Internet-Scale IPv4 Alias Resolution with MIDAR: System Architecture

CAIDA Tech Report, May 27, 2011

Ken Keys, Young Hyun, Matthew Luckie, and k claffy

***Abstract*—A critical step in creating accurate Internet topology maps from traceroute data is mapping IP addresses to routers, a process known as alias resolution. Recent work in alias resolution inferred aliases based on similarities in IP ID time series produced by different IP addresses. We design, implement, and experiment with a new tool that builds on these insights to scale to Internet-scale topologies, i.e., millions of addresses, with greater *precision* and *sensitivity*. MIDAR, our Monotonic ID-Based Alias Resolution tool, provides an extremely precise ID comparison test based on monotonicity rather than proximity. MIDAR integrates multiple probing methods, multiple vantage points, and a novel sliding-window probe scheduling algorithm to increase scalability to millions of IP addresses. Experiments show that MIDAR's approach is effective at minimizing the false positive rate sufficiently to achieve a high *positive predictive value* at Internet scale. We provide sample statistics from running MIDAR on over 2 million addresses, and validate these results against available ground truth. Tools such as MIDAR can help preserve longitudinal history of the Internet's topological evolution.**

I. Introduction

VARIANTS of the traceroute tool [1] are widely used for discovering Internet topology [2]–[5]. Traceroute shows the sequence of router interfaces on the path from the source to the destination, and executing traceroute from multiple sources to multiple destinations reveals many router interfaces and allows us to infer links between them. A router by definition has at least two interfaces; Internet core routers often have dozens. Alias resolution is the process of identifying which interface IP addresses belong to the same routers and is required to convert the IP-level topology discovered by traceroute to a more useful router-level topology [6], [7].

There are many alias resolution techniques and implementations available [8]. The Mercator technique [6], [9], [10] identifies aliases by sending a probe packet to one address and getting a response from a different address. Ally [4] infers that a pair of addresses are aliases if probe packets sent to them produce responses with IP ID values in the correct order. Spring [11] described techniques for drawing alias inferences from similarities in reverse DNS lookups, and from simple analysis of traceroute graphs. APAR [12], [13] and kapar [8] use more sophisticated graph analysis techniques to infer subnets linking routers, and from that, aliases. DisCarte [14] infers aliases from analysis of a graph created from combined traceroute and Record Route data. RadarGun [15] looks for similarities in IP ID time series collected from many addresses. Sherry [16] describes the use of the IP prespecified timestamp option to infer aliases.

In this paper, we introduce MIDAR, our Monotonic ID-Based Alias Resolution tool, an IP ID-based alias resolution technique inspired by Ally and RadarGun. An *IP ID value* is a 16-bit number stored in the IP ID field in the IP header, which the sender of a packet sets to some unique value so that the recipient can identify and reassemble fragmented packets. For alias resolution purposes, we are concerned with the IP ID values of packets originated by a router in the control plane, rather than packets forwarded by a router in the data plane. Routers themselves can send packets, for example, by responding to ping or traceroute; by running BGP or NTP; and by providing NetFlow, SNMP, or remote terminal access. There is no standard method for generating IP ID values, but many routers maintain a simple IP ID counter that is incremented for packets it generates and which *wraps* from 65 535 to 0. The key observation is that if a router uses a *shared IP ID counter* for generating IP ID values, then the router will use consecutive IP ID values when sending consecutive packets no matter which interface address it uses as the source address. Thus, if two addresses share a counter, then they are conclusively aliases, and their *IP ID time series*, a sequence of IP ID values collected over time, will have similar values in a given measurement period and will form a monotonically increasing IP ID sequence when merged together, except during counter wraps. The latter expresses the *monotonicity requirement*, a necessary condition for two time series to be derived from a shared counter. IP ID-based alias resolution techniques infer aliases by analyzing the IP ID values in response packets and inferring which interface addresses use a shared counter. RadarGun infers a shared counter by looking for similar time series values, whereas Ally and MIDAR infer a shared counter by checking for the monotonicity requirement, though in different ways.

Most routers seem to use a single IP ID counter shared across all interfaces and protocols, but any IP ID based alias resolution technique must account for those that do not. A small subset of routers sets the IP ID to zero or some other constant value, a random value, or the value used in the probe packet [15]. Such non-counter IP ID values can be detected and excluded from IP ID-based alias resolution. Another small subset of routers appears to use separate IP ID counters

K. Keys, Y. Hyun, and k claffy are with the Cooperative Association for Internet Data Analysis (CAIDA), University of California, San Diego.

