Stepping Stone Detection at The Server Side

Ruei-Min Lin, Yi-Chun Chou, and Kuan-Ta Chen
Institute of Information Science, Academia Sinica
ray@iis.sinica.edu.tw, ploymikie@iis.sinica.edu.tw, swc@iis.sinica.edu.tw

***Abstract*—Proxy server was originally invented to enhance the performance of web browsing; however, it has been commonly used to perform online crime and malicious activities without being traced. Nevertheless, there is no general method available for detecting the use of stepping stones from the server's perspective.**

In this paper, based on Nagle's algorithm, we propose a server-based scheme to detect whether a host that establishes a TCP connection to the server is a stepping stone or not. Via Internet experiments on the PlanetLab, we show that our scheme achieves an average of 92% detection rate whenever our scheme applies. We believe the scheme, as a strong complement to current methods, can secure critical Internet services from being jeopardized by anonymous attacks.

I. Introduction

Proxy server was originally invented to enhance the performance of web browsing [3, 5]; however, today, it has been commonly used to meet the need of anonymous surfing [9]. By using a proxy server as a *stepping stone* [4, 14], users can anonymously surf the Internet without revealing their own IP addresses. Besides, Internet users may be "forced" to use stepping stones in order to access Internet services that are blocked by their organizations or service providers. In this way, users' connections look as if they are targeting proxy servers rather than the blocked services, and therefore address-based censorship mechanisms would fail. FreeGate [1], as an example, is one of the popular software used to penetrate the GFW (Great Firewall) with the help of a large number of proxy servers.

While Internet surfing over proxy is an effective way to protect users' anonymity and liberty of speech, like a double-sided sword, it may as well raise security problems. Representive examples include the following:

- Client usage becomes no longer accountable because the clients' identity from the server's perspective is not trustworthy. Normally, a server identifies a client by using its IP address. However, as open proxies [13] are prevalent today, a regular Internet user can easily access an Internet service via an unaccountable open proxy, not to mention malicious users. Therefore, a server can no longer assure whether the IP address associated with a connection is actually the address of a client or that of a stepping stone.
- The usage of (open) proxy is usually associated with botnets [2, 15], which have become a common infrastructure for cyber threats and online crime [6]. On one hand, bots are often offered or sold as stepping stones to anyone who have the needs to do anonymously browsing or malicious activities without being traced. On the other hand, the use of stepping stones significantly increases the difficulty to trace the originator of malicious attacks, no matter the stepping stones are provided by bots or regular open proxies.

The major challenge to the above problems lies in *the lack of the capability to unambiguously identify the originator of a request*. When a server receives a request (e.g., a HTTP request) from a host, there is no systematic way to determine whether the host itself generates the request, or it is relaying the request for another host. To the best of our knowledge, there is no general method available for detecting the use of stepping stones from the server's perspective.

In this paper, based on Nagle's algorithm [12], *we propose a server-based scheme to detect whether a host that establishes a TCP connection to a server is a stepping stone or not*. For brevity, we call a host a "client" if an application (e.g., a web browser, a mail agent, or a FTP client) is running on it and asking for certain services from a server. Also, we call a host a "proxy (node)" if it accepts requests from a client and forwards them to another host, and call a host a "server" if it provides an Internet service to clients. Following the terminology, when a server accepts a request from a host, our proposed scheme will be able to determine whether the host is a proxy or a client.

Nagle's algorithm is built in most of, if not all, TCP implementations today. It is designed to reduce the number of packets generated by a TCP sender, with a principle to combine small packets into a larger packet while maintaining the introduced latency under a moderate degree. We find that because Nagle's algorithm changes the traffic patterns of TCP packet streams, *the traffic generated by a client would have different patterns compared with the traffic relayed by a proxy*. Specifically, our scheme can detect the presence of a stepping stone by observing the interarrival times and payload sizes of the packets arriving at a server. We use both controlled and Internet experiments to verify the effectiveness of the proposed scheme. The Internet experiment results show that our scheme achieves an average of 92% detection rate that whether a host is a stepping stone or not in the scenarios that our scheme applies[1]. The detection rate is still as high as 91% if we disregard the validity of the assumption of the detection scheme. We acknowledge that the proposed scheme may not detect the presence of stepping stones in certain cases. However, we believe that it is a good starting point to give

This work was supported in part by the National Science Council under the grant NSC99-2221-E-001-015.

