ONIS: Inferring TCP/IP-based Trust Relationships Completely Off-Path

Xu Zhang
Department of Computer Science
University of New Mexico
xuzhang@cs.unm.edu

Jeffrey Knockel
Department of Computer Science
University of New Mexico
jeffk@cs.unm.edu

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

Abstract—**We present ONIS, a new scanning technique that can perform network measurements such as: inferring TCP/IP-based trust relationships off-path, stealthily port scanning a target without using the scanner's IP address, detecting off-path packet drops between two international hosts. These tasks typically rely on a core technique called the idle scan, which is a special kind of port scan that appears to come from a third machine called a zombie. The scanner learns the target's status from the zombie by using its TCP/IP side channels.**

Unfortunately, the idle scan assumes that the zombie has IP identifiers (IPIDs) which exhibit the now-discouraged behavior of being globally incrementing. The use of this kind of IPID counter is becoming increasingly rare in practice. Our technique, unlike the idle scan, is based on a much more advanced IPID generation scheme, that of the prevalent Linux kernel. Although Linux's IPID generation scheme is specifically intended to reduce information flow, we show that using Linux machines as zombies in an indirect scan is still possible. ONIS has 87% accuracy, which is comparable to nmap's implementation of the idle scan at 86%. ONIS's much broader choice of zombies will enable it to be a widely used technique which can fulfill various network measurement tasks.

I. Introduction

Port scanning is a critical first step for penetration testers to understand network structure, in which a measurement machine sends probes directly to a target and, *e.g.*, determines if a given port is open or closed based on the received responses. Other direct scans perform OS detection, service versioning, and obtain other information by directly probing a machine.

In 1998, Antirez proposed an indirect type of scan called idle scan [1]. In the idle scan, the measurement machine spoofs the return IP address of probes so that the scan appears to be coming from another machine. Side-channel information is then used by the measurement machine to infer how the target responded. Side channels are necessary because the *zombie*, which is the machine used as the return IP address of the probe, is not under the scanner's control. Thus, the network scanner has no direct way of knowing what packets the zombie receives from the target. We further discuss Antirez's idle scan Section II-C.

The idle scan, although originally intended for learning the status of a port, in general can be used to learn the trust relationship between two arbitrary hosts that a network researcher does not control. For example, consider a network researcher in country X who wants to learn if network traffic from a host in country Y can connect to a Tor server in country Z. Performing this measurement off-path is necessary when vantage points (VPNs, Planet Lab nodes, *etc.*) are limited or unavailable in some countries. Ensafi *et al.* detail this off-path trust relationship testing by using the idle scan in [2]. Specifically, they measured packet drops from clients to Tor directory servers by using machines with global incrementing IPIDs as vantage points without those machines being under their control.

Unfortunately, use of the idle scan has two major issues.

1) It requires that the zombie has a globally incrementing IPID. Many modern network stacks are specifically designed to prevent information flow through IPIDs. One of the most advanced network stacks in this respect is Linux. The Linux IPID generation algorithm is described in Section II-D.
2) It also assumes that the zombie is idle, hence the name "idle scan". Internet-connected hosts are seldom idle. Ensafi *et al.* proposed using an autoregressive moving average (ARMA) model to handle the noise on zombie machine in [2], but sometimes the process of fitting an ARMA model to the data fails if the zombie machine is often not idle (*e.g.*, web servers).

Motivated by the goal of overcoming both of these drawbacks of the idle scan, we propose **ONIS**: **O**NIS is **N**ot an **I**dle **S**can, which uses an up-to-date Linux machine as the zombie. ONIS extends the choices of zombies of the idle scan, by using up-to-date linux machines as zombies. According to our estimate in Section VI, 17% of web servers are potential zombies that can be used. Unlike an idle scan, ONIS does not require the zombie to be completely idle. Although ingress filtering prevents our scan by not allowing packets with spoofed IP addresses into a network, only 23.9% of the networks on the Internet actually perform this [3].

We summarize our major contributions as follows:

1) We propose ONIS: a novel indirect scanning technique using TCP/IP side channels. The idle scan requires a global incrementing IPID zombie machine, which has been gradually phased out in many major OSes. ONIS uses Linux machines with kernel 3.16 or later as zombies and does not require the zombie to be idle. ONIS

achieves 87% accuracy, which is roughly as accurate as the idle scan at 86%. In the meantime, it allows a much broader choice of zombies.

2) We propose a new technique to do IPv4/IPv6 alias resolution on Linux machines with kernel 3.16 or later.
3) We perform a detailed analysis of noise in our scan technique and propose an effective model selection method to handle noise.

The rest of this paper is structured as follows: Section II gives a review of what an IPID is and how Linux generates IPIDs. Section III talks about the methodology of ONIS, as well as a new technique to perform IPv4 and IPv6 alias resolution and a model selection method called "AIC". Section IV describes the details of our experimental setup. We provide a direct comparison of results between ONIS and nmap implementation of the idle scan in Section V. In Section VI we discuss the applicability of ONIS, ethics concerns and possible defenses against ONIS. We document related work in Section VII and our conclusion in Section VIII.

II. Background

In this section, we briefly review IP identifiers and then discuss Linux's changing approach to generating them in response to different attacks.

A. IP Identifiers

Every IPv4 packet contains a 16-bit field known an *IP identifier* (IPID). When an IP datagram is too large to be transmitted over a link, a router can break it up into smaller packets called *fragments*. The datagram's final destination can reassemble the original datagram by collecting each incoming fragment until all fragments have been received. The final destination uses each fragments' IPID to determine which datagram it belongs to.

IPv6 is different from IPv4 in this matter in that every IPv6 packet does not necessarily have an IPID. If fragmentation of a datagram is required for it to reach its final destination, then the original sender fragments the datagram, adding to each fragment an IPv6 extension header containing a 32-bit identifier, a field 16 bits larger than an IPv4 packet's IPID.

