Off-Path Round Trip Time Measurement via TCP/IP Side Channels

Geoffrey Alexander
Department of Computer Science
University of New Mexico, USA
alexandg@cs.unm.edu

Jedidiah R. Crandall
Department of Computer Science
University of New Mexico, USA
crandall@cs.unm.edu

***Abstract*—We present a novel technique for estimating the round trip time network latency between two off-path end hosts. That is, given two arbitrary machines, A and B, on the Internet, our technique measures the round trip time from A to B. We take advantage of information side-channels that are present in the TCP/IP network stack of modern Linux kernels to infer information about off-path routes. Compared to previous tools, ours does not require additional resources, machines, or require additional protocols beyond TCP. The only requirements are that one end host have an open port and be running a modern Linux kernel and that the other end host responds to unsolicited SYN-ACK packets with a RST packet.**

We evaluate our technique "in the wild" and compare our off-path estimates to on-path measurements. Our experiments show that our technique provides accurate, real-time estimates of off-path network latency. In over 80% of measurements our technique provides off-path round trip time measurements within 20% of the actual round trip time. We also discuss possible causes of errors that impact the accuracy of our measurements.

I. Introduction

Round trip time measurements represent a fundamental network metric that can be used in a wide range of applications, including: determining the fastest route, closest server, and some forms of IP geolocation. While direct measurement of round trip time is a simple task it can only provide information about round trip times to a single end host. This places limitations on the extent to which large scale measurements can be performed, as it requires access to a large number of well distributed end hosts to provide sufficient network coverage.

In delay based IP geolocation techniques, such as those described by Gueye *et al.* [1] and GeoPing [2] , the round trip time is often used as a proxy for geographic distance. By measuring the round trip time between the target machine and a series of "landmark" machines whose geographic positions are known, it is possible to provide an estimate of the target machine's geographic location. Usually these systems require access to the "landmark" servers to perform round trip time measurements, which limits their ability to choose a sufficient number of appropriate "landmarks" to allow for accurate geolocation for all possible end hosts.

In this paper, we introduce a novel technique for off-path round trip time estimation, based on the use of information side-channels in the modern Linux TCP/IP network stack, to infer the round trip time between two arbitrary end hosts. Our technique requires the use of no machines beyond the measurement machine and the chosen end hosts and does not require the use of any protocols beyond standard TCP. The only requirements are that one end host be a Linux machine, running a kernel of version 2.6 or higher, with an open port, and that the second end host respond to unsolicited SYN-ACK packets with RST packets.

When clients want to connect to a server over TCP they first use a three-way handshake to initialize the connection. Any connections that have yet to complete this three-way handshake are often referred to as "half-open" connections. Modern network stacks will only store such connections for a limited amount of time, to avoid these entries exhausting the allocated system resources. Once this time limit has passed these "half-open" connections are removed from the backlog to make space for new connection attempts. By manipulating the number of "half-open" connections in a given SYN-backlog, it is possible to use the behavior of the backlog to infer useful information about connections present in the backlog, including round trip time.

The contributions of this paper are as follows:

- First, we describe a method for using information side-channels in the Linux TCP/IP network stack to infer the round trip time of off-path routes.
- We design and implement a measurement technique for using these side-channels to measure off-path round trip times between arbitrary Internet end hosts.
- We provide a detailed analysis and evaluation of the technique's accuracy across a range of routes and round trip time values. We discuss possible sources of error and the effect these have on the accuracy of our measurements.

The rest of the paper is structured as follows: In Section II we provide a brief overview of the behavior of the SYN-backlog in modern Linux kernels and discuss how this behavior leaks information that can be used to perform useful network measurements. In Section III, we describe our technique for inferring off-path round trip time in depth. We then describe our experimental setup in Section IV and discuss our results in Section V. Section VI addresses some ethical concerns. Section VII discusses related works, followed by our conclusion, in Section VIII.

II. Background

Standard TCP connections are established via a "three-way handshake" as defined in RFC793 [3]. This process consists of three TCP packets being sent between each end point of a connection, in order to establish all the necessary state required for the TCP connection to be established. In the first step the client sends a TCP SYN packet to the server. Assuming the server is accepting connections, it responds with a TCP SYN-ACK packet. The client finally responds with a TCP ACK packet and can then begin sending data.

Connections which have not completed the "three-way handshake" are stored in a data structure called the SYN-backlog. Connections stored in the backlog are considered to be "half-open" connections. The SYN-backlog plays an important role in ensuring the establishment of TCP connections. Until the "three-way handshake" is complete the TCP connection has not been fully initiated and data can not be sent between hosts. To ensure that "half-open" connections can be completed various information about the connection must be stored, so that the "three-way handshake" can be successfully completed. This information is stored in the SYN-backlog. The backlog also provides a protection mechanism against packet loss, during a TCP connection setup. If either the SYN-ACK or the ACK in the handshake is lost or dropped, then the presence of a SYN packet in the backlog allows for the retransmission of the SYN-ACK, in an attempt to allow the connection to be completed. But if the SYN packet is allowed to remain in the backlog indefinitely, then system resources will be tied up storing connection information that may never be used.

In order to prevent these "half-open" connections from taking up too many system resources, most modern network stacks only store "half-open" connections for a limited amount of time before they are removed from the backlog. In addition to storing these connections, network stacks will often resend SYN-ACK packets for which no ACK has been received. The number of retransmissions and the length of time for which a "half-open" connection remains in the SYN-backlog varies across TCP implementations.