M. Luckie is with the University of Waikato.

This project is sponsored by the U.S. Department of Homeland Security (DHS) Science and Technology (S&T) Directorate.

for each interface. Use of such unshared IP ID counters is undetectable from an analysis of IP ID values alone. Because two addresses may be aliases but not share an IP ID counter, IP ID-based techniques can not find all alias pairs, and cannot definitively conclude that two addresses that do not share a counter are not aliases. Thus, IP ID-based techniques can produce three results: 1) *positive* shared counter and *positive* aliases; 2) *negative* shared counter and *inconclusive* aliases; 3) *inconclusive* shared counter and *inconclusive* aliases.

Because of the limited precision of IP ID values and the limited variation in observed rates of change, or *velocities*, of IP ID counters (see Appendix A), it is inevitable that any large collection of addresses will have many pairs of addresses with similar or aligned IP ID time series out of sheer coincidence, as predicted by the birthday paradox and the pigeonhole principle. Suppose a given alias resolution technique has a *false positive rate* of α based on how much similarity or alignment between two time series it accepts as indicating a shared counter. Then for N addresses, we can expect $O(N)$ true positives (TP) and $O(\alpha \times N^2)$ false positives (FP) (see Appendix C). When $N > 10^6$, as in the case of Internet-scale alias resolution, true positives (aliases) are extremely rare, approximately 1 in a million; that is, the *prevalence* of aliases is extremely low. Hence, unless the false positive rate is extremely low, $\alpha \ll 1/N = 10^{-6}$, false positives can overwhelm true positives, and the alias technique will not be useful for reliably identifying aliases. We can quantify the degree of usefulness with the *positive predictive value* (PPV) metric, which specifies the fraction of positive test results—"shared counter" and thus "alias"—that are correct; that is, $\mathrm{TP}/(\mathrm{TP}{+}\mathrm{FP})$. Another important metric for evaluating IP ID-based techniques is *sensitivity*, the fraction of cases of counter sharing that produce a positive test result; that is, $\mathrm{TP}/(\mathrm{TP} + \mathrm{FN})$, where FN stands for false negatives. For the purposes of comparing the effectiveness of IP ID-based techniques, a false negative only means a technique failed to detect counter sharing when sharing was present, which is narrower in scope than a definition based on a failure to detect aliases. (See Appendix B for further discussion of these and related terminology.)

There are two main challenges for an alias resolution technique as the number of addresses N increases: 1) probing and testing the $O(N^2)$ candidate alias pairs, and 2) minimizing the false positives relative to true positives; that is, ensuring a high PPV. The Ally technique requires $O(N^2)$ probes to test all possible pairs of addresses. RadarGun avoids Ally's scalability problems by probing an entire set of addresses as a whole with $O(N)$ probes, but scalability is still limited by a need to obtain overlapping time series from all addresses. Neither Ally nor RadarGun has a sufficiently low false positive rate to handle the millions of addresses that appear in macroscopic-scale Internet topology graphs such as that collected by Ark [17]. Repeating the alias tests of Ally and RadarGun can lower the overall false positive rate and thus increase the PPV, but because these tests suffer from false negatives, repetition can also decrease the sensitivity, causing aliases to be missed.

MIDAR is an attempt to overcome these and other limitations of Ally and RadarGun and to scale to millions of IPv4 addresses, the order of topology graph sizes observed with Ark. In a nutshell, MIDAR collects IP ID time series data from many different vantage points, then mines the data using our Monotonic Bounds Test (Section III-B) to discover which IP addresses are likely aliases to the same router. The key features of MIDAR are the *Monotonic Bounds Test* (MBT), an ID comparison test with near perfect sensitivity based on monotonicity rather than proximity, which allows MIDAR to achieve an extremely low false positive rate and thus a high PPV; the use of multiple probing methods to increase the responsiveness of targets and thus extend the coverage of IP ID-based techniques; and the use of multiple vantage points and a novel sliding-window scheduling algorithm to achieve probing scalability.

This paper is organized as follows. Sec. II provides background on the features and limitations of the two best known IP-ID based approaches: Ally and RadarGun. Sec. III presents the essential concepts and components of MIDAR. Sec. IV reviews our implementation of MIDAR, including four stages of probing: Estimation, Discovery, Elimination (of false positives), and Corroboration. Sec. V reports results from a preliminary Internet-scale experiment with MIDAR, and Sec. VII summarizes our contributions and plans to integrate MIDAR into a larger system for alias resolution.