B. Early Linux IPID generation

The Linux kernel originally determined each IPv4 datagram's IPID by using a globally incrementing counter. Every time a host sends a datagram, the value of the counter is incremented (mod 2^{16}) and then used as that datagram's IPID.

In 1998, a technique called an *idle scan* was discovered to port scan machines off-path by exploiting globally incrementing counters as a side-channel [1]. In response, kernel developers switched to having a separate counter for each IP destination.

C. The idle scan

A measurement machine can use the idle scan technique to port scan a target machine completely off-path by performing the procedure described in this section.

Before the scan, the measurement machine identifies a suitable *zombie* machine with the following characteristics: the measurement machine can communicate with the zombie, the zombie can communicate with the target, the zombie responds to SYN-ACKs with RSTs, the zombie has a globally incrementing IPID counter, and its network communication is *idle*, *i.e.*, aside from the scan, it is not otherwise sending any datagrams.

The measurement machine first probes the current value of the zombie's IPID counter by sending it a SYN-ACK packet. The zombie responds with a RST packet with the current value of its IPID counter. Next, the measurement machine sends a SYN packet to the target with the source address spoofed to be that of the zombie. If the destination port number of the spoofed SYN packet is open on the target, the target will send a SYN-ACK to the zombie, and the zombie, not expecting the SYN-ACK since it did not send the spoofed SYN, sends a RST to the target, incrementing the zombie's IPID counter. Otherwise, if the port on the target is closed, the target sends a RST to the zombie, which does not cause the zombie to send any packets, and so the zombie's IPID counter is unaffected. Finally, the measurement machine sends another SYN-ACK probe to the zombie to once again measure the current value of its IPID counter. If the IPID of the responding RST packet is one greater (mod 2^{16}) than that of the last probe, then the destination port of the spoofed SYN must be closed on the target. However, if the IPID of the RST is two greater (mod 2^{16}) than that of the original probe, then the port is open, since the SYN-ACK the target sent to the zombie incremented its counter in between the probes.

D. Recent Linux IPID generation

In 2014, in Linux 3.16, the kernel developers recognized performance issues with using a separate counter for each IP destination [4]. Since having a globally incrementing counter was still undesirable, they adopted a hybrid approach consisting of 2048 globally incrementing counters. To determine which counter to use for an IP datagram, that datagram's destination address is hashed with a secret value randomly generated at system startup. The resulting hash (mod 2^{11}) is used to determine the index of the counter. Each counter is 32 bits to accommodate IPv6, and for IPv4 the IPID is taken from only the lower 16 bits of the counter.

During the same time, a side-channel technique was discovered that could count the number of datagrams sent between machines off-path [5]. The technique worked by inferring the values of per-destination IPID counters off-path but could also be extended to work for the kernel's new hybrid approach [6]. In response, the kernel developers made additional changes to make each counter less predictable. Every time a counter is used to assign an IPID, instead of incrementing it by one, the kernel adds to it a number uniformly distributed between 1 and the number of system ticks since the counter was last used.

In light of defending against machines being used as zombies in the idle scan, this new hybrid approach was identified as partially having the problems of per-destination counters and partially having the problems of a globally incrementing counters [7]. It has the problems of per-destination counters in that the counters would still partially isolate information about which hosts a zombie is sending packets to, since the probability of any two destination machines hashing to the same counter is only $1/2048$. This means that a zombie need not necessarily be completely idle, only the counter that it uses to send packets to the target need be idle. Moreover, it partially has the problems of globally incrementing IPID counters in that, if a measurement machine has an address hashed to the same counter on the zombie as that of the target, then that counter shares information between the target and the measurement machine the same way a single globally incrementing counter would. However, the probability of this occurring between any one measurement machine address and any one target machine address is only $1/2048$.

The effort to make IPID counters less predictable by adding a value uniformly chosen at random also makes the idle scan more difficult to perform due to the random noise being added. The addition of this random noise results in the number of datagrams the zombie sends no longer significantly affecting the expected value of the observed increase to its IPID counter. However, it has been noted [7] that, although the expected value does not significantly change, the number of packets sent by the zombie still changes the distribution by which the IPID counter is observed to have increased. This is because, if $\mathcal{U}(a, b)$ is a discrete uniformly distributed observation between a and b, then $\mathcal{U}(1, n)$, the observed increase when no datagram from some counter has been sent in between IPID probes received n system ticks apart, has a different distribution than $\mathcal{U}(1, n/2) + \mathcal{U}(1, n/2)$, which would be the approximate observed increase if one datagram were sent exactly in between the two IPID probes having been received. Note that for large k and n, $\sum_{i=1}^{k} \mathcal{U}(1, n/k)$ approximates a normal distribution due to the central limit theorem.

III. Implementation

In this section, we describe how ONIS works by using Linux machines with kernel 3.16 or later as zombies. We start by describing a general approach to perform ONIS. Then we propose a new technique to do IPv4 and IPv6 alias resolution on Linux machines with kernel 3.16 or later. Next, we introduce our implementation of ONIS which uses dual-stack Linux machines as zombies. Finally, we talk about how to process the result by using a model selection technique called "Akaike information criterion" (AIC).

A. Overview of ONIS

ONIS requires that the zombie machine is running Linux with kernel version 3.16 or later and replies to unsolicited SYN-ACKs with RSTs as per RFC 793 [8]. Similar to the idle scan described in Section II-C, there are also three steps for ONIS.

In the first step, the measurement machine sends a SYN-ACK packet to the zombie machine, using the measurement machine's IP address as its source IP address. According to RFC 793 [8], the zombie will reply with a RST packet back to the measurement machine, since the zombie did not send any SYN. Let the IPID in the RST packet be x_1 and the time in system ticks when the zombie generates x_1 be t_1. After receiving the RST packet from the zombie, the measurement machine records x_1.