[1]According to our sampling result on PlanetLab, the proposed scheme applies in around 99% of the cases (c.f. Section VI).

Internet servers that provide critical services the capability to refuse anonymous (i.e., unaccountable) access whenever appropricate. Meanwhile, we consider the proposed scheme a strong complement to existing methods (c.f., Section II) in enhancing the servers' defense against anonymous attacks.

The remainder of this paper is organized as follows. Section II describes related works and Section III gives a recap of Nagle's algorithm. In Section IV, we elaborate how Nagle's algorithm affects traffic dynamics in the presence of an intermediary proxy node. We present our proposed algorithm in Section V. In Section VI, we evaluate the performance of the proposed scheme with controlled and Internet experiments. We discuss several potential attacks against our method and corresponding solutions in Section VII. Finally, Section VIII draws our conclusion.

II. Related Work

There has been a strong demand from web masters to know whether a host is connecting to their web servers directly or indirectly (i.e., via a proxy node). However, there is no general ready-to-use method available that can settle this problem. In this section, we discuss attempts for tackling this problem from the practitioners and the academia respectively.

A commonly seen "solution" to this problem one can find on the Internet is to check the header variables in a HTTP request. If a HTTP request contains headers like `X-Forwarded-For` or `Via`, the request should have been relayed by a web proxy. This solution, however, does not work for anonymous proxies, as the latter can simply not add such variables in all the HTTP requests they forward. One another solution from the practitioners' community is to let the client create a direct TCP connection to the server, which could be done by calling the socket API in a Java virtual machine. The reason is that some web browsers do not force the TCP connections from the embedded Java virtual machine following the same proxy configuration used by the browser. Thus, if a direct TCP connection is made, the server can figure out whether a proxy is used by comparing the source IP addresses of HTTP requests and that of the direct connection. However, this approach is not general , as sophisticated attacks can easily evade this detection by forcing all outgoing connections go through a proxy node before reaching the server.

While a number of earlier research from the academia also work on the detection of stepping stones [4, 14], the problem formulation is quite different. In previous works, the problem is to detect whether a host is being used as a stepping stone or not. On the other hand, in this work, we aim to detect whether a TCP connection is initiated by a stepping stone or not, from an Internet server's perspective.

Another line of research focuses on the anonymity via the anonymous routing networks, such as `Tor` [8]. In practice, malicious users may use a single proxy node, either an open proxy or a bot-hosted proxy, to launch attacks rather than use a sophisticated anonymous network such as `Tor`. Among this line of works, Hopper et al [10] proposed a latency-based attack to detect the location of a client behind the `Tor` network, based on the network latency between the client and the entry node of the `Tor` network. This approach can be reformulated to detect the use of stepping stones from the server side by contrasting the network and application-layer latencies, because the network latency denotes the distance between the server and the proxy, while the application-layer latency denotes the distance between the server and the client, as depicted in Figure 1. However, how to measure the application-layer latency reliably is challenging because the measurement can be contaminated by having the proxy (partially) simulate applications. For example, in [10], Hopper et al proposed to measure the application-layer latency based on the time difference between the proxy receives a HTML page and the client processes the HTML page. However, a malicious user can jeopardize the measurement by making the proxy parse the HTML page and issue corresponding HTTP requests on behalf of the client. By so doing, the measured network and application-layer latency will be similar. Therefore, Hopper et al's method fails to know whether a client is behind the host connecting to the server.