II. Existing IP ID Techniques and Limitations

A. Ally

The Ally component of Rocketfuel was the first tool to examine IP ID values for alias resolution. Several papers describe the Ally alias resolution tool [4], [11], [15]. We base our description on Bender et al. [15] and on the Ally source code included in the latest distribution of Scriptroute (v0.4.8) [18] (earlier standalone releases of Ally are now deprecated). The Ally implementation can send probes using UDP (default), TCP ACK, or ICMP. The user chooses the single probing method to use with a command-line option.

One of the key steps of Ally is checking whether the IP ID values of two candidate addresses are *in order*; that is, the values form an increasing sequence consistent with the use of a shared counter. Because IP ID counters can wrap from 65 535 to 0, Ally must use sequence space arithmetic, similar to that defined in RFC 1982 [19]. We will use the notation $X \prec Y$ to denote the *less than* relationship within sequence space.

Ally uses the following procedure to test whether addresses A and B are aliases. First, Ally sends a probe to A, waits 1 ms, then sends a probe to B. Suppose the IP ID values in the responses are A_1 and B_1, respectively. Ally can match responses to probes, so there is no ambiguity if the responses arrive out of order in time. Ally first checks whether A_1 and B_1 are in order and close enough to each other; namely, that $A_1 - 10 \prec B_1 \prec A_1 + 200$. If so, then Ally waits 400 ms, sends a probe to B, waits 1 ms, and sends a probe to A. Ally then checks that the resulting IP ID values B_2 and A_2 meet the condition $B_2{-}10 \prec A_2 \prec B_2{+}200$ and that $A_1 \prec A_2$ and $B_1 \prec B_2$. If all these conditions are met, then Ally declares A

and B to be aliases; otherwise, they are declared to be "non-aliases," but like all IP ID-based techniques, Ally can only infer that they do not share a counter.

Because Ally cannot know nor control the exact generation time of each collected IP ID value, Ally uses a margin of error when comparing IP ID values to accommodate uncertainties. The $A_1 - 10$ margin is to accommodate reordering of probe packets along the forward path, which would cause IP ID values to be generated in the order (B_1, A_1), with $B_1 \prec A_1$. The $A_1 + 200$ margin is to accommodate the advancement of the IP ID counter during the inter-arrival time of the probe packets at their destinations. For the most accurate results, Ally requires IP ID values to be sampled closely in time, but probe packets will typically undergo a certain amount of dispersion beyond the 1 ms separation they had when sent, due to cross traffic, differences in routing (for example, if A and B reside in different prefixes), load-balanced paths with different lengths, and other causes. The greater the packet dispersion, the greater the potential counter advancement between collected IP ID samples.

Ally has the following limitations. First, it is unclear whether these empirically-derived margins of error ($X-10$ and $X+200$) are universally applicable to typical packet dispersion amounts and counter velocities. Second, using fixed margins of error is a fragile balancing act between minimizing false positives and false negatives. The wider these margins are, the more they allow false positives from chance alignments of IP ID values. However, if these margins are too narrow, then they can lead to false negatives if counters advance at a high rate or in bursts, or if probe packets undergo a significant amount of dispersion. Third, Ally relies on only four IP ID samples to infer aliases, which makes Ally susceptible to false positives caused by chance alignments, independent of the margins of error. Fourth, Ally cannot perform IP ID-based alias resolution on a router that rate limits its responses, because Ally needs the responses to be generated closely in time.

Finally, a significant drawback of the Ally technique is that, given N addresses, it requires $O(N^2)$ probes to test all possible pairs. To make Ally more practical, some heuristics are needed to reduce the size of the search space. For example, Rocketfuel considered a pair of addresses as candidates for testing with Ally if both addresses are a similar hop distance away from each of several vantage points (but the hop distance can be different across vantage points). Although this heuristic significantly reduces the amount of testing needed with Ally, the reduction is not enough for practical use on the millions of addresses that appear in macroscopic-scale Internet topology graphs. Also, any pruning heuristic carries the risk of excluding some candidate pairs that would otherwise have been identified as aliases (for example, aliases that are in different prefixes may be routed along paths of different lengths from a vantage point).

Even if it were possible to apply Ally to one million addresses, Ally's false positive rate (α) would be too high to produce a useful positive predictive value. The margins of error in Ally's test allows samples to be 210 ID values apart, or $210/65536 = 0.32\%$ of the ID space. The two halves of the test are closely correlated because they are taken only 400 ms apart and velocities are typically low (Fig. 7), suggesting α is only slightly lower than 0.0032. But even if the two halves of the test were completely independent, α would be at best about 0.00001. Extrapolating these rates to one million target addresses suggests there would be at least 5 million false positives, and probably closer to 1.6 billion, which is orders of magnitude more than the expected 1 to 10 million true positives.