Our technique focuses on the SYN-backlog implementation used in Linux kernels of version 2.6 and higher. The kernel maintains a backlog, for each listening socket, whose length is chosen when the socket starts listening for new connection attempts. The kernel maintains a threshold for the number of retransmission attempts that will be made for each SYN-ACK that is not ACK'd. The threshold is set to five as a default value. The Linux kernel distinguishes between "young" connections, that is connections for which there have been no SYN-ACK retransmissions, and "mature" connections, for which at least one retransmission has been sent.

The kernel attempts to ensure that at least half of the backlog is available for "young" connections. If the backlog is at least half full then the kernel will attempt to remove "mature" connections, based on the number of retransmission attempts that have been sent, until the SYN-backlog's current capacity drops below half. When attempting to remove "mature" connections the kernel will temporarily compute a new threshold value, depending on the number of "young" attempts present in the backlog. The more "young" connections that are present the larger this temporary threshold will be.

Any "mature" connections that have sent a total number of retransmissions greater than this threshold will be removed, until at least half of the backlog is free for new connection attempts. This behavior, of removing "mature" connections that have not yet retransmitted the full five times if the backlog is filled beyond half capacity, provides an information side-channel that allows backlog information state to be leaked. If a "half-open" connection does not elicit the full five retransmission attempts, then we know the backlog was filled beyond half, leaking information about the state of the backlog.

III. Technique Overview

We designed our system based on some observations about SYN-backlog behavior in Linux kernels and how this behavior changes under certain conditions. The first observation is that the ratio of "young" connections to "mature" connections can be influenced by an off-path host. If a host sends a large amount of SYN packets, each using different sequence numbers and source ports, these will each be stored as separate backlog entries. By never completing the TCP three-way handshake (i.e: never sending an ACK in response to any SYN-ACKs), it is possible to ensure that after a few seconds the backlog will contain mostly "mature" entries. If enough entries are sent then the backlog will be filled beyond the halfway mark with "mature" entries, causing the kernel to try and remove entries which have been retransmitted too many times, in order to ensure there is sufficient space in the backlog for newer "young" entries.

The second observation is that it is possible to influence the threshold value for which connection attempts are removed by the backlog when more than half the backlog is taken up by "mature" connections. Since the kernel determines this value based on the number of "young" connections, a backlog that has mostly "mature" connections and very few "young" connections will use a low threshold value when determining which connections to remove from the backlog. By filling the backlog with "half-open" connections and waiting until all these connections have received a number of retransmissions greater than the temporary threshold, we can ensure that some of these connections will be removed if the backlog is then filled past the half way point.

As previously mentioned there are a few requirements that must be met in order for the measurements to be performed. First, one of the end hosts, referred to as the "Server", must be running a modern Linux kernel and have an open TCP port. Second, that other end host, referred to as the "Client", must respond to unsolicited SYN-ACK packets with a RST packet. Round trip time measurements can then be carried out by an off-path machine, referred to as the "Measurement Machine".

Overall our technique consists of a binary search over a range of possible round trip time values. Each round of the

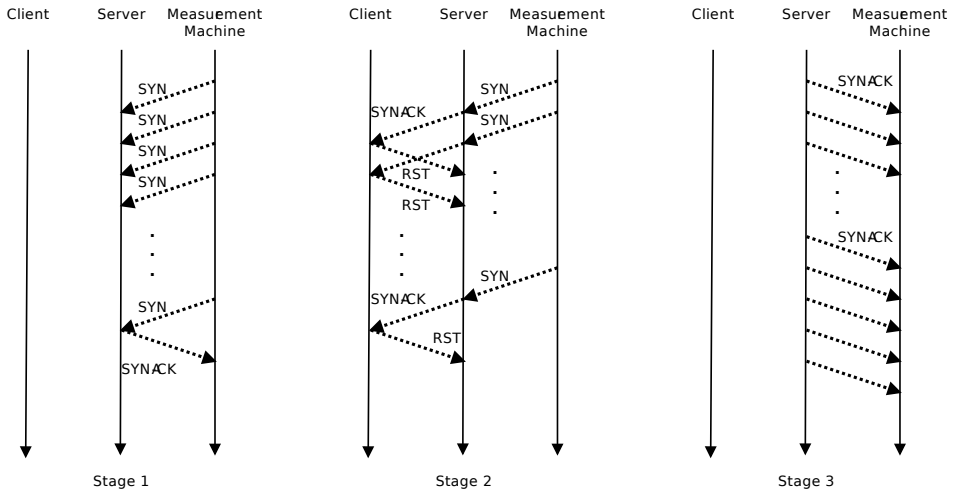

Fig. 1. Three stages of our scan. Stage 1 partially fills the SYN-backlog of the Server with SYN packets from the Measurement Machine. Stage 2 sends SYNs to the Server which appear to come from the Client. These trigger SYN-ACKs which are then reset. In Stage 3 the number of retransmissions for each SYN sent in Stage 1 is tallied and a determination is made about the chosen RTT estimate.

binary search consists of three stages, shown in Figure 1. These stages:

- Establish the necessary conditions for use of the information side-channel
- Attempt to cause a change in the state of the SYN-backlog
- Infer whether or not the desired state change occurred

Depending on whether or not the desired state change occurred, we can then determine how to adjust the upper and lower bounds used for the next iteration in the binary search.

A. Determining scan parameters

Before the round trip time between two hosts can be measure some parameters about the Server must be determined. These parameters are the size of the Server machine's SYN-backlog and the total amount of time taken for the Server to send all five SYN-ACK retransmissions to received SYN packets.