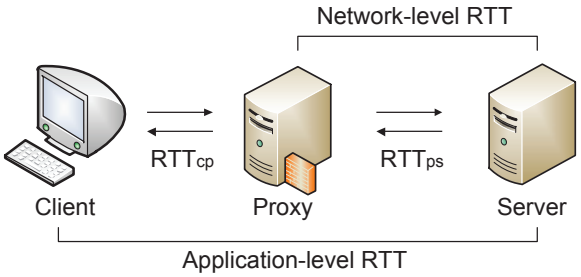
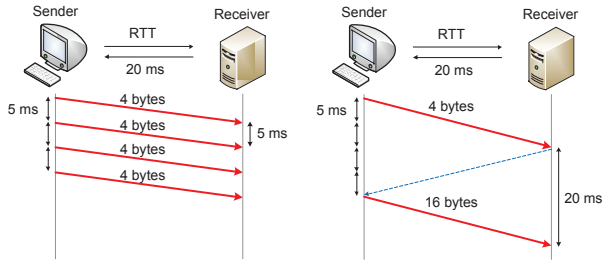


Fig. 1. Stepping stone detection based on the difference between network and application-layer round-trip time (RTT)

III. Recap of Nagle's Algorithm

Nagle's algorithm [12] is designed to reduce the header overhead of TCP packets especially for applications which may generate small packets. To do so, Nagle's algorithm buffers small outgoing messages whenever appropriate. More specifically, as long as there are unacknowledged data in flight, a sender will wait until it has a full packet's worth of output or all the data transmitted before are acknowledged by the receiver.

To elaborate how Nagle's algorithm affects the patterns of outgoing packets, we now consider a sender-receiver pair, as shown in Figure 2(a). We assume that the sender application transmits small packets (each with a 4-byte payload) to the receiver at a regular time interval (5 ms). As Figure 2(a) indicates, without applying Nagle's algorithm, the inter-arrival times (IAT) of consecutive packets are retained as the data generation intervals at the sender application. On the other hand, if Nagle's algorithm is enabled, as shown in Figure 2(b), it will buffer data and form a larger packet until old data are acknowledged. Therefore, in most cases, the IAT of generated packets will be close to the round-trip time (RTT) between the sender and the receiver.



(a) Nagle's algorithm disabled (b) Nagle's algorithm enabled

Fig. 2. The effect of Nagle's algorithm on traffic patterns

IV. Behavioral Analysis

In this section, we elaborate how Nagle's algorithm and the distance of the proxy from the client and the server will change the patterns of the traffic sent by a client toward the server.

We assume that Nagle's algorithm is enabled on the client[2]. We divide all proxy-present scenarios into 4 cases, which are classified by two factors, as shown in Figure 1. The first factor is whether Nagle's algorithm is enabled on the proxy node, while the second one is whether the RTT between the client and the proxy (i.e., RTT_{cp}) is longer than that between the proxy and the server (i.e., RTT_{ps}). In following, we assume that the client sends out 4-byte messages every 5 ms to the server via the proxy and discuss the patterns of the packets arriving at the server in each of cases:

1) Nagle-disabled proxy with $RTT_{cp} < RTT_{ps}$;
2) Nagle-disabled proxy with $RTT_{cp} > RTT_{ps}$;
3) Nagle-enabled proxy with $RTT_{cp} < RTT_{ps}$;
4) Nagle-enabled proxy with $RTT_{cp} > RTT_{ps}$.

For brevity, we denote the inter-arrival times of the packets arriving at the server as IAT_s and the payload sizes of those packets as PS_s.