B. RadarGun

We base our discussion of RadarGun on the RadarGun v0.3 source code [20] in addition to the RadarGun paper [15]. The RadarGun implementation can send probes using TCP ACK (default), UDP, or both. When both protocols are used, RadarGun sends the TCP ACK and UDP probes consecutively for the same address and analyzes the resulting data independently without regard for their common destination address.

RadarGun avoids Ally's scalability problems by probing an entire set of addresses as a whole, $O(N)$ probes, rather than as individual pairs, $O(N^2)$ probes. RadarGun makes 30 probing passes through the address list to obtain 30 IP ID samples from each address, with samples from different addresses interleaved with each other; for example, given addresses A, B, C, RadarGun takes the samples $A_1, B_1, C_1, A_2, B_2, C_2, \ldots$. This probing scheme produces an IP ID time series for each address. An IP ID time series A (for address A) consists of a sequence of samples $\{A_i\}$, where each A_i specifies the sample time and the IP ID, (t_i, ID_i). RadarGun uses the measured receive time of a response packet as an approximation of the sample time, since it does not know exactly when a router generated a given IP ID sample. RadarGun discards a time series as unusable if (1) fewer than 25% of the 30 probes elicited responses (that is, RadarGun has fewer than 7 IP ID samples), (2) all collected samples have an IP ID of zero or all have the IP ID used by probes, or (3) the time series is *nonlinear*—that is, either the IP ID counter is advancing too quickly to measure, or IP ID values are randomly generated.

RadarGun considers a time series to be nonlinear if either of two conditions is met. The first condition is based on the frequency of *negative deltas* in a time series. A *delta* is the difference in value of adjacent IP ID samples in a time series; that is, $\Delta ID_i = ID_{i+1} - ID_i$. A *negative delta* is when $\Delta ID_i < 0$. Negative deltas occur naturally as an IP ID counter wraps from 65 535 to 0. For any given sampling interval, the faster an IP ID counter advances, the more frequently a negative delta will appear in a time series, since the counter can wrap more often within the time period sampled by RadarGun. Negative deltas can also occur when IP ID values are generated randomly, since the average probability of an *individual* delta being negative is 50% in a sequence of random values. Regardless of the exact cause, whether too fast a counter or random IP ID values, RadarGun discards a time series as nonlinear if greater than 30% of the deltas are negative.

The second condition for nonlinear time series is based on the apparent rate of advancement, or velocity, of an IP ID counter. RadarGun computes the velocity from an *unwrapped*

IP ID time series. A time series is typically unwrapped by adding 65 536 (the full IP ID space) whenever a negative delta occurs. RadarGun also tries to account for one or more counter wraps that may have occurred in large gaps in time between samples, even when the delta is positive. RadarGun infers the number of possible wraps in each gap from an estimate of the time between wraps, Δt_{wrap}, derived from a simplistic calculation on the positive deltas in a time series. For a gap of duration Δt_{gap}, there are $\lfloor \Delta t_{\text{gap}} / \Delta t_{\text{wrap}} \rfloor$ inferred wraps, and RadarGun adds this many multiples of the IP ID space when unwrapping the time series. Once a time series has been unwrapped, RadarGun computes the velocity by finding the linear least squares line that best fits the unwrapped IP ID values; the slope of the line is the velocity in ID/s. The more negative deltas there are in a time series, the higher the apparent velocity of the unwrapped samples. Therefore, both true high velocity counters and random IP ID values will lead to high apparent velocities. RadarGun discards a time series as nonlinear if the velocity is greater than 800 ID/s, regardless of the cause.