1) Determining Server SYN backlog size: To determine the Server's SYN-backlog size a technique similar to the full scan described below can be used. The Linux kernel places a minimum threshold on the size of the SYN-backlog of eight. Based on this an attempt to determine the backlog size can proceed as follows: First, set our estimated size to eight and then send half the estimated size minus 1 unique SYN packets to the Server. Second, wait until two retransmissions have been seen for each SYN packet and then send two more unique SYN packets to the Server. Next, see if any of the first round of SYN packets do not retransmit SYN-ACKs. If all SYNs continue to receive SYN-ACKs then the estimated backlog size is too small, otherwise it is too big. If the estimate is too small, double the estimate and repeat the previous steps and if the estimate is too big carry out a binary search using the estimate as an upper bound and half the estimate as the lower bound.

2) Determining Server Retransmission timing: Measuring the amount of time taken by the Server to send all five SYN-ACK retransmissions is a simple process. The Measurement Machine sends a SYN packet to the Server and then waits until it sees five retransmissions of the SYN-ACK sent in response to the initial SYN packet. In determining how long to wait between each retransmission attempt the Linux kernel waits twice the amount of time it waited on the previous retransmission. As an example if the kernel waits one second before sending a retransmission, it will wait two seconds after sending the first before sending then second, then four, eight and so on until five retransmission have been sent. After five retransmissions have been sent the kernel then waits for another interval before removing the initial SYN packet from the SYN-backlog. Given this behavior, once the fifth retransmission is seen the amount of time between the first SYN-ACK's arrival and the fifth retransmission provides the Server retransmission timing, whose use is described below.

B. Scanning the target machine

Before starting each binary search iteration, a new round trip time estimate ($eRTT$) is chosen as the midpoint between an upper and lower bound determined for the actual round trip time. Each iteration will attempt to determine if the actual round trip time is greater than, or less than the chosen estimate. In practice we use 0.0 as a starting lower bound and the time interval between the first SYN-ACK retransmission as the upper bound. For most Linux systems this time interval is one second. However, to measure longer round trip times a larger upper bound should be chosen.

In the first stage of our technique, the Measurement Machine sends $\frac{n}{2} - 7$ SYN packets to the Server to partially fill

the SYN-backlog, where n is the size of the SYN-backlog for the open socket on the Server. This amount of SYN packets was chosen to ensure that we do not fill the backlog beyond halfway and to provide a small amount of padding space to allow other connection attempts that are not a part of the measurement to be completed without impacting our measurements. The Measurement Machine is programmed not to respond to any SYN-ACKs sent in response to these SYNs, which causes each of these SYNs to be treated as "half-open" connections by the Server. Each SYN uses a different sequence number and source port to ensure they are each treated as new, unique connection attempts. The Measurement Machine then waits until it has received 2 retransmission attempts from each SYN packet, before moving on to Stage 2.

Next, in the second stage, the Measurement Machine sends $\frac{6}{eRTT}$ SYN packets per second to the Server for ten seconds. $eRTT$ is measured in seconds. Each of these packets uses a spoofed source IP address to appear as if sent from the Client. All spoofed packets also use different sequence numbers and ports so that they are all treated as separate connection attempts. Each of the SYNs will cause the Server to send a SYN-ACK to the Client. Once the Client receives these SYN-ACKs it will determine that it sent no accompanying SYN packet and respond with a RST packet. Once these RST packets arrive at the Server they will cause the corresponding SYN packets to be removed from the backlog. After the Measurement Machine has finished sending at the determined rate for ten seconds the process moves on to Stage 3.

In the third stage, the Measurement Machine simply waits and records the number of SYN-ACK retransmissions it received for each of the SYN packets that were sent in Stage 1. Modern Linux kernels will typically make five retransmission attempts for each "half-open" connection before removing a SYN packet from the backlog. Typically these attempts will occur at 1, 2, 4, 8, and 16 second intervals after the previous SYN-ACK was sent, with the packet being removed from the backlog 32 seconds after the fifth SYN-ACK is sent. That is after the first SYN-ACK is sent the Server waits one second and if no ACK has been received the SYN-ACK is retransmitted. If after 2 seconds no ACK has been received another SYN-ACK is retransmitted and so on, until 5 retransmissions have been sent. We have also seen these 5 retransmission attempts being sent at 3, 6, 12, 24, and 48 second intervals after the previous SYN-ACK, with the kernel waiting 96 seconds after the fifth and final retransmission before the SYN is removed from the backlog. The exact timing of a given machine can be determined before any measurement attempt by sending a single SYN and determining the interval between subsequent retransmission attempts.

After the third stage, it is possible to determine if the chosen round trip time estimate is greater or less than the actual round trip time between the two end hosts. If the chosen estimate is less than the actual round trip time then the sent packets will begin to accumulate in the backlog, as new packets will arrive before previous packets are removed due to the Client sending RST. This will cause the backlog's size to exceed half of its maximum. This causes the kernel to remove "mature" entries until the backlog size is below half the maximum. Any entries that are removed will stop retransmitting SYN-ACKs. If the estimated round trip time is greater than the actual round trip time, each packet will be reset before the next arrives and the backlog will never contain enough packets to cause the kernel to try and remove any "mature" connection attempts.

We then check to see if any of the initial SYNs sent by the Measurement Machine in Stage 1 did not elicit five retransmissions. If any did not then we can infer that some SYNs were removed by the Server's kernel and our estimated round trip time is less than the actual round trip time. Otherwise, we can infer that our estimate is greater then the actual round trip time. Using this information we can continue our binary search, adjusting our round trip time estimate accordingly. Our technique then continues iterating over possible ranges of round trip times until the scan has provided an accurate bounds on the round trip time.