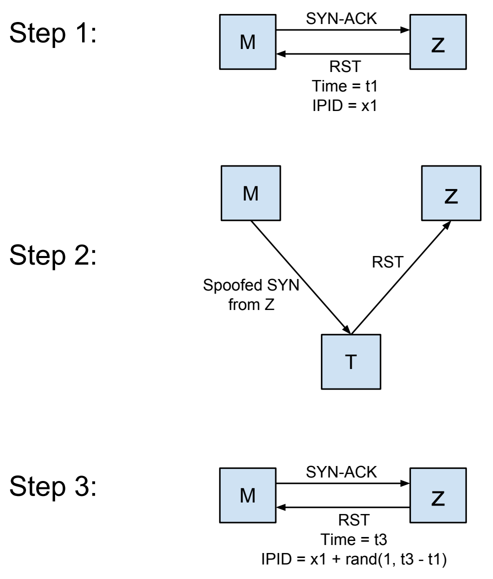

Fig. 1. Scan of closed port on the target using ONIS.

In the second step, the measurement machine sends a spoofed SYN packet to a port of the target machine using the zombie machine's IP address as the source IP address. Depending on the status of the port (open, closed/filtered), the target will respond differently.

1) In the case that the port on the target is closed (see Figure 1), the target will reply with a RST packet to the zombie. The zombie will simply ignore the RST and not send any packets in response. For a filtered port in the target, the SYN packet spoofed from the zombie is silently filtered and thus there is no traffic between the zombie and the target. The result is the same as a closed port scenario, since the zombie will not generate any new IPIDs.
2) In the case that the port on the target is open (see Figure 2), the target will reply with a SYN-ACK packet to the zombie. The zombie, which did not send any SYN to the target, will send a RST packet to reset the handshake. Let the IPID in this RST packet be x_2 and the time in system ticks it was generated at be t_2. Here we assume that x_2 draws from the same Linux IPID counter as x_1, as we will later show how to ensure this, which we discuss in Section III-B. Then, according to the Linux behavior of generating IPIDs (see Section II-D), we have $x_2 = x_1 + \mathcal{U}(1, t_2 - t_1)$,

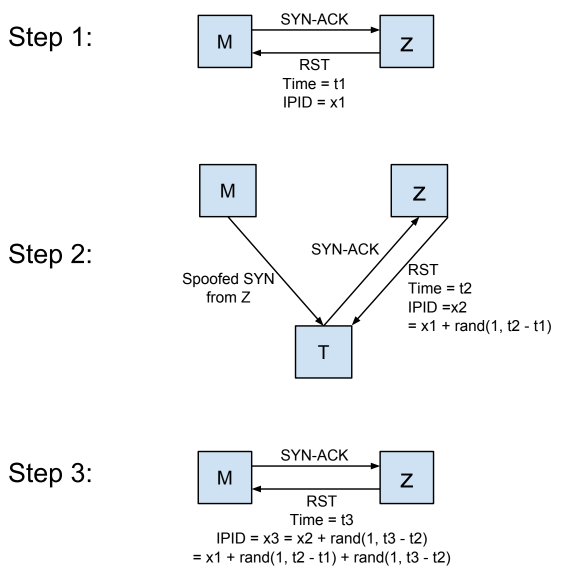

Fig. 2. Scan of an open port on the target using ONIS.

where $\mathcal{U}(a, b)$ is a discrete uniformly distributed random variable as before.

In the third step, similarly to step one, the measurement machine sends a SYN-ACK to the zombie to collect x_3, the IPID in the following RST. Let the time in system ticks when the zombie generates x_3 be t_3. Then the distribution of x_3 will differ according to the status of the port of the target.

1) If the port is closed, the zombie only generates an IPID in the first and third steps, thus yielding $x_3 = x_1 + \mathcal{U}(1, t_3 - t_1)$.
2) If the port is open, as seen in step 2, the zombie has an additional access to the IPID counter in which case $x_3 = x_2 + \mathcal{U}(1, t_3 - t_2) = x_1 + \mathcal{U}(1, t_2 - t_1) + \mathcal{U}(1, t_3 - t_2)$.

Whether x_2 is generated is not directly known to the measurement machine. However, from repeated measures of the values of x_1 and x_3, it is possible to infer the status of the port on the target by analyzing the distribution of their differences, $x_3 - x_1$.

B. Finding dual-stack Linux machines

As described in Section II-D, Linux 3.16 or later uses 2048 global counters to generate IPIDs. The scan method we talked about in the previous section relies on the fact that the measurement machine has an address that hashes to the same IPID counter on the zombie as that of the target. It is possible to try differing measurement machine addresses until a collision is found with the target's IP address. Each time, we have a possibility of $1/2048$ for the hashes to collide. If we have 10,000 IP addresses to try, the chance to have a collision of a certain IP address at least once is more than 99%. $(1 - (\frac{2048-1}{2048})^{10000} \approx 99.2\%)$

Such resources are usually within the capabilities of network researchers, especially considering how easy to obtain a /64 of IPv6 addresses nowadays (a /114 of IPv6 addresses would be sufficient). In our experiment, we demonstrate how ONIS works by using multiple IPv6 addresses in our measurement machine since the same 2048 IPID global counters are used by both IPv4 and IPv6.

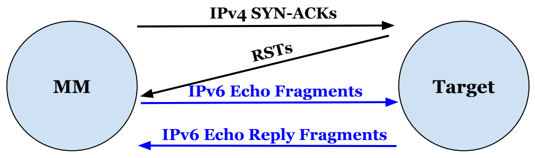

Fig. 3. IPv4 and IPv6 alias resolution.

Now we present a new technique to do IPv4 and IPv6 alias resolution on a Linux machine with kernel 3.16 or later. Previous alias resolution techniques are either IPv4 or IPv6 only, and so our alias resolution technique is novel. We use a TCP/IP side channel discussed above in Section II-D to achieve this, as shown in Figure 3. Here we call the machine we perform IPv4 and IPv6 alias resolution on the "target", although for ONIS this target will become the zombie.