The key insight of RadarGun is that if two addresses share an IP ID counter, then their time series should have nearby IP ID values when overlapping in time. RadarGun infers whether two addresses are aliases by employing a *distance test* to measure how closely their time series describe the same underlying counter. The key building block of the distance test is the calculation of the *sample distance*, the distance between an individual sample point in one time series and the expected value of the IP ID counter in the other time series at the same moment in time. There are two cases to computing the sample distance, with all calculations performed on the unwrapped time series A and B of the two addresses being tested. Let $B_j = (t_{B,j}, ID_{B,j})$ be a sample of B. In the first (and more common) case, B_j lies between adjacent samples of A in time, that is, there is some i for which $t_{A,i} \leq t_{B,j} < t_{A,i+1}$. RadarGun then uses linear interpolation between A_i and A_{i+1} to estimate $ID_{A,\text{est}}$, the expected IP ID value of A at $t_{B,j}$. In the second case, B_j does not lie between any two samples of A, and RadarGun extrapolates the best fit line through A (the same line used to calculate the velocity) to estimate $ID_{A,\text{est}}$. In either case, RadarGun then computes the sample distance $\delta_{B,j} = |ID_{A,\text{est}} - ID_{B,j}|$. After computing all sample distances $\delta_{B,j}$ between B and A, and similarly computing the sample distances $\delta_{A,i}$ between A and B, RadarGun calculates the average sample distance:

$$\Delta_{A,B} = \frac{\sum_i \delta_{A,i} + \sum_j \delta_{B,j}}{|A| + |B|}$$

If $\Delta_{A,B} < 200$, then RadarGun concludes A and B are aliases; if $\Delta_{A,B} > 1000$, then they are not aliases. Otherwise, the distance test is inconclusive.

By employing the distance test on time series, RadarGun is more tolerant than Ally of ICMP rate limiting and less susceptible to false positives caused by chance alignments. However, RadarGun's distance test has several weaknesses. First, the distance thresholds for aliases and non-aliases are derived empirically and subjectively from a particular dataset and may not apply to other datasets. Second, even for a particular dataset, there is no inherently right choice for the distance thresholds, since the thresholds must be low to minimize false positives with lower velocity addresses, and high to minimize false negatives with higher velocity addresses. Even if the thresholds were chosen adaptively to velocities, the thresholds must still have margins of error to allow for bursty IP ID counter advancement and other uncertainties, which prevents exact separation of aliases from non-aliases. As a result, adjusting the distance thresholds never fully eliminates false positives, false negatives, or inconclusive results, but merely shifts the balance between them. Third, there can always be a false positive under the distance test regardless of the threshold used, as Fig. 1 shows, because the distance test does not check for the monotonicity requirement.

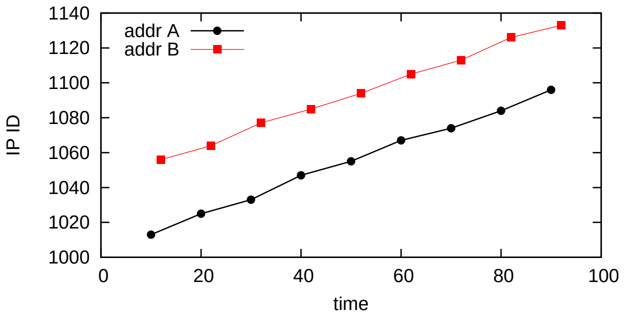

Fig. 1. There can always be a false positive under the RadarGun distance test regardless of the threshold used. For example, the average IP ID distance between these addresses is only 40, which is below the 200 threshold for being aliases, but these addresses cannot share a counter because the merged sample points do not form a monotonic sequence over time. A similar construction exists for any threshold.

As a consequence of the above weaknesses, the distance test produces too many false positives for RadarGun to scale to millions of addresses. For example, extrapolating the false positive rate (0.0005) implied in Bender et al. [15] to one million target addresses suggests there will be an order of magnitude more false positives (264 million pairs) than true positives, giving a very poor PPV.

Furthermore, because RadarGun needs to obtain overlapping samples from all addresses in order to apply the distance test, there is a practical limit to the number of addresses RadarGun can handle before requiring network-unfriendly levels of probing bandwidth. For example, handling one million targets with 10 s probe spacing would require 100 000 packets per second, or 35 Mb/s.

III. MIDAR Design

To find aliases among a large number N of router addresses, MIDAR collects an IP ID time series from each of the addresses and tests for a shared IP ID counter in each of the $O(N^2)$ address pairs. We take a bottom-up approach to describing MIDAR. In this section, we describe the essential concepts and key features of MIDAR, and discuss our approach to mitigating false positives. In the following section we will describe how we integrate these components into the complete MIDAR system.

A. Time series in MIDAR

MIDAR takes a sampling of IP ID values to construct the time series used for alias resolution. MIDAR considers a time series *unusable* for alias resolution if (1) fewer than 75% of the probes to the target elicited responses, (2) 25% or more of the collected samples have a constant IP ID value (such as zero) or echo the IP ID used by probes, or (3) the time series cannot be modeled as a monotonically increasing sequence; that is, the observed frequency of negative deltas is so high that we are not confident we are detecting all wraps of the IP ID counter, or we suspect the IP ID values are randomly generated.