C. An example iteration

Consider an off-path measurement being performed between a Server, S, and a Client, C, by a Measurement Machine, M. Assume that the open socket on S has a backlog size of 256. The route between S and C has an actual round trip time of 60ms and previous iterations of the binary search have determined that the next estimate should be in between 100ms and 50ms. First, a new estimate is determined to be 75ms, the midpoint between 100 and 50. In Stage 1 M sends $\frac{256}{2} - 7 = 121$ SYN packets to S and does not respond to any SYN-ACKs sent in response. After receiving two retransmission attempts per SYN packet Stage 2 begins.

M then sends $6/0.075 = 80$ SYN packets per second for 10 seconds to S with the spoofed source IP address of C. S will respond to each SYN by sending SYN-ACKs to C. Since C did not send any SYN packets it responds to these SYN-ACKs with RST packets. Once these RSTs arrive at S the corresponding SYN packets are removed from the SYN-backlog. Based on our previous choice of packet rate, S receives 6 SYN packets for every estimated round trip time between itself and C. However, because the chosen estimate, 75ms, is greater than the actual round trip time of 60ms each SYN in this group of 6 will have already been removed before the next group of 6 arrive. Because of this the SYN-backlog will never fill beyond 127, which is less than the halfway value required to remove "mature" connections from the backlog.

Finally, in Stage 3, M will receive 5 retransmissions for each SYN that was sent in Stage 1 since the removal behavior was never triggered. M will then conclude that the chosen estimate 75ms is too large and determine the bounds for the next binary search iteration to be an upper bound of 75ms and a lower bound of 50ms.

IV. Experimental Setup

To test the effectiveness of our technique we compared our technique with actual round trip time measurements between end hosts on the Internet. We used a set of 15 PlanetLab nodes:

3 from each in North America, South America, Europe, Asia, and Australia/New Zealand, as the Server end host. This group was chosen to provide a distributed set of Server hosts, on most continents. Each PlanetLab node runs a simple server program which creates an open port, for use by our scan, with a backlog size of 256. Each PlanetLab node was running a Linux kernel of either version 2.6.X or 2.7.X, which meets our requirements for the Server end host.

These PlanetLab nodes do not provide any additional assistance to our Measurement Machine during any part of our experiment. These nodes only provide an open port, provide access to ground-truth round trip time measurements to compare our estimates to, and perform occasional traceroutes to provide information about any potential routing changes that may influence our results. As a result all our round trip time measurements are performed off-path, from the Measurement Machine's point of view. Actual round trip time values for each measurement are gathered from direct packet captures of the off-path scan being performed. **In other words, the exact same SYN-ACK packets and RST responses between the server and client are being used for both direct and indirect measurements, so that any error in our indirect measurements can be attributed to our technique and not other factors.** This provides us with the actual round trip that each off-path scan was measuring. Another important aspect of our measurements to note is that when we say we are measuring the round-trip time, what we mean is the average round trip time over the entire time of the experiment. So our ground truth is based on this average.

The Measurement Machine was a machine running Ubuntu Linux 12.04LTS with kernel 3.5.0, located in our lab, with a direct connection to the local Internet backbone to avoid any local filtering or firewalls.

Client end hosts were chosen by IP address at random from the entire publicly routable IPv4 address space. For each randomly chosen Client IP address, the Measurement Machine checked to see if the host met our requirements for a Client end host, that is that the host responded to unsolicited SYN-ACKs with a RST packet. During our experiments we noted that some machines respond with packets that had only the RST flag set while others responded with both the RST and ACK flags set. Both of these packets resulted in the desired Server SYN-backlog behavior and were treated as meeting the Client machine requirements.

Once a Client IP address was found that met our requirements, an available PlanetLab node from our set of 15 was chosen at random to act as the Server end host for our technique. The Measurement machine then carried out our round trip time measurement technique, using the two chosen end hosts. During the scan, the chosen PlanetLab node captured all traffic being sent to the open port it had created and ran periodic TCP traceroutes, to record any route changes that occurred during the course of the scan. The Measurement Machine also captured all traffic between itself and the Server. Each measurement scan took between 5 and 10 minutes to complete. During our measurements the Measurement Machine made no contact with any Client machines, beyond checking to ensure that they met our requirements.

Dataset	Within 10%	Within 20%
Overall	60.7% (68.87%)	81.33% (90.84%)
RTT $>$ 25ms	63.6% (72.1%)	83.7% (92.7%)
RTT $<$ 25ms	18.0% (16.0%)	46.1% (60.0%)
RTT $>$ 100ms	67.1% (75.9%)	87.2% (96.0%)
RTT $<$ 100ms	35.5% (39.3%)	58.06% (69.05%)
25ms$<$ RTT $<$ 100ms	43.5% (49.3%)	63.5% (72.9%)

TABLE I
PERCENT OF MEASUREMENTS WITHIN GIVEN PERCENT OF ACTUAL ROUND TRIP TIME. VALUES FOR MEASUREMENTS WITH NO PACKET LOSS ARE IN PARENTHESES.

V. RESULTS

To evaluate the accuracy of our technique we collected data over a two week period, from 14 July 2014 through 28 July 2014. During this time span we collected 616 round trip time estimates. Each scan measured both the actual round trip time and the round trip time determined by our off-path technique, as measured by the Measurement Machine. Figure 2 shows a CDF plot of the ratio between the actual round trip time, R, and the off-path estimate, E, E/R.

If our measurements were perfect then the ratio E/R would be 1.0 for each measurement estimate. Any ratio that is less than 1.0 indicates an underestimated round trip time, a ratio greater than 1.0 indicates an overestimated round trip time. Overall, we found that approximately 60% of the round trip times measured in our dataset resulted in ratios of 1.0 or lower, providing a lower bound on the round trip time in these cases. Over 60% of scans in our dataset were within 10% of the actual round trip time measured, with over 80% of our off-path measurements falling within 20% of their actual round trip times. Table I summarizes our results.