Given an IPv4 address and an IPv6 address, in each round we simultaneously send an IPv4 SYN-ACK packet and a large IPv6 Echo Request to the target to collect its IPv4 IPID and IPv6 fragment ID. We fill the IPv6 Echo Request's body such that the unfragmented size of the datagram is 2000 bytes and so the subsequent reply will require fragmentation to reach the measurement machine, ensuring that it will contain an IPv6 extension header containing an IPv6 fragment ID. Then, we vary the source IPv6 address with another, and resend the same type of packets.

If an IPv4 address (a_4) and an IPv6 address (a_6) are aliases of the same Linux 3.16 or later machine, then it is possible to find a measurement machine IPv6 address that hashes to the same IPID counter receiving probe replies from a_6 as our measurement machine's IPv4 address does receiving probe replies from a_4. We test 10,000 different measurement machine IPv6 addresses. For each measurement machine IPv6 address we would like to test, we generate ten IPv4 SYN-ACK probes and ten large, 2000 byte IPv6 Echo Requests, large enough so that the probed machine will need to reply adding IPv6 fragment extension headers containing fragment IDs. We alternate sending each IPv4 and IPv6 probe, waiting 0.05 seconds between sending each probe.

If x_i stands for the ith result of the IPv4 probe, and y_i stands for the ith result of the IPv6 probe (mod 2^{16}), then if we have a strictly increasing sequence $x_1 < y_1 < x_2 < y_2 \ldots x_n < y_n$ and $y_i - x_i \geq 1$ and $\sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i > n$, where $n = 10$, then we conclude that the machine is dual-stacked. (In this analysis, we say that $X < Y$ if Y occurs in X's upcoming half of the 16-bit sequence space.)

C. *Finding a collision with target's IP address*

In the previous section, we discussed a new technique to do IPv4 and IPv6 alias resolution on Linux machines with kernel 3.16 or later. Assuming we will use an IPv4 and IPv6 dual stack machine as the zombie machine in ONIS, we also need to know the exact source IPv6 address that causes the collision with the target machine's IPv4 address. Similar to the previous method, we pick a source IPv6 address and send a 2000 byte IPv6 Echo Request to the zombie at t_1. Then we send a spoofed IPv4 SYN-ACK packet using the target's return address to the zombie at t_2. Finally, we send another 2000 byte IPv6 Echo Request at t_3.

We want our three probes to arrive in the order in which we sent them, but also, to eliminate the random noise that Linux adds to its IPID counters, we want them to arrive within one system tick from each other. This eliminates all noise because when they arrive one tick after each other, the kernel will increase the IPID counter by $\mathcal{U}(1,1) = 1$, but if they arrive within zero change of the system clock, the kernel still increments by one. If we can have the packets arrive in order and within one system tick of each other, then, for the open port case, we will observe an IPID increase of $1+1=2$ and for the closed port case, we will observe an IPID increase of only one.

Naively, we might want to send the probes at $t_2 = t_1 + 0.5$ms, and $t_3 = t_1 + 1$ms. (We choose milliseconds because, while the tick rate of the kernel is never faster than one tick per millisecond, it may be slower.) However, often the round trip time (RTT) between the measurement machine and the zombie are different for IPv4 versus IPv6 routes and so the order of IPv4 packets and IPv6 packets arriving at the zombie might be at different times than we expect.

To overcome this, we first send IPv4 and IPv6 probe packets to the zombie to estimate the average RTT between the zombie and the measurement machine both in IPv4 and IPv6. Let the measured IPv4 RTT be r_4, the measured IPv6 RTT be r_6, and the difference between them δ such that $r_4 + \delta = r_6$. By approximating the path from the measurement machine to the zombie as half of the RTT, we divide δ by two and use the adjustment $t_2 = t_1 + 0.5 + \delta/2$, leaving t_3 unchanged. This way we can increase the chance that all three probes arrive within one system tick of each other. Note than an IPv4-only version of ONIS would not face this challenge.

If the IPID returned from the third probe is at least two greater than that of the IPID returned by the first probe, we conclude that the tested IPv6 address shares the same IPID counter as that of the target. Otherwise, we conclude that it does not. (Here we say that $Y > X$ if Y occurs in X's upcoming half of the 32-bit sequence space.)

D. *An implementation of ONIS using dual-stack zombies*

After discovering a source IPv6 address that shares an IPID counter with the target's IPv4 address, we can adapt ONIS to use dual stack Linux machines as zombie machines. In the first step, the measurement machine sends a fragmented IPv6 Echo Request, whose unfragmented datagram is 2000 bytes, to

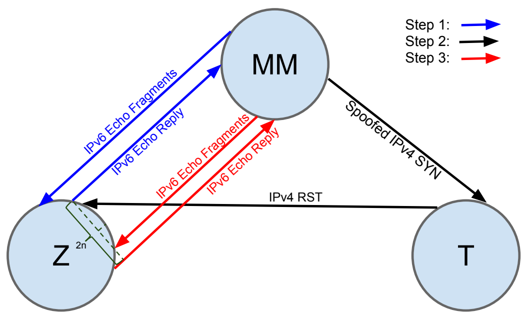

Fig. 4. Scan of a closed port with a dual stack zombie using ONIS.

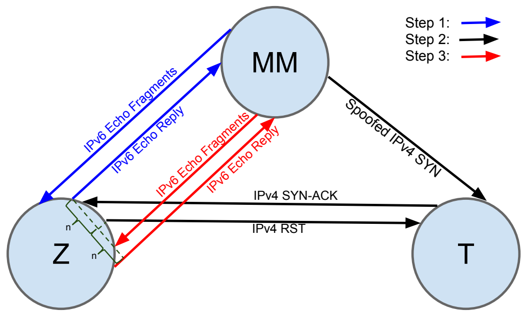

Fig. 5. Scan of an open port with a dual stack zombie using ONIS.

the zombie and records the IPID in the fragmented response. As before, a large request is used to ensure that the response is fragmented, ensuring that the IPv6 fragmentation extension header containing an IPID is included. The second step is exactly as before. In the final step, the measurement machine queries the IPID again by sending another fragmented IPv6 Echo Request. Figure 4 shows using ONIS when the target's port is closed. Figure 5 shows using ONIS when target's port is open. After collecting a group of IPIDs, we use the method in Section III-E to determine which model fits the data.