- **Case 1:** We first begin with the case that the proxy server disables Nagle's algorithm and RTT_{cp} is less than RTT_{ps}. As shown in Figure 3(a), the proxy server acts as a simple repeater in this case. In other words, it replicates exactly every packet generated by the client. Because of Nagle's algorithm at the client, the inter-arrival times of the packets sent out by the client will be roughly RTT_{cp}. Therefore, IAT_s will also be roughly equal to RTT_{cp}.
- **Case 2:** This case is similar to case 1 except that RTT_{cp} is greater than RTT_{ps}. In this case, IAT_s is also roughly equal to RTT_{ps}, as exemplified in Figure 3(b).
- **Case 3:** In this case, the proxy server enables Nagle's algorithm, and RTT_{cp} is less than RTT_{ps}. This case is especially complicated because a *two-phase buffering effect* will occur, as shown in Figure 3(c). Simply put, small data chunks are aggregated into one lager packet at the client in the first phase; later, when the packets arrive at the proxy, they may be again buffered and aggregated into an even larger packet because of Nagle's algorithm at the proxy server.

  How much data are aggregated in each of the two phases depends on RTT_{cp} and RTT_{ps} respectively. In principle, the client sends out one packet every RTT_{cp}, and the proxy sends out one packet every RTT_{ps}. Unless RTT_{ps} is exactly a *multiple* of RTT_{cp}, every time when the proxy is allowed to send out a packet (given the restriction of Nagle's algorithm), the data amount it has received from the client may be different, depending on the relative ratio of RTT_{cp} and RTT_{ps}, as exemplified in Figure 3(c). Because of the payload of the packets sent out by the proxy is always multiples of those generated by the client, it likely varies, but follows a certain pattern. As a result, the payload sizes of the packets arriving at the server, PS_s, tend to follow a *n-modal distribution*.

  Here we use an example, which was collected during an Internet experiment shown in Section VI, to demonstrate the n-modal property. As shown in Figure 5, the payload sizes of the packets arriving at the server are bi-modally distributed and cluster around 120 and 180 bytes respectively.
- **Case 4:** This case is similar to case 3 except that RTT_{cp} is greater than RTT_{ps}, as shown in Figure 3(d). In this case, because the proxy can send out packets in a faster pace than the client, the two-phase buffer effect in case 3 is not present. Therefore, IAT_s will be close to RTT_{cp}.

In sum, we show that if a proxy relays TCP traffic for a client towards a server, the patterns of the packets arriving at the server will reveal the presence of the proxy node. Also the traffic patterns may be different depending on whether Nagle's algorithm is enabled at the proxy and the relative magnitude of RTT_{cp} and RTT_{ps}. Our elaborations show that IAT_s and PS_s will be indicators for detecting the presence of the proxy node between a client and a server.

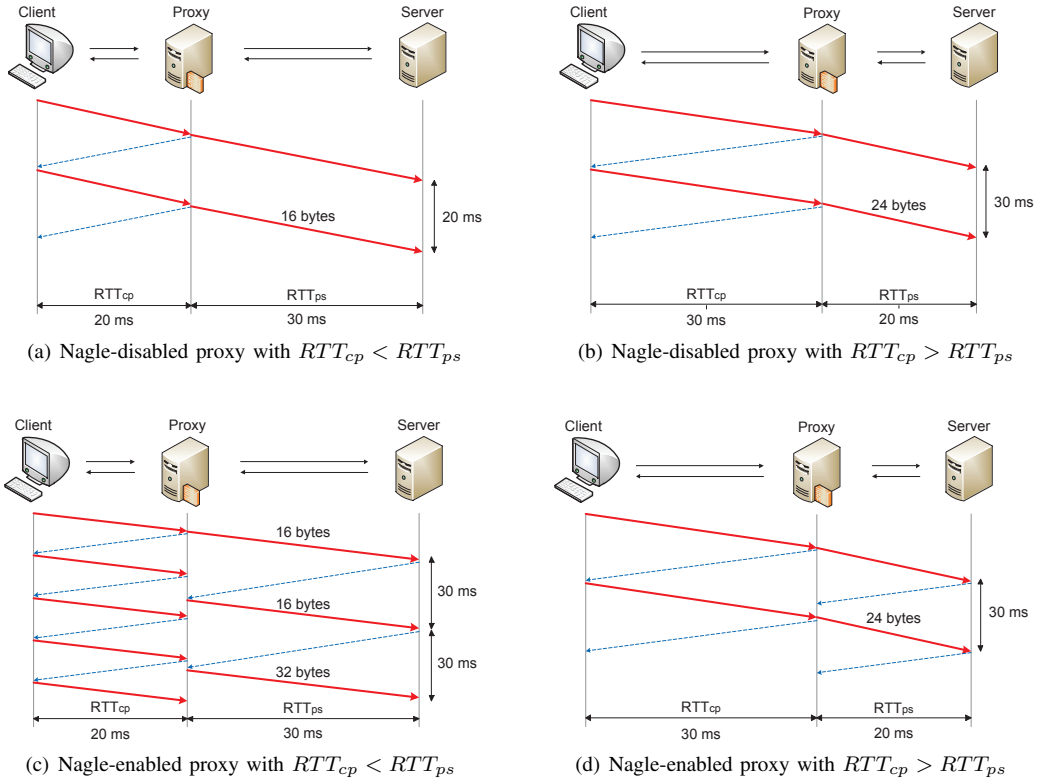
[2]We will discuss the effectiveness of our algorithm in the case if Nagle's algorithm is disabled on the client in Section VII.