To reliably detect all counter wraps, we must use a sampling interval that is shorter than the time between wraps, so that we obtain exactly one negative delta whenever the counter wraps and positive deltas at all other times. Sampling even more frequently will yield more positive deltas during the time the counter is increasing monotonically but still only one negative delta at each counter wrap, so the overall fraction of deltas that are negative will decrease. We adopt RadarGun's 30% threshold on the maximum allowed fraction of negative deltas before we consider a time series unusable. This 30% threshold is intentionally more conservative than the 50% threshold suggested by the Nyquist-Shannon sampling theorem when a counter wrap is thought of as a "signal" occurring at a certain frequency. The 30% limit on negative deltas also has the advantage of excluding 98.8% of random time series, which cannot be used for alias resolution (see Appendix G).

We define the *maximum sampling interval* I_{max} to be the largest sampling interval that still ensures the fraction of negative deltas is no more than 30%. MIDAR collects an initial time series from each target address using a small fixed sampling interval and then calculates I_{max} individually for each target based on the target's observed velocity. MIDAR uses the computed I_{max} to customize the sampling interval individually for each target when collecting additional time series actually used for alias resolution (Sec. III-E).

Observe that limiting the fraction of negative deltas to 30% is equivalent to limiting the average counter advancement per sample to 30% of the ID space, because the counter advances through 100% of the ID space between each counter wrap. Hence, the maximum sampling interval for a time series with velocity $\bar{v}$ is

$$I_{\text{max}} = (0.3 \times 2^{16})/\bar{v} \tag{1}$$

We define the velocity $\bar{v}$ of a time series to be the average slope of the segments weighted by segment duration; that is,

$$\bar{v} = \frac{\sum \Delta ID_i}{\sum \Delta t_i} \tag{2}$$

where ΔID_i and Δt_i are the change in ID and time, respectively, between samples i and $i+1$. If there is a negative delta between samples i and $i+1$, then we define $\Delta ID_i = ID_{i+1} + 2^{16} - ID_i$; that is, the distance between the *unwrapped* ID samples at negative deltas. To avoid distortions due to sampling gaps or atypical counter behavior, we exclude discontinuities (Appendix H) when calculating velocity.

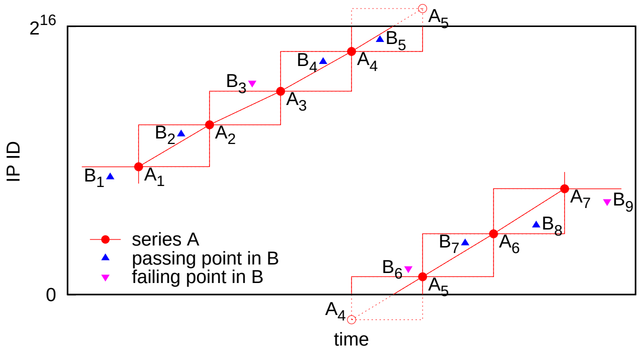

Fig. 2. Illustration of the sample-wise execution of the Monotonic Bounds Test (MBT). MBT checks an IP ID sample of one time series (e.g., B_2) against the closest surrounding samples (in time) of the other time series (e.g., A_1 and A_2). The tested sample must lie within the monotonic bounds set by the surrounding samples (e.g., B_2 must fall within the bounding box with corners at A_1 and A_2). When there is a counter wrap between the surrounding samples (e.g., between A_4 and A_5 when checking B_5), the monotonic bounds split into two parts. (Velocity is exaggerated for clarity.)

B. Monotonic Bounds Test

The Monotonic Bounds Test (MBT) checks whether the IP ID times series of two addresses meet the monotonicity requirement, a necessary condition for sharing an IP ID counter; that is, whether the time series form a monotonically increasing IP ID sequence when considered as a single merged time series. The MBT is a rigorous test that does not employ ad hoc thresholds to accommodate uncertainties.