E. *Model selection and noise handling*

As mentioned in Section III-A, in order to find out if a target's port is open or closed, we need to be able to distinguish between two distributions, $\mathcal{U}(1, t_3 - t_1)$ and $\mathcal{U}(1, t_2 - t_1) + \mathcal{U}(1, t_3 - t_2)$. Let $t_3 - t_1 = 2n$. Moreover, we will approximate $t_2 - t_1 = t_3 - t_2 = n$. Now we need only distinguish between the distributions $\mathcal{U}(1, 2n)$ and $\mathcal{U}(1, n) + \mathcal{U}(1, n)$. We showed that it is trivial to distinguish simulated cases. However, in the experiment we found noise on the network made such analysis challenging. For example, the round trip time between the measurement machine and the zombie changed in different rounds when collecting IPIDs. Thus in practice we have $\mathcal{U}(1, 2n + \delta)$, where δ is a variant.

To overcome these issues, we use a model selection method which is resistant to noise when collecting IPIDs called Akaike information criterion (AIC). Unlike the null hypothesis testing approach, AIC does not give the quality of a single model with respect to a null hypothesis, but rather estimates the relative quality of one model with respect to another. Because of this, AIC is ideal for us to handle noise in the scan.

We use AIC to select between $\mathcal{U}(1, 2n)$ and $\mathcal{U}(1, n) + \mathcal{U}(1, n)$ for a given measurement dataset of IPIDs. By definition, AIC is defined as

$$AIC = 2k - 2\ln(\hat{L}) \tag{1}$$

where $\hat{L}$ is the maximum value of the likelihood function, and k is the number of parameters in the model. In our case, $k = 1$. Thus, according equation 1, we wish to find the model with larger maximum likelihood, $\hat{L}$, in order to get a smaller AIC value. Below we will show how to calculate $\hat{L}$ in both of the cases of our scan.

1) Case 1: Closed or filtered port: Assuming that an observed IPID increase was generated by $\mathcal{U}(1, 2n)$, the probability density function is:

$$f(x) = \begin{cases} \frac{1}{2n} & \textit{for } 1 \le x \le 2n \\ 0 & \textit{otherwise.} \end{cases} \tag{2}$$

The likelihood function is:

$$L(n) = \prod_{i=1}^{k} f(x_i) = \begin{cases} \frac{1}{(2n)^k} & \textit{for } 1 \le x_i \le 2n \textit{ for all } i \\ 0 & \textit{otherwise.} \end{cases} \tag{3}$$

where k is our sample size.

Note that the likelihood function defined in equation 3 may have maximum value when $1 \le x_i \le 2n$ for all i. Therefore, $\max\{x_1, \ldots, x_k\} \le 2n$, and $n \ge \lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil$. Note that in equation 3, $\frac{1}{(2n)^k}$ is monotonically decreasing when $n \ge 1$, *i.e.*, when $n = \lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil$, the likelihood function $L(n)$ in equation 3 gets its maximum value.

2) Case 2: Open port: Assuming that an observed IPID increase is generated by $\mathcal{U}(1, n) + \mathcal{U}(1, n)$, the probability density function is

$$f(x) = \begin{cases} \frac{n-|x-(n+1)|}{n^2} & \textit{for } 2 \le x \le 2n \\ 0 & \textit{otherwise.} \end{cases} \tag{4}$$

The likelihood function is:

$$L(n) = \frac{1}{n^{2k}} \prod_{i=1}^{k} (n - |x_i - (n+1)|) \tag{5}$$

for $2 \le x_i \le 2n$, for all i.

The monotonicity of the likelihood function in equation 5 cannot be determined *a priori*. However, we know that to get the maximum likelihood, $2 \le x_i \le 2n$ for all i, *i.e.* $\max\{x_1, \ldots, x_k\} \le 2n$, and $n \ge \lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil$. Therefore, we adopt a numerical approach and enumerate multiple possible n such that $n \ge \lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil$ to see which n maximizes $L(n)$ in equation 5. We try m such values from $\lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil$ to $\lceil \frac{\max\{x_1,\ldots,x_k\}}{2} \rceil + m$, finding the n that gives the maximum $L(n)$. The parameter m is tunable, but we provide evidence that it is typically very low in Section IV-B.

IV. Experimental Setup

In this section we describe the details of our experimental setup.

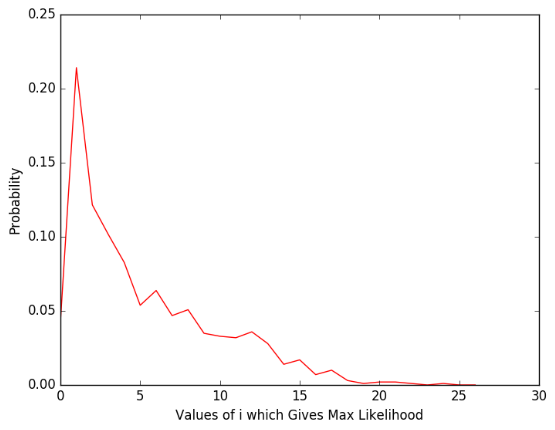

Fig. 6. After trying $m = 1,000$ different n, the distribution of iterations i it took to find the n that maximizes the likelihood function while performing ONIS.