Fig. 4. The scenario of our detection scheme: A host is connecting to a server with round-trip times RTT_{us}, and we expect the scheme to tell us whether the host is a stepping stone (proxy) or a client.

V. Proposed Scheme

In this section, we present our detection scheme that detects whether a host is a client or a proxy server.

To start the detection process, we require that the client application generates a number of small data chunks regularly toward the server. Practically, the data generation can be implemented in an application-specific way. For example, if the client application is a web browser, we can feed it with a HTML page containing a number of small images, as Hopper et al did in their work [10]. If the client application is a telnet client, we can write a script so that the client would repeatedly generates key press events within a short period. At the same time, for every established TCP connection, the server monitors the packet arrival processes, extracts IAT_s and PS_s, and infers RTT_{us} based on the TCP sequence numbers and acknowledgements [11] , as shown in Figure 4 .



(a) Nagle-disabled proxy with $RTT_{cp} < RTT_{ps}$

(b) Nagle-disabled proxy with $RTT_{cp} > RTT_{ps}$

(c) Nagle-enabled proxy with $RTT_{cp} < RTT_{ps}$

(d) Nagle-enabled proxy with $RTT_{cp} > RTT_{ps}$

Fig. 3. Effect of Nagle's algorithm on the traffic patterns when a proxy is used to relay TCP packets

Our detection algorithm applies to each TCP connection that the server has collected a sufficient number of IAT_s and PS_s samples. First, it compares IAT_s and IAT_{gen}, which is the interval that the client application generates small data chunks. If $IAT_s \approx IAT_{gen}$, it indicates that Nagle's algorithm is not enabled at the client, which contradicts to our pre-requirement; we defer this issue to Section VII. We then compute whether $IAT_s \approx RTT_{us}$ is valid; if the statement is not held, it indicates that the remote peer of this particular TCP connection is a proxy based on the behavioral analysis of cases 1, 2, and 4 in Section IV. We test if IAT_s and RTT_{us} possess statistically equivalent medians by applying Wilcoxon-Mann-Whitney test with a significance level of 0.05. In case $IAT_s \approx RTT_{us}$, we further determine if the case 3 (i.e., Nagle-enabled proxy with $RTT_{cp} < RTT_{ps}$) applies by checking if PS_s holds the n-modal property. We adopt a rather relaxed algorithm here: Instead of checking whether PS_s follows a n-modal distribution, we check if PS_s consistently spreads over a certain value (i.e., a single-modal distribution) over time. We first normalize PS_s into PN_s by dividing PS_s by the size of the data chunk generated by the client application (4 bytes in our case). By so doing, PN_s stands for the number of chunks delivered by a packet arriving at the server. We then divide the PN_s sequence into a number of segments, PN_{s1}, PN_{s2}, ..., PN_{sk}, with a window size n, which is set to 10 in our experiments. If the number of chunks in a segment PN_{si} is clustered around a value, we consider that the segment is single-modal. Finally, if the ratio of single-modally distributed segments is lower than a threshold, we consider the remote peer of the TCP connection as a proxy node, or otherwise it is deemed a client. We summarize our detection algorithm in pseudo code in Table III.