MBT checks that two time series A and B meet the monotonicity requirement by individually checking that each sample of B meets the monotonicity requirement with respect to the samples of A, and that each sample of A meets the requirement with respect to the samples of B. If all sample tests pass, then A and B as a whole meet the monotonicity requirement. We first describe the sample-wise execution of MBT in a slightly simplified form and then provide the full details. Let $B_j = (t_{B,j}, ID_{B,j})$ be a sample of B. Suppose we are checking that B_j meets the monotonicity requirement with respect to the samples of A. Let $A_i = (t_{A,i}, ID_{A,i})$ and $A_{i+1} = (t_{A,i+1}, ID_{A,i+1})$ be adjacent samples in A such that $t_{A,i} < t_{B,j} < t_{A,i+1}$; that is, A_i and A_{i+1} are the nearest adjacent samples of A in time to B_j. Fig. 2 illustrates the two different MBT cases. In the first case, the counter has not wrapped between the samples A_i and A_{i+1} (that is, $\Delta ID_{A,i} > 0$), and so we can simply check that $ID_{A,i} < ID_{B,j} < ID_{A,i+1}$. We can imagine this constraint on the ID values as a bounding box whose corners are defined by A_i and A_{i+1}, and B_j must fall within this box to pass MBT. For example, in Fig. 2, B_2 lies between A_1 and A_2 in time and falls within the bounding box of these samples, and thus B_2 meets the monotonicity requirement. In contrast, B_3 is between A_2 and A_3 in time but does not fall within the bounding box (because $ID_{B,3} \not< ID_{A,3}$) and thus violates the monotonicity requirement. In the second MBT case, the counter has wrapped between A_i and A_{i+1} (that is, $\Delta ID_{A,i} < 0$). Therefore, the bounding box between A_i and A_{i+1} is split into two pieces, and we must have either

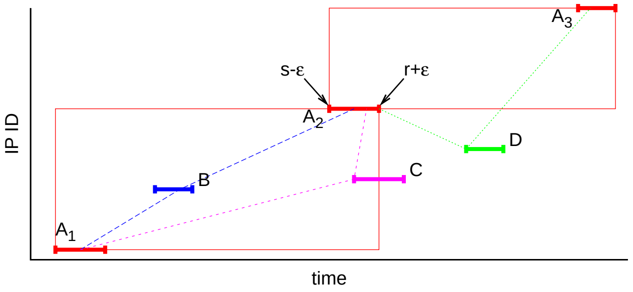

Fig. 3. Monotonic Bounds Test with imperfect time data. *Sample time ranges* are shown as horizontal bars (with exaggerated size for clarity). Samples B and C lie at least partially within the bounding boxes, making it possible to draw a monotonic curve (dotted lines) through them connecting neighboring samples from interface A, showing that samples B and C may come from a counter shared with A. Sample D lies completely outside the bounding boxes, so no monotonic curve can connect samples in A and still pass through D, so D cannot come from a counter shared with A.

$ID_{A,i} < ID_{B,j}$ or $ID_{B,j} < ID_{A,i+1}$. For example, B_5 lies between A_4 and A_5 and passes, since $ID_{A,4} < ID_{B,5}$. B_6 also lies between these samples but violates the monotonicity requirement by lying outside both pieces of the bounding box. If all samples of B pass, then MBT swaps the roles of A and B and individually checks the samples of A against B with the same procedure. If any sample-wise test fails, we can immediately conclude that that A and B do not share a counter without performing the remaining sample-wise tests.

So far, we have described a time series as being $\{(t_i, ID_i)\}$ with t_i being the sample time. In order for MBT to maintain a virtually zero false negative rate (a crucial property relied on in MIDAR), MBT needs accurate sample times to determine which samples define the monotonic bounds for each sample-wise test. The *true sample time*, τ_i, is the exact moment in time that a router generated a response with the given IP ID value. We cannot determine τ_i with active measurement, but we can calculate accurate bounds on τ_i. We know the measured time s_i when we sent our probe and the measured time r_i when we received the response, and that the true send and receive times are within $\pm\epsilon$ of the measured times, where ϵ is the maximum clock error of all monitors during a MIDAR run (see Sec. III-D). Since the response must have been generated between sending and receiving, we know that the true sample time must be within the *sample time range* ($s_i - \epsilon$, $r_i + \epsilon$), which we will substitute for τ_i in MBT execution.

MIDAR obtains the samples of a single time series sequentially by sending the next probe only after receiving the response to a prior probe or after a timeout, so there is never any uncertainty about the ordering of the samples within a single time series. However, since MIDAR probes multiple interfaces in parallel (Sec. III-E), two samples from two separate time series can have overlapping time ranges, and consequently the true relative ordering of these samples is uncertain. When the time range of one of the bounding samples overlaps with the time range of the test sample, the MBT widens its bounds to the next closest sample whose time range does not overlap with the test sample. This makes the monotonic bounds larger than they could have been if the exact relative ordering of samples were known, but the sensitivity of the test is preserved despite these uncertainties—a test failure against the larger bounds conclusively means that there is no shared counter. We can thus accommodate uncertainties in both the response time and clock error without compromising the rigor of MBT.