A. Zombie selection

Our measurement machine ran Ubuntu server 16.04 with kernel version 4.4.0. The zombie machines we selected were IPv4 and IPv6 dual-stack Linux machines with kernel 3.16 or later. In the beginning, we collected a list of domains from the Alexa top one million websites and sixy.cn. For each domain name, we performed a DNS lookup to find its corresponding A record and AAAA record. Given pairs of IPv4 and IPv6 address, we performed IPv4 and IPv6 alias resolution using the method described in Section III-B. The IPv4 and IPv6 resolution tests were performed three times for every IPv4 and IPv6 pair to make sure it was a desired zombie. The rate we created packets on the zombie's network was about 30 packets per second. Note that to compare IPIDs in IPv4 and IPv6, we used a 16-bit mask to mask out the leftmost 16 bits of IPIDs in IPv6. We found 78 zombies which we used in our implementation of ONIS.

B. Performing the scan

To select a target for our ONIS scan, we randomly generated an IPv4 address and started to send SYN packets to three commonly open ports (22, 80 and 443) at that address. Each packet was re-sent three times to avoid possible packet loss. We recorded the IPv4 address as a valid target address if we received a SYN-ACK response for any of the three ports.

Once the scan started, the measurement machine created multiple threads, each randomly picking up a zombie machine from the pool and finding a valid target address as discussed above to perform the scan on ports 22, 80, and 443. The scanning methodology was described in Section III. For each port on a target, we collected 100 IPv6 IPID pairs, at a packet rate of three packets per second in the zombie's network. Each experiment takes about 20 minutes to finish. Then we processed the IPID samples by using AIC (see Section III-E).

In Section III-E, we mentioned that in order to find the maximum likelihood for the open port model, we had to enumerate the parameter n in m times. Each time we added one to n and calculated the corresponding likelihood L. During our experiment, we set $m = 1000$, as in practice we found out from our result that most of the time the first few i, where $0 \leq i \leq m$ lead to the maximum likelihood. Figure 6 shows the distribution of i which gives the maximum likelihood for model $\mathcal{U}(1,n)+\mathcal{U}(1,n)$. n is virtually always less than 1000.

To compare the accuracy of our results, we ran nmap's built-in idle scan (*nmap -Pn*) on the same targets that we performed ONIS on. Nmap implements the idle scan technique by using zombies with globally incrementing IPIDs. To find machines with globally incrementing IPIDs, we generated random IP addresses and tested to see if the IPIDs were globally incrementing. After we got a list of zombies, we culled the list for several rounds until all zombies appeared to be idle. We were able to identify 175 machines with globally incrementing IPIDs. Both the nmap idle scan results and ONIS results were compared with direct scan (SYN scan) results in order to calculate the accuracy.

V. Results

We collected 1309 results using 78 zombies starting May 1, 2017 and ending May 12, 2017. Each result showed whether specific ports (22, 80, 443) on a target machine were open or closed. We compared ONIS results with direct scan results and found that 1141 out of 1309 are correct, with an accuracy of 87.2%. There were 145 false negatives (failed to find an open port) and 23 false positives (reported a port as open that was not) out of 168 incorrect results. One possible reason for more false positives might be due to the fact that we found an incorrect collision with target IPv4 address using our IPv6 address before the scan. Thus once the scan started, the IPIDs in the TCP flow were not hashed into the expected bucket.

For comparison, we also collected 175 zombies with global incrementing IPIDs. To form a direct comparison between nmap results and ONIS results, we performed nmap idle scan on the same 1309 results. The nmap idle scan had 86.4% accuracy, with 1131 correct results and 178 incorrect results. 64 of the results were false positive, 57 of the results were false negative. 57 of the results showed that the zombie was too noisy to be used to perform the idle scan.

The resulting comparison of the two methods is shown in Table I. We can see that the overall accuracy of the two methods is comparable. Both scans have a certain amount of false negatives. One reason for that might be the possible ingress filtering in the target's network which prevents the spoofed SYN packets from the zombie. As a result, there is no subsequent traffic created. The zombie will not generate an IPID in response to the client. Both scans falsely assume that the port is closed in this case.

We also noticed that ONIS has more false negatives than the idle scan. We believe that is due to the fact that the previous step of collision finding is very sensitive to round trip time variations between IPv4 and IPv6. For example, route changes of IPv6 can cause the round-trip time for IPv6 between the zombie and the measurement machine to be larger. As a result, the second IPv6 echo fragments arrive at the zombie later than 1 millisecond. In this case, it is possible that the $\mathcal{U}(1, 1+\alpha)$ generates a number larger than 1, where α is a delay of the second IPv6 packet fragments.

	Corret	False Positive	False Negative	Failed
ONIS	1141	23	145	0
Idle Scan	1131	64	57	57

TABLE I
Result Comparision of ONIS and the Idle Scan

Table I also shows that nmap has more false positives. Although we ensure that zombies with global IPID on our list are all idle before the scan, it is possible that during the nmap idle scan the zombie is connecting with other hosts, causing it to no longer be idle thus breaking the scan. As a result, nmap falsely thinks that the target has an open port. We also noticed that during the nmap idle scan, every zombie may become active at certain time. In order to get ideal results with nmap's current implementation, we need to cull the list such that the zombies were noiseless right before we start the idle scan. Otherwise, the accuracy of results drops significantly. While for ONIS, there is only a $1/2048$ chance of interference with any other host because of Linux's 2048-bucket implementation.

VI. Discussion

A. Scan applicability

ONIS allows a broader choice of zombies and it is more reliable compared to the idle scan, which uses zombies with globally incrementing IPIDs.

The idle scan requires the zombie to be completely idle, while ONIS does not because the chance of every other connection on the zombie with the same global incrementing counter is just $1/2048$. Machines on the Internet are seldom idle, so in this sense, ONIS greatly improves on the reliability of the idle scan.

We set up experiments by using dual-stack zombies with Linux kernel 3.16 or later. This is just one implementation of ONIS to show that it works. We also provided a new technique to perform IPv4 and IPv6 alias resolution for Linux systems. However, ONIS is not limited to use on IPv4 and IPv6 dual stack systems.