VI. Performance Evaluation

In this section, we verify the effectiveness of our proposed detection scheme by using both controlled and Internet experiments.

A. Controlled Experiments

We first validate our scheme in a controlled environment. In the experiments, we use three computers, a client, a proxy, and a server, all installed with Ubuntu Linux 2.6.32 and connecting to a gigabit Ethernet switch. We switch between two configurations to emulate the conditions with and without the proxy as the stepping stone. In the C-S configuration, the client establishes a TCP connection to the server directly; in contrast, in the C-P-S configuration, we use the program `Socksify` to redirect the client's TCP connection to the server via the proxy, where `Dante` is installed as a SOCKS 5 server.

In each experiment run, the client sends out 1,000 4-byte data chunks every 5 ms. We implement the proposed scheme on the server so that the server infers RTT_{us}, IAT_s, and PS_s based on the received packets and determines whether the proxy is involved to relay the packets. In order to simulate different RTT_{cp} and RTT_{ps}, we use `dummynet` to inject latency before sending packets out from the client and the proxy respectively.

TABLE I
DETECTION PERFORMANCE IN CONTROLLED EXPERIMENTS

Configuration		$RTT_{cp} < RTT_{ps}$	$RTT_{cp} > RTT_{ps}$
C-S		93.9%	
C-P-S	Nagle-enabled proxy	83.0%	97.8%
	Nagle-disabled proxy	99.1%	100.0%

TABLE II
DETECTION PERFORMANCE IN INTERNET EXPERIMENTS

Configuration		$RTT_{cp} < RTT_{ps}$	$RTT_{cp} \approx RTT_{ps}$	$RTT_{cp} > RTT_{ps}$
C-S		94.0% (18541)		
C-P-S	Nagle-enabled proxy	83.8% (2987)	32.5% (502)	98.7% (3155)
	Nagle-disabled proxy	98.9% (3241)	14.6% (555)	99.2% (3144)

TABLE III
THE PROPOSED DETECTION ALGORITHM

procedure *Stepping_Stone_Detection*

if $IAT_s \approx IAT_{gen}$ **then**
  Aborts because Nagle's algorithm is disabled at the client
else
  if not $IAT_s \approx RTT_{us}$ **then**
    A stepping stone is detected
  else
    if PS_s is n-modally distributed **then**
      A stepping stone is detected
    else
      No stepping stone is detected
    end if
  end if
end if

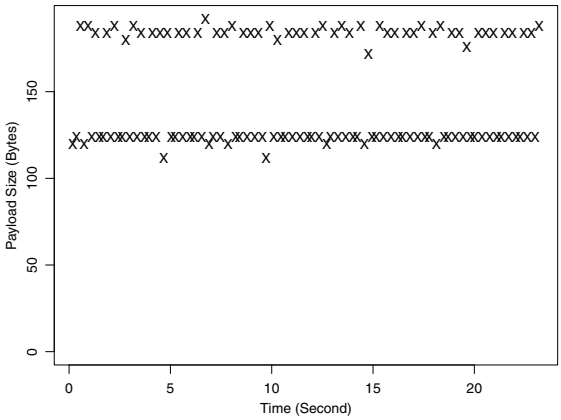


Fig. 5. Payload size distribution

We have run the experiments repeatedly in each of the two configurations and in each of the 4 cases discussed in Section IV. In each run, RTT_{cp} and RTT_{ps} were uniformly chosen from the interval $(5, 200)$ ms. Overall, we have iterated 100 runs for the C-S configuration, and at least 50 runs for each of the 4 cases with the C-P-S configuration. The average accuracy with each setting is summarized in Table I. From the table, we can see that for the C-S configuration, our algorithm can correctly decide that no proxy is used in 93% of the runs. For the C-P-S configuration, the average accuracy is 91%, with most of the wrong decisions were made in the case 3 scenarios. The reason is that occasionally RTT_{ps} is close to a multiple of RTT_{cp}, so we do not observe a n-modally distributed PS_s. Another reason is that PS_s was actually n-modally distributed, but the number of samples was not enough to enable a robust detection.