A. Effect of packet loss

One prominent source of error that affects the accuracy of our technique is packet loss between the Server and Client. In cases where there is noticeable packet loss, greater than 5%, between the Server and the Client our measurements often differ from the actual round trip time by 25% or more. These large error values are the result of packet loss which can affect the accuracy of our off-path measurements.

Consider the case where a SYN-ACK sent from the Server to the Client as a result of a spoofed SYN is lost. In this case the Client will never send a RST packet and the spoofed SYN will remain the Server's SYN-backlog longer than was expected. If enough SYN-ACKs are lost in transit the SYN-backlog will fill beyond halfway, causing the kernel to evict some "half-open" connections. Our technique measures this eviction as an indication that the current round trip time estimate is too high resulting in an incorrect measurement. A similar situation occurs if the RST sent by the Client in response to a SYN-ACK sent by the Server is lost.

Figure 3 compares our overall estimates with those that had no packet loss. For measurements with a packet loss rate of 5% or less, a lower bound is provided for 64% of paths. Over

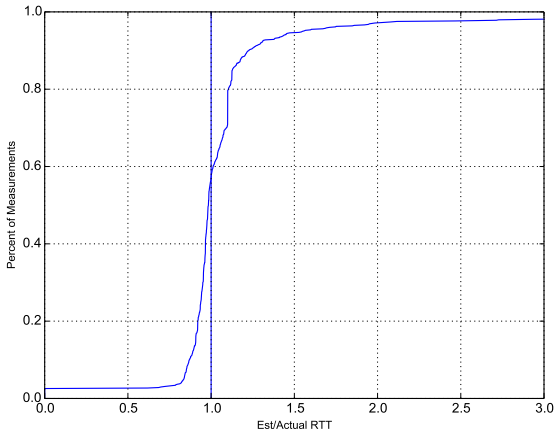

Fig. 2. Accuracy of round trip time estimates

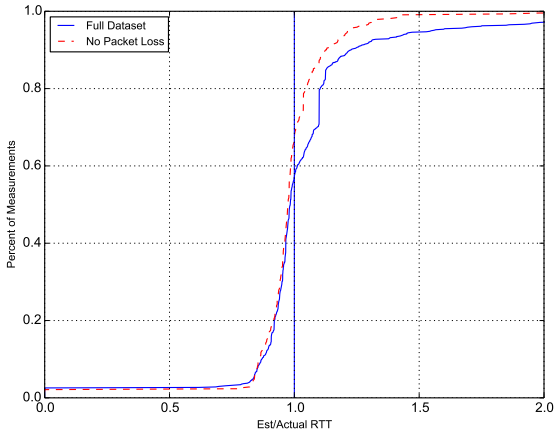

Fig. 3. Effect of packet loss on measurement accuracy

65% of measurements are within 10% of the actual round trip time, while over 85% are within 20%. When considering only those measurements with no packet loss a lower bound is found in over 65% of measurements. Over 68% of estimates are within 10% of the actual round trip time and over 90% are within 20%. Compare this to King [1], a similar system that uses recursive DNS queries between DNS servers close to the chosen end hosts, and measures round trip times to within 10% error for roughly two-thirds of chosen routes and upwards of three-quarters of routes within 20% error.

B. Accuracy when actual round trip time is small

To further explore the accuracy of our measurements, we consider situations in which specific network characteristics are present. In cases where the round trip time we attempt to measure is small we found that our technique performed quite poorly. We consider actual round trip times of 25ms or less as small. When we consider routes with actual round trip times less than 25ms our estimate represents a lower bound in 43.6% of cases. Only 18% are within 10% of the actual round trip

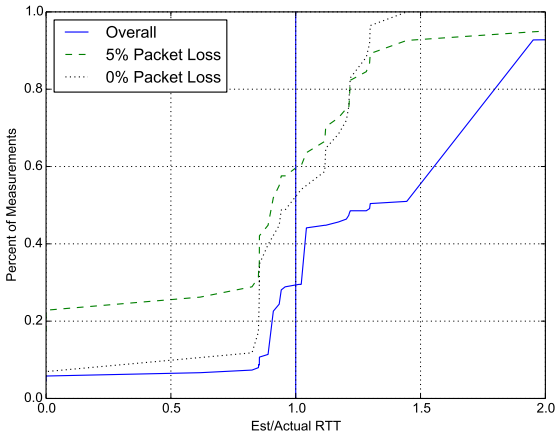

Fig. 4. Accuracy of round trip time estimates with actual RTT less than 25ms

time and 46.1% are within 20%. As we previously discussed, high packet loss rates can have a significant impact on the accuracy of our measurements. When considering small round trip time routes with no packet loss, a lower bound is found in 52% of measurements, with 16% and 60% falling within 10 and 20 percent of the actual round trip time as shown in Figure 4.

We next considered our accuracy when excluding low round trip time routes. For all routes with round trip times greater than 25ms a lower bound is found for 60.83% of routes, with 63.6% being measured within 10% of the actual round trip time, and 83.7% measured within 20%. Figure 5 shows these results. Excluding routes that experienced packet loss, 70.6% of measurement provided a lower bound, 72.1% were within 10% of the actual round trip time, and 92.7% were within 20%.

One possible explanation for this drop in accuracy is that, as the round trip time we are measuring gets smaller, the impact that small variations in the round trip time have on our measurements is magnified. When dealing with round trip times below 25ms, a difference of a few ms can affect the rate with which our technique fills the SYN-backlog. If the backlog fills faster than expected as a result of these variances this could cause packets to be removed when they should not be. This unexpected change in SYN-backlog behavior would cause incorrect binary search decisions, which results in the inaccuracy of our measurements for small round trip times.