In applications where the zombie can be, *e.g.*, from a specific large network (such as a particular country) and need not be a specific machine, we can scan and get a list of Linux machines with kernel 3.16 or later. Then for a randomly selected target, we can try every zombie in our list until a collision is found. Each time there's a 1/2048 chance that the measurement machine will share the same IPID counter on the zombie as the target. For 10,000 zombies, there is more than a 99% probability of finding a collision.

ONIS allows broader choices of zombies when inferring TCP/IP-based trusting relationship off-path. We performed a IPv4 SYN-ACK scan on the Alexa top web 1 million machines and found 170,630 of them to have per-flow IPIDs (about 17%), which are potential zombies that can be used in ONIS.

B. Ethical concerns

Compared to the idle scan, we perform an extra step to perform collision finding. However, the packets we send in this step are only IPv6 echo fragments and IPv4 SYN-ACKs, which should not consume too much computation resources on zombies' systems. Our IPv6 echo request is 2000 bytes long, and an IPv4 SYN-ACK is just 60 bytes. At a packet rate of 30 packets per second, we can create 40.6 kilobytes per second per zombie. During the scan on the target, the packet rate is only 3 packets per second. Since the spoofed SYNs to the target will end up reset by the zombie, it will not cause any denial of service on the target.

C. Defending against ONIS

One way that the Linux kernel could protect against being used as a zombie is to switch to a different kind of IPID counter. However, it is unclear which kind could protect widely against this sort of scan. Linux previously used per-destination counters, but this exposed them to a side channel attack that could leak the number of packets a machine sends another [5]. However, as the authors of that attack note, RFC 791 [9] mandates that IPIDs must be unique for every in-flight path, and so there will always be non-zero information flow in any shared sequence of numbers that has restrictions on repetition.

A strategy that may help defend against the technique used in this paper is to use a Poisson distribution instead of a uniform distribution for generating IPID noise. Our technique takes advantage of the fact that $\mathcal{U}(0, n)$ has a different distribution than $\mathcal{U}(0, n/2) + \mathcal{U}(0, n/2)$. However, the Poisson distribution does not have this limitation. If $\mathcal{P}(\lambda)$ is a Poisson random variable parameterized by rate λ, then $\mathcal{P}(\lambda) + \mathcal{P}(\mu) = \mathcal{P}(\lambda + \mu)$, and so $\mathcal{P}(\lambda) = \mathcal{P}(\lambda/2) + \mathcal{P}(\lambda/2)$. However, Poisson random numbers are computationally expensive to generate, and so they may not be suitable as a means to add noise to an IPID counter that may be accessed frequently.

VII. Related Work

Here we describe related work that is concerned with port scanning and that uses side channels.

Ensafi *et al.* [10] demonstrated that their SYN backlog TCP/IP side channel could be used to determine open *vs.* filtered ports on certain hosts, but the host-based firewall configurations that make this possible are not a common case. Follow-on work [11], [12] presented a hybrid method that combined the idle scan with the SYN backlog idle scan to improve the results and determine in which direction packets were being blocked by a firewall. The hybrid method still assumes that the zombie has a globally incrementing IPID, and is intended for global-scale measurements of national firewalls, *e.g.*, Internet censorship, rather than port scans of local networks. Zhang *et al.* [3] showed that the SYN backlog side channel can be used to find hidden machines behind firewalls, but did not attempt to make inferences about open ports.

Port scanning is an active research area. Nmap FTP Bounce Attack [13] is able to make FTP servers port scan an target server (Modern FTP servers are configured by default to prevent this). Staniford *et al.* [14] and Gates *et al.* [15] focus on large enterprise network protection. Leckie and Kotagiri [16] use a probabilistic approach to detect port scans. Treurniet [17] aims to detect stealthy scans using classification schema. Muelder *et al.* [18] proposes a visualization for port scan detection. Jung *et al.* [19] develop a fast port scanning detection method using the theory of sequential hypothesis testing. Other work [20], [21], [22] use a neural network approach to detect malicious port scanning. Gates [23], [24] and Kange *et al.* [25] considers stealthy port scans that are based on using many distributed hosts. There has also been some research on improving port scans, such as port scan techniques that increase the speed of horizontal scans based on techniques that use the same principle as SYN cookies [26], [27], [28], [29].

Idle scans and TCP/IP side channels are a nascent area of research, but there has been a considerable amount of work. Morbitzer [30] explores idle scans in IPv6. Qian *et al.* [31], [32] infer the TCP sequence number of a connection and perform off-path TCP/IP connection hijacking using a firewall-based side channel. Some work uses global IPID fields to perform inference for Internet measurement purposes. Chen *et al.* [33] explore new uses of the IPID to infer the amount of internal traffic generated by a server, the number of servers in a large scale server complex, and one-way delays to a target computer. Bellovin [34] describes a technique to detect NATs and count the number of hosts behind them. Kohno *et al.* [35] use the IPID to perform remote device fingerprinting. Knockel and Crandall [5] demonstrated that it was possible in a previous iteration of the Linux IPID generation algorithm to count packets sent to a specific destination by a remote server, and subsequently Cao *et al.* [36] demonstrated that related techniques can be used to interfere with connections completely off-path. Quach *et al.* [37] performed a comprehensive measurement of the impact of the ACK limiting vulnerability. The work of Gilad and Herzberg in this area is also notable [38], [39], [40].

Spoofed return IP addresses and side channel inferences [10], [31], [11] have been shown to be very useful for Internet measurement, see, *e.g.*, Chen *et al.* [33]'s inferences based on IPIDs, reverse traceroute [41], or PoiRoot [42], or Flach *et al.* [43]'s use of spoofed IP addresses to locate Destination-Based Forwarding rule violations.

VIII. Conclusion

We presented ONIS, a novel scanning technique which provides much broader choices of zombies when performing off-path TCP/IP trust relationship measurement. ONIS's accuracy is comparable to nmap's implementation. One caveat is that ONIS requires access to many IP addresses for the measurement machine, but the scan is flexible enough to enable different trade-offs in this sense, and IP addresses are easily obtainable in various ways. We expect that ONIS will become an essential part of a network researcher's toolbox and fulfill the practical potential for various network measurement tasks.