B. Internet Experiments

We further conduct Internet experiments on PlanetLab [7] in order to verify whether our proposed scheme still works with the presence of uncontrollable cross-traffic on the Internet. For the experiments, We have recruited 304 nodes on the PlanetLab. We randomly chose two nodes as the client and the server for each run in the C-S configuration and three nodes for each run in the C-P-S configuration. Whether Nagle's algorithm on the proxy is enabled (in the C-P-S configuration) is randomly decided. In contrast, RTT_{cp} and RTT_{ps} were naturally decided as we randomly chose three nodes as the client, the proxy, and the server in each run.

We conducted more than 10,000 runs with the C-S configuration and more than 2,000 runs for each of the 4 cases with the C-P-S configuration. The average accuracy of our scheme as well as the number of runs with each setting is reported in Table II. Note that because we let the client application generate data chunks with a 5-ms interval, our scheme is unable to detect the presence of proxy if $RTT_{cp} < 5$ ms; thus we removed runs with $RTT_{cp} < 5$ ms, which account less than 1% of the runs.

From Table II, we can see that the detection accuracy is all above 90% except two conditions: 1) $RTT_{ps} \approx RTT_{cp}$ when a proxy is present and 2) $RTT_{cp} < RTT_{ps}$ when a proxy is present and Nagle's algorithm is enabled at the proxy node. The reason responsible for the low accuracy in the first condition is because our scheme relies on the fact that $IAT_s \approx RTT_{cp} \neq RTT_{ps}$. When RTT_{cp} happens to be approximately close to RTT_{ps}, our scheme will not consider the remote peer as a proxy node (cf. Section V). Fortunately, this condition does not occur frequently and accounts only 7% of the runs, which should be roughly the proportion of such cases in real-life scenarios. The second condition corresponds to the case 3 in Section IV, where the two-phase buffering effect is present. We find that the lower accuracy is partly due to the fact that Internet queueing delays may lead to unstable round-trip times between the client and the server;

thus, PS_s cannot be consistently variable even without the presence of the proxy, and misguides the decision of the n-modality detector. In addition, the interplay of the delay acknowledgement mechanism and Nagle's algorithm can also make PS_s less regular and more like n-modally distributed.

To sum up, our detection scheme achieves 92% accuracy in the 99% of the runs that it applies. We believe that it can serve as a strong complement to other stepping stone detection algorithms, such as Hopper et al [10], to complete the solution.

VII. Security Analysis

In this section, we discuss potential countermeasures from malicious attackers to jeopardize the proposed detection scheme.

