuracy and detection rate. The result from experiments performed on different network conditions has shown that it is possible for the receiver of a TCP connection, under certain circumstances, to infer if the other end of the TCP connection is the originator or if it is acting as a TCP proxy. This can be inferred by measuring (τ) values of the incoming bursts and correlate with the measured round trip time (Δ) for the connection.

The accuracy in detecting intermediary hosts depends upon the how accurate are the values of Δ and τ. if the Δ value is high compared to real value, due to network congestion or due to the outlier value of Δ as seen in the long tail nature of its distribution, then there will be a probability of false negatives. However if the τ value is high again either due to congestion or outlier value, then there is a significant probability to have the false positives. The lower bound for both Δ and τ values is bounded by the δ_{prop}, so the probability of having false positives or false negatives due to low values is very low. If the detection mechanism detects the incoming connection as proxy connection, then server can reset the connection. With given 5% false positive rate, the probablity of detecting the

same connection will decrease by $(5\%)^n$ due to independence between the incoming connection. Thus probability of detecting the legitimate connection as a proxy connection decreases exponentially. Moreover higher the value of Δ_{cp} is compared to Δ_{ps} higher the accuracy is in detection of intermediary host.

During the research work, we have observed that the proxy type used in the communication session results in different behavior at the receiver end. In the above experiments, we have used Netcat as a proxy, which sends the ack to the client C of received data and then forwards the data to the sever S. However during our research work, we have found some programs, which works as a TCP level proxy such as *iprelay* [16], which does not send the ack to the client C until they have received from the server S. This behavior will result in higher τ value, which is equal to sum of the Δ values on both the connections.

The experiments in this work have been conducted in a controlled lab environment and the delay conditions in the lab has shown similar behavior as of real Internet environment. However, considering the congestion in the network and effect of application layer protocols on the observed behavior for intermediary hosts detection is an area of further study.

The generalized question of a sequence of intermediary host connections can be further investigated.

8 Conclusion

This work has explored the possibility of determining whether a host communicating via a TCP connection is the data originator or just acting as a TCP proxy, by measuring the inter packet arrival at the receiving end of the connection. Our results indicate that this is possible, if the network latency between the originator and proxy is larger than the network latency between the proxy and the receiver. This novel method has applications in various domains such as reject malicious remote logins through TCP proxies, or reject spam messages send through a proxy bot network, or block the access to restricted media contents when request arrives from a proxy host or detection of the Tor [8] usage in an incoming connection.

References

1. M. Allman, V. Paxson, and W. Stevens. TCP congestion control. *RFC 2581*, 1999.
2. P. Barford, J. Ullrich, and V. Yegneswaran. Internet intrusions: global characteristics and prevalence. *Proceedings of the 2003 ACM SIGMETRICS conference*, pages 138–147, 2003.
3. G. Combs. Wireshark - packet analyzer. *http://www.wireshark.org/*, Accessed April 2010.
4. B. Coskun and N. Memon. Online Sketching of Network Flows for Real-Time Stepping-Stone Detection. In *Proceedings of the 2009 Annual Computer Security Applications Conference*, pages 473–483. IEEE Computer Society, 2009.

5. H. Etoh and K. Yoda. Finding a connection chain for tracing intruders. *Proceedings of the 6th European Symposium on Research in Computer Security*, pages 191–205, Springer Verlag, 2000.
6. G. Giacobbi. The GNU netcat project. *http://netcat.sourceforge.net/*, Accessed April 2010.
7. S. Lee and C. Shields. Tracing the source of network attack: A technical, legal and societal problem. *Proceedings of the 2001 IEEE Workshop on Information Assurance and Security.*, pages 239–246, 2001.
8. N. Mathewson, R. Dingledine, and P. Syverson. Tor: The second generation onion router. *In Proceedings of the 13th USENIX Security Symposium*, pages 303–320, 2004.
9. P. McKenney, D. Lee, and B. Denny. Traffic generator tool. *http://www.postel.org/tg/*, Accessed April 2010.
10. J. Nagle. Congestion control in IP/TCP internetworks. *RFC 896.*, January, 1984.
11. V. Paxson and Y. Zhang. Detecting stepping stones. *Proceedings of the 9th USENIX Security Symposium*, pages 171–184, 2000.
12. D. Reeves and X. Wang. Robust correlation of encrypted attack traffic through stepping stones by manipulation of interpacket delays. *Proceedings of the 10th ACM conference on Computer and Communication security*, pages 20–29, 2003.
13. J. Riden. Know your enemy lite: Proxy threats - socks v666. *Honeynet Project, http://www.honeynet.org/ papers/proxy*, August 2008.
14. M. Roesch. Daemonlogger, packet logger. *http://www.snort.org/users/roesch/Site/Daemonlogger/Daemonlogger.htm*, Accessed April 2010.
15. S. Staniford-Chen and L. T. Heberlein. Holding intruders accountable on the internet. In *SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy*, page 39, Washington, DC, USA, 1995. IEEE Computer Society.
16. G. Stewart. iprelay - a user-space bandwidth shaping TCP proxy daemon. *http://manpages.ubuntu.com/manpages/hardy/man1/iprelay.1.html*, Accessed April 2010.
17. R. Walpole, R. Myers, S. Myers, and K. Yee. *Probability and statistics for engineers and scientists*. Macmillan New York, 2007.
18. L. Zhang, A. Persaud, Y. Guan, and A. Johnson. Stepping stone attack attribution in non-cooperative IP networks. In *Proc. of the 25th IEEE International Performance Computing and Communication Conference (IPCCC2006)*, Washington, DC, USA, April, 2006. IEEE Computer Society.