IX. Acknowledgments

We would like to thank the anonymous reviewers and Antonio Espinoza for valuable feedback. We would like to thank Ben Edwards for helpful discussions about Akaike information criterion. This material is based upon work supported by the U.S. National Science Foundation under Grant Nos. #1518523, #1518878.

References

[1] Antirez, "new tcp scan method," Posted to the bugtraq mailing list, 18 December 1998, 1998.

[2] R. Ensafi, J. Knockel, G. Alexander, and J. R. Crandall, "Detecting intentional packet drops on the Internet via TCP/IP side channels."

[3] X. Zhang, J. Knockel, and J. R. Crandall, "Original SYN: Finding machines hidden behind firewalls," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 720–728.

[4] E. Dumazet, "inetpeer: get rid of ip_id_count," 2014. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/commit/?id=73f156a6e8c1074ac6327e0abd1169e95eb66463

[5] J. Knockel and J. R. Crandall, "Counting packets sent between arbitrary internet hosts," in *4th USENIX Workshop on Free and Open Communications on the Internet (FOCI 14)*, 2014.

[6] E. Dumazet, "ip: make ip identifiers less predictable," 2014. [Online]. Available: https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git/commit/?id=04ca6973f7c1a0d8537f2d9906a0cf8e69886d75

[7] J. Knockel and J. R. Crandall, "Counting packets sent between arbitrary internet hosts (slides)," 2014. [Online]. Available: https://www.cs.unm.edu/~jeffk/publications/foci2014counting.pdf

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

[9] J. Postel *et al.*, "Rfc 791: Internet protocol," 1981.

[10] 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.

[11] 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.

[12] ——, "Detecting intentional packet drops on the Internet via TCP/IP side channels: Extended version," *CoRR*, vol. abs/1312.5739, 2013, available at http://arxiv.org/abs/1312.5739.

[13] "FTP-bounce," https://nmap.org/nsedoc/scripts/ftp-bounce.html.

[14] S. Staniford, J. A. Hoagland, and J. M. McAlerney, "Practical automated detection of stealthy portscans," *Journal of Computer Security*, vol. 10, no. 1, pp. 105–136, 2002.

[15] C. Gates, J. J. McNutt, J. B. Kadane, and M. I. Kellner, "Scan detection on very large networks using logistic regression modeling," in *Computers and Communications, 2006. ISCC'06. Proceedings. 11th IEEE Symposium on*. IEEE, 2006, pp. 402–408.

[16] C. Leckie and R. Kotagiri, "A probabilistic approach to detecting network scans," in *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*. IEEE, 2002, pp. 359–372.

[17] J. Treurniet, "A network activity classification schema and its application to scan detection," *Networking, IEEE/ACM Transactions on*, vol. 19, no. 5, pp. 1396–1404, 2011.

[18] C. Muelder, K.-L. Ma, and T. Bartoletti, "Interactive visualization for network and port scan detection," in *Recent Advances in Intrusion Detection*. Springer, 2006, pp. 265–283.

[19] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Security and Privacy, 2004. Proceedings. 2004 IEEE Symposium on*. IEEE, 2004, pp. 211–225.

[20] B. Soniya and M. Wiscy, "Detection of TCP SYN scanning using packet counts and neural network," in *Signal Image Technology and Internet Based Systems, 2008. SITIS'08. IEEE International Conference on*. IEEE, 2008, pp. 646–649.

[21] J. Cannady, "Artificial neural networks for misuse detection," in *National information systems security conference*, 1998, pp. 368–81.

[22] J. Li, G.-Y. Zhang, and G.-C. Gu, "The research and implementation of intelligent intrusion detection system based on artificial neural network," in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 5. IEEE, 2004, pp. 3178–3182.

[23] C. Gates, "Co-ordinated port scans: a model, a detector and an evaluation methodology," 2006.

[24] ——, "Coordinated Scan Detection." in *NDSS*, 2009.

[25] M. G. Kang, J. Caballero, and D. Song, "Distributed evasive scan techniques and countermeasures," in *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2007, pp. 157–174.

[26] "Scanrand," https://www.sans.org/security-resources/idfaq/scanrand.php.

[27] "Unicorn Scan," http://www.unicornscan.org/.

[28] "Zmap," https://zmap.io/.

[29] "MASSCAN: Mass IP port scanner," https://github.com/robertdavidgraham/masscan.

[30] M. Morbitzer, "TCP Idle Scans in IPv6," Master's thesis, Radboud University Nijmegen, The Netherlands, 2013.

[31] 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.

[32] Z. Qian, Z. M. Mao, and Y. Xie, "Collaborative TCP sequence number inference attack: how to crack sequence number under a second," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12. New York, NY, USA: ACM, 2012, pp. 593–604. [Online]. Available: http://doi.acm.org/10.1145/2382196.2382258

[33] 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.

[34] S. M. Bellovin, "A technique for counting NATted hosts," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. ACM, 2002, pp. 267–272.

[35] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 93–108, 2005.

[36] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-path tcp exploits: Global rate limit considered dangerous," in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, pp. 209–225.

[37] A. Quach, Z. Wang, and Z. Qian, "Investigation of the 2016 linux tcp stack vulnerability at scale," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 1, no. 1, p. 4, 2017.

[38] Y. Gilad and A. Herzberg, "Spying in the dark: Tcp and tor traffic analysis," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2012, pp. 100–119.

[39] ——, "Fragmentation considered vulnerable: blindly intercepting and discarding fragments," in *Proceedings of the 5th USENIX conference on Offensive technologies*. USENIX Association, 2011, pp. 2–2.

[40] ——, "Off-path tcp injection attacks," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 4, p. 13, 2014.

[41] 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

[42] 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

[43] 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