1) **Nagle's algorithm is disabled on both the client and the proxy node.** The proposed detection scheme requires that Nagle's algorithm is enabled at the client. Therefore, an intuitive countermeasure against our detection is to turn off Nagle's algorithm on the client. We can easily detect such countermeasures if the proxy also turns off Nagle's algorithm as IAT_s would be close to IAT_{gen} in this case. Therefore, we can determine this countermeasure by observing if $IAT_s \approx IAT_{gen}$.
2) **Nagle's algorithm is disabled on both the client and the proxy, but the client simulates Nagle's algorithm at the application layer.** Following the first countermeasure, a malicious client may simulate the effect of Nagle's algorithm as if it directly connects to the server. In this way, the modified client application can purposely send out packets with interval time RTT_{ps} and therefore IAT_s will be close to RTT_{ps}. To detect such countermeasure, we can block the TCP acknowledgement occasionally during the detection process. The client which follows Nagle's algorithm will have to buffer packets and wait until an acknowledgement returns. Because a client implementing this countermeasure will have to release a packet every RTT_{ps}, if we block an acknowledgement longer than RTT_{ps} and still see new packets arriving at the server, the client is likely facilitating this countermeasure to evade the proposed detection scheme. The client cannot release data packets upon the receipt of acknowledgements because acknowledgements come back after $RTT_{cp}+RTT_{ps}$ but the next packet needs to be released after RTT_{ps}.
3) **Nagle's algorithm is disabled on the client but enabled on the proxy node.** Another countermeasure to bypass the proposed detection scheme is to disable Nagle's algorithm at the client but enable it at the proxy node. In this case, the client and the proxy together can be seen as a virtual node with a long internal processing delay. Therefore, it is not an easy task to detect such countermeasures. One solution is to complement our scheme with additional information, such as Hopper et al's distance-based scheme [10], to detect the usage of stepping stones.

VIII. Conclusion

In this paper, we have proposed a novel scheme that enables an Internet server to detect whether a remote peer of a TCP connection is a stepping stone or not. We have shown that because Nagle's algorithm changes the traffic patterns of TCP packet streams, the traffic generated by a client would have different patterns compared with the traffic relayed by a proxy. Based on this property, our scheme detects the presence of a stepping stone by observing the interarrival times and payload sizes of the packets arriving at a server. We consider this work a starting point to give critical Internet servers the power to refuse anonymous accesses whenever necessary.

References

[1] [Online]. Available: http://en.wikipedia.org/wiki/Freegate
[2] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," ser. IMC '06. ACM, 2006.
[3] M. F. Arlitt, R. Friedrich, and T. Jin, "Performance evaluation of web proxy cache replacement policies," in *Proceedings of the 10th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, ser. TOOLS '98. Springer-Verlag, 1998.
[4] A. Blum, D. Song, and S. Venkataraman, "Detection of interactive stepping stones: Algorithms and confidence bounds," in *in Conference of Recent Advance in Intrusion Detection (RAID), (Sophia Antipolis, French Riviera*. Springer, 2004, pp. 258–277.
[5] R. Cáceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, "Web proxy caching: the devil is in the details," *SIGMETRICS Perform. Eval. Rev.*, vol. 26, December 1998.
[6] N. I. Cert/cc and A. H. Cert/cc, "Botnets as a vehicle for online crime," 2005.
[7] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman, "Planetlab: an overlay testbed for broad-coverage services," *SIGCOMM Comput. Commun. Rev.*, July 2003.
[8] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: the second-generation onion router," in *Proceedings of the 13th conference on USENIX Security Symposium - Volume 13*, ser. SSYM'04. USENIX Association, 2004.
[9] E. Gabber, P. B. Gibbons, Y. Matias, and A. J. Mayer.
[10] N. Hopper, E. Y. Vasserman, and E. Chan-tin, "How much anonymity does network latency leak," in *In CCS 07: Proceedings of the 14th ACM conference on Computer and communications security. ACM*, 2007.
[11] H. Jiang and C. Dovrolis, "Passive estimation of TCP round-trip times," *SIGCOMM Comput. Commun. Rev.*, July 2002.
[12] J. C. Mogul and G. Minshall, "Rethinking the tcp nagle algorithm," *SIGCOMM Comput. Commun. Rev.*, January 2001.
[13] V. S. Pai, L. Wang, K. Park, R. Pang, and L. Peterson, "The dark side of the web: an open proxy's view," *SIGCOMM Comput. Commun. Rev.*, January 2004.
[14] Y. Zhang and V. Paxson, "Detecting stepping stones," in *In Proceedings of the 9th USENIX Security Symposium*, 2000, pp. 171–184.
[15] Z. Zhu, G. Lu, Y. Chen, Z. J. Fu, P. Roberts, and K. Han, "Botnet research survey," in *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference*, Washington, DC, USA, 2008.