C. Accuracy on routes with large round trip times

Next, we consider the accuracy of our measurements on routes that have a large round trip time. We define a large round trip time to be greater than 100ms. For such routes 63.2% are lower bounded, 67.1% are within 10% of the actual round trip time, and 87.2% are within 20% of the actual round trip time. On routes that experience no packet loss, our measurements determine a lower bound for 73.6% of measurements and estimate the round trip time within 10% for

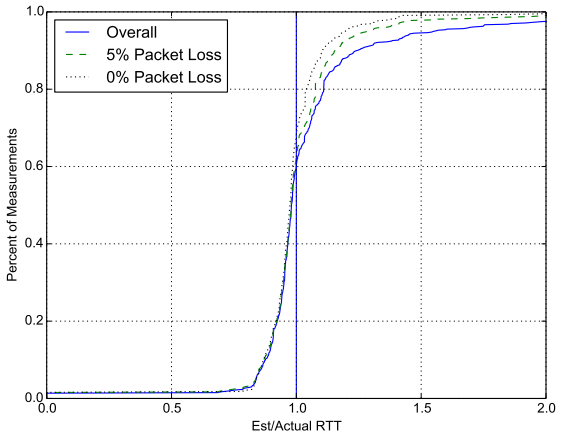

Fig. 5. Accuracy of round trip time estimates with actual RTT greater than 25ms

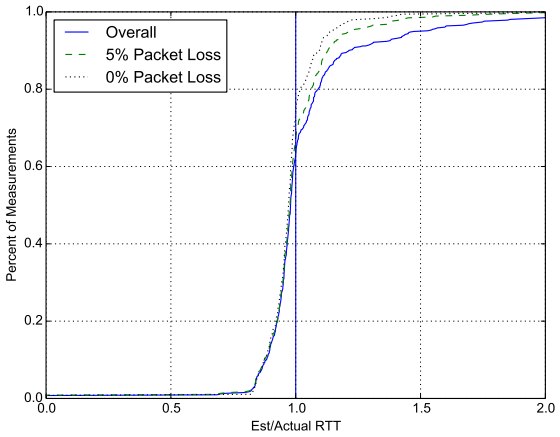

Fig. 7. Accuracy of round trip time estimates with actual RTT greater than 100ms

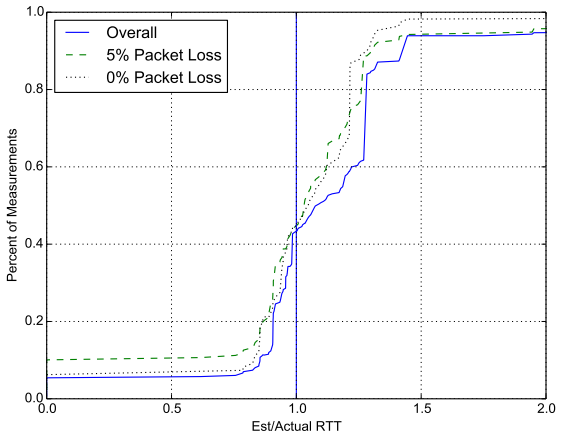

Fig. 6. Accuracy of round trip time estimates with actual RTT less than 100ms

75.9% of routes and within 20% on 96% of routes. Figure 6 compares measurements of routes with round trip time less than 100ms, with and without packet loss, and Figure 7 compares routes with round trip times greater than 100ms. Figures 8 and 9 compare our overall results across various round trip times, with and without packet loss.

D. Effect of large RTT variations

Another source of error is the impact that variations in the round trip time between the chosen hosts have on the accuracy of our measurements. Consider a route between two hosts, where the round trip time varies between a high of 400ms and a low of 200ms. In one iteration of the binary search the round trip time is measured as 200ms and we adjust our bounds accordingly. However, during the next iteration the round trip time jumps to 400ms. This new value is then measured and the bounds are adjusted. If the round trip time then drops near 200ms for the next iteration we will have incorrectly adjusted our bounds closer to 400ms, when they should be closer to 200ms. In some cases, depending on the chosen bounds, such an error could result in the actual round trip time no longer being within these new bounds, making accurate measurement impossible. For one particular pair of end hosts in our dataset, we noticed round trip time values as high as 2 seconds and as small as 87ms, during a single measurement period.

E. Effect of route changes

The final source of error we noted was changes to the route between the chosen end hosts. While minor changes to the route often had little impact on our final measurement, in a few cases new routes resulted in different overall round trip times. If these differences in round trip time are large enough then some packets may arrive sooner or later than anticipated. If packets arrive too soon the SYN-backlog will not fill up and evict "mature" SYNs in cases where it should. However, if packets arrive too late then the backlog will fill in cases where it should not and cause packets to be evicted. Both cases can result in inaccurate measurements.

VI. Ethical Concerns

When performing network measurement scans there are potential ethical issues that each scan must address and our scan is no exception. Every packet sent by a scan consumes resources for each scanned machine. These resources include network bandwidth, CPU cycles dealing with incoming packets, and time spent by system administrators determining if packets are a threat to the network. For our scan there are a few possible ethical issues that we feel should be discussed and addressed.

The first is the use of the SYN-backlog as a side channel. As part of our measurements we need to fill the SYN-backlog of the Server machine at least half full with SYN packets that will never be responded to. This creates a drain of resources on the Server by causing the creation of half-open TCP connections for the duration of the scan. If our scan were to fill the

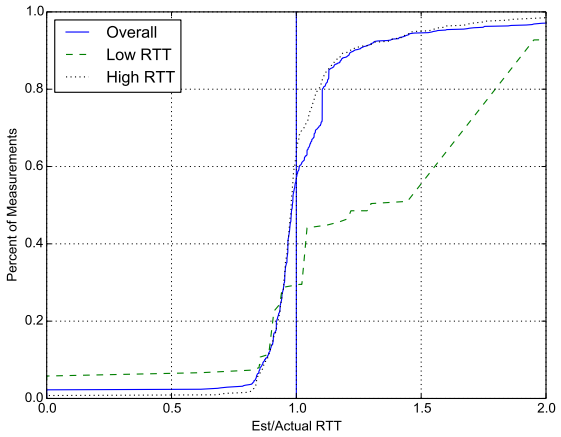

Fig. 8. Comparison between overall dataset, lower RTT only, and high RTT only

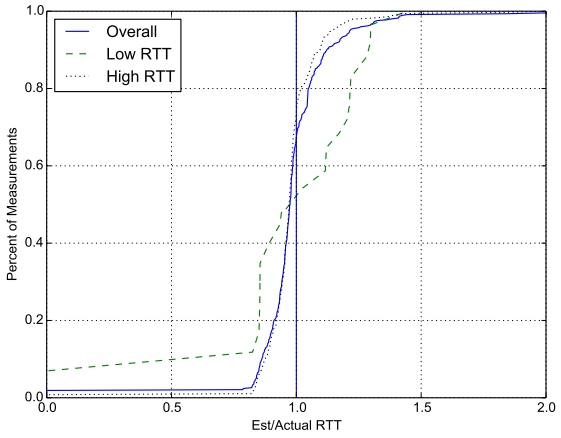

Fig. 9. Comparison between overall dataset, lower RTT only, and high RTT only with no Packet Loss

backlog completely this might cause a denial-of-service if preventative measures are not taken by the Server. However, our scan should never fill the backlog beyond half full for more than one-fifth of a second, due to the behavior of the Linux kernel. When the backlog is half full the kernel will remove "mature" packets, including those that we sent to fill the backlog initially five times every second. Because of this behavior, which we exploit to perform our scan, our scan will never cause a denial-of-service due to filling the SYN-backlog with unfinished connection attempts.

Another potential issue is that the act of partially filling the SYN-backlog over a short period of time will be incorrectly identified as the start of an attempted denial-of-service attack by a system administrator, firewall, or intrusion detection system. While we cannot guarantee that this will never happen as an initial reaction to our scan we feel that further inspection of our network traffic will reveal that the transmission and creation of half-open TCP connections lasts a short time, typically only a few seconds, and is not continuous or large enough to cause sufficient resource allocation to result in a denial-of-service, as discussed above.

The third issue to be addressed is the amount of network traffic generated by our scan. When attempting to measure paths with small round trip time values, usually below 20ms, our scan can send upwards of 800 packets per second when attempting to trigger the SYN-backlog behavior we use to gather our data. Each of these packets is an unsolicited SYN-ACK packet which will be immediately reset by the Client on arrival resulting in very little system resources being used to process the packet since it is immediately reset without any further processing. Each packet is 60 bytes in size and at a rate of 800 packets per second would result in 48 kilobytes per second of network traffic, a level that we feel modern systems and networks are more than capable of handling without noticeably affecting the quality of communication.

VII. Related Works

Previous work to measure off-path round trip times has often involved access to multiple vantage points to act as one end host. IDMaps [4] takes this approach. iPlane [5] uses multiple vantage points to measure round trip times, along with packet loss and other useful network performance metrics. De A. Rocha *et al.* [6] present a technique for estimating delay and variance using forged return IP addresses. King [1] and Queen [7] estimate off-path round trip time and off-path packet loss rate by using recursive DNS queries to DNS servers topologically near each end point. Due to this King, like our technique, does not require the use of multiple vantage points or direct access to either end host. In contrast our technique does not require the use of additional machines, such as King's DNS servers, or protocols beyond TCP.

Another approach for estimating round trip times is to attempt to provide a coordinate system in which end hosts can be placed. Once a host has been given a set of coordinates round trip time estimates can be made based on the distance between two end hosts, based on their coordinates. Vivaldi [8] and GNP [9] use such coordinate systems. Hariri *et al.* [10] develop a distributed latency estimation system using a coordinate based approach while Zheng *et al.* [11] explore how routing policies on the Internet can affect the accuracy of such coordinate systems.

Network Tomography [12], [13], [14] is a method for inferring large-scale characteristics about the internal structure of a network using probes sent from edge hosts, on the outer edge of the network. Previous work by Tsang *et al.* [15] has shown that tomography can be used to measure network delay.

Information side-channels have seen use in other network measurement or scanning applications. Chen *et al.* [16] show how inferences made using the IPID field can be used to determine the amount of internal traffic generated by a server, the number of servers used for load-balancing, and one-way delays in remote networks. Ensafi *et al.* show how side-channels can be used to detect intentional packet drops [17] and present additional side-channels for off-path port scanning [18]. Qian

et al. [19] discuss methods for inferring off-path TCP sequence number using side-channels present in firewall middleboxes.

Our technique makes use of spoofed, also referred to as forged, source IP addresses. Reverse traceroute [20] utilizes spoofed source IP addresses to perform a reverse traceroute. That is, instead of the path to an end host, the return path from an end host is traced. PoiRoot [21] uses spoofed IP addresses to investigate Interdomain Path changes, while Flach *et al.* [22] use spoofed IP addresses to locate Destination-Based Forwarding rule violations.

VIII. Conclusion

We have presented a novel technique for inferring the round trip time between two Internet end hosts, who meet a few simple behavioral requirements, using side-channels in the TCP/IP stack of modern Linux kernels. Specifically our technique uses a side-channel present in the kernel's SYN-backlog to infer off-path round trip times using a simple binary search. Compared to similar techniques, our system does not require the use of any additional machines, beyond the end hosts and the scanning machine, nor the use of any protocols beyond standard TCP/IP. We only require that one end host have an open port and be running a Linux kernel of version 2.6 or higher and that the other end host respond to unsolicited SYN-ACKs with RST packets.

We have provided an evaluation of our technique "in the wild" and compared our estimates with actual round trip time measurements. We evaluated our technique across a range of round trip times and routes and discussed its strengths and weaknesses for specific route characteristics. We have shown that our technique is accurate and discussed possible causes of error that impact our measurement accuracy.

IX. Acknowledgements

We would like the thank Xu Zhang for his helpful discussions about Linux TCP SYN-backlog behavior and Ben Mixon-Baca and Bianca Bologa for their help setting up virtual machines for our preliminary experiments. We would also like to thank Antonio Espinoza, Stephen Harding, and our anonymous reviewers for their helpful feedback on this work. This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. #0844880, #0905177, #1017602, #1314297, and #1420716. Jed Crandall is also supported by the DARPA CRASH program under grant #P-1070-113237.

References

[1] K. P. Gummadi, S. Saroiu, and S. D. Gribble, "King: Estimating latency between arbitrary Internet end hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 5–18.

[2] V. N. Padmanabhan and L. Subramanian, "An investigation of geographic mapping techniques for Internet hosts," in *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4. ACM, 2001, pp. 173–185.

[3] J. Postel, "Transmission Control Protocol," Internet Requests for Comments, RFC Editor, RFC 793, September 1981. [Online]. Available: http://tools.ietf.org/html/rfc793

[4] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "IDMaps: A global Internet host distance estimation service," *Networking, IEEE/ACM Transactions on*, vol. 9, no. 5, pp. 525–540, 2001.

[5] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, "iPlane: An information plane for distributed services," in *Proceedings of the 7th symposium on Operating systems design and implementation*. USENIX Association, 2006, pp. 367–380.

[6] A. d. A. Antonio, R. M. Leao, and E. d. S. e Silva, "A non-cooperative active measurement technique for estimating the average and variance of the one-way delay," in *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet*. Springer, 2007, pp. 1084–1095.

[7] Y. A. Wang, C. Huang, J. Li, and K. W. Ross, "Queen: Estimating packet loss rate between arbitrary Internet hosts," in *Passive and Active Network Measurement*. Springer, 2009, pp. 57–66.

[8] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," in *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4. ACM, 2004, pp. 15–26.

[9] T. E. Ng and H. Zhang, "Predicting Internet network distance with coordinates-based approaches," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1. IEEE, 2002, pp. 170–179.

[10] N. Hariri, B. Hariri, and S. Shirmohammadi, "A distributed measurement scheme for Internet latency estimation," *Instrumentation and Measurement, IEEE Transactions on*, vol. 60, no. 5, pp. 1594–1603, 2011.

[11] H. Zheng, E. K. Lua, M. Pias, and T. G. Griffin, "Internet routing policies and round-trip-times," in *Passive and Active Network Measurement*. Springer, 2005, pp. 236–250.

[12] Y. Vardi, "Network tomography: Estimating source-destination traffic intensities from link data," *Journal of the American Statistical Association*, vol. 91, no. 433, pp. 365–377, 1996.

[13] R. Castro, M. Coates, G. Liang, R. Nowak, and B. Yu, "Network tomography: recent developments," *Statistical science*, pp. 499–517, 2004.

[14] M. Coates, A. Hero, R. Nowak, and B. Yu, "Internet tomography," *Signal Processing Magazine, IEEE*, vol. 19, no. 3, pp. 47–65, 2002.

[15] Y. Tsang, M. Coates, and R. D. Nowak, "Network delay tomography," *Signal Processing, IEEE Transactions on*, vol. 51, no. 8, pp. 2125–2136, 2003.

[16] W. Chen, Y. Huang, B. F. Ribeiro, K. Suh, H. Zhang, E. d. S. e Silva, J. Kurose, and D. Towsley, "Exploiting the IPID field to infer network path and end-system characteristics," in *Passive and Active Network Measurement*. Springer, 2005, pp. 108–120.

[17] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting intentional packet drops on the Internet via TCP/IP side channels," in *Passive and Active Measurement*. Springer, 2014, pp. 109–118.

[18] R. Ensafi, J. C. Park, D. Kapur, and J. R. Crandall, "Idle port scanning and non-interference analysis of network protocol stacks using model checking." in *USENIX Security Symposium*, 2010, pp. 257–272.

[19] Z. Qian and Z. M. Mao, "Off-path TCP sequence number inference attack-how firewall middleboxes reduce security," in *Security and Privacy (SP), 2012 IEEE Symposium on*. IEEE, 2012, pp. 347–361.

[20] E. Katz-Bassett, H. V. Madhyastha, V. K. Adhikari, C. Scott, J. Sherry, P. Van Wesep, T. Anderson, and A. Krishnamurthy, "Reverse traceroute," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 15–15. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855711.1855726

[21] U. Javed, I. Cunha, D. Choffnes, E. Katz-Bassett, T. Anderson, and A. Krishnamurthy, "PoiRoot: Investigating the root cause of interdomain path changes," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 183–194. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486036

[22] T. Flach, E. Katz-Bassett, and R. Govindan, "Quantifying violations of destination-based forwarding on the Internet," in *Proceedings of the 2012 ACM Conference on Internet Measurement Conference*, ser. IMC '12. New York, NY, USA: ACM, 2012, pp. 265–272. [Online]. Available: http://doi.acm.org/10.1145/2398776.2398804