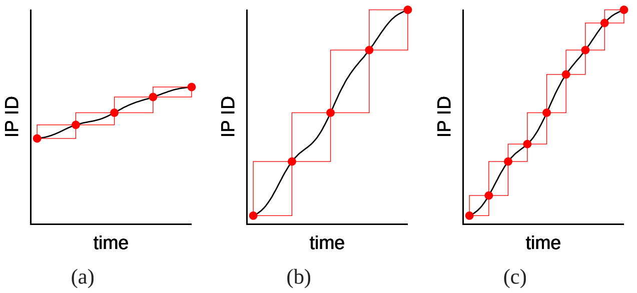

Fig. 4. With all other things being equal, monotonic bounds (that is, the range of IP ID values allowed by the MBT) become tighter when a time series has a lower velocity (subfigure (a) compared to (b)) or when IP ID values are sampled on shorter intervals ((c) vs. (b)).

Fig. 3 illustrates the execution of MBT using sample time ranges. We wish to individually test the samples B, C, and D against the surrounding samples of $\{A_i\}$. The time ranges of B and D (shown as horizontal bars) do not overlap with the time ranges of A_i, so we know the true relative ordering of these samples, and MBT execution is straightforward. For B, the monotonic bounds are defined by A_1 and A_2, the nearest adjacent samples of A_i that do not have overlapping time ranges with B, and B passes. For D, the monotonic bounds are defined by A_2 and A_3, and D fails since it lies outside the bounding box. Sample C is the interesting new case, since the time ranges of C and its nearest surrounding sample A_2 overlap. Because of the overlap, we cannot know whether C precedes A_2 and therefore should be bounded by A_1 and A_2 (the left bounding box), or whether C follows A_2 and should be bounded by A_2 and A_3 (the right bounding box). MIDAR simply avoids relying on the indeterminate A_2 and looks outward toward A_1 and A_3, which are the next nearest samples that do not overlap with C, to find the suitable monotonic bounds to use. C falls within this larger bounding box (that is, $ID_{A,1} < ID_C < ID_{A,3}$) and therefore passes. The dashed line passing through A_1, C, and A_2 illustrates a possible monotonic counter consistent with these sample values and time ranges.

Because the MBT is based strictly on the definition of monotonicity, we must detect and account for discontinuities (Appendix H) and other anomalies (Appendix I) that occur occasionally in time series that otherwise adhere to the definition. Whenever a sample-wise MBT test involves an ID value that is questionable due to a discontinuity or anomaly, the test may generate a false negative. Rather than risk this rare error, we do not apply the MBT in that case, and rely on the remaining sample-wise tests for the most accurate result.

In general, the more ID samples we have available to test with MBT, and the tighter the monotonic bounds, the more confident we can be that a positive test result means a shared

TABLE I
SUMMARY OF PROBING METHODS.

Method	Probe Packet	Expected Response
TCP	TCP ACK to port 80 on target	TCP RST or (rarely) ICMP *port unreachable* from target
UDP	UDP packet to port 33435 on target	ICMP *port unreachable* from target
ICMP	ICMP *echo request* to target	ICMP *echo reply* from target
Indirect	TTL-limited ICMP *echo request* to a host *past* the target	ICMP *time exceeded* from target

counter. Although monotonic bounds can be large in theory, they are typically small in practice for two reasons. First, monotonic bounds are defined separately by each pair of samples and are by construction as tight as possible. Thus, the lower velocity time series in Fig. 4a has tighter bounds than the higher velocity time series in Fig. 4b. Low velocity time series make up the majority of the cases observed in our data, and the monotonic bounds can be quite small; for example, adjacent samples with ID values 5 and 7 define monotonic bounds that can be satisfied by only a single ID value, 6. Second, we can use a shorter sampling interval to tighten the bounds independently of the target velocity, as illustrated by Fig. 4b and 4c, which have identical counters but different probe spacing. To the extent possible, MIDAR tries to keep monotonic bounds small by adapting probe spacing to the actual measured velocity of each target interface (Sec. III-E).

C. Multiple probing methods

Recent implementations of Ally and RadarGun offer a choice of several probing methods selectable by the user (typically only one method per run). Bender et al. [15] mentioned the possibility of combining multiple methods (they suggest TCP and UDP) in order to increase the number of targets with usable IP ID samples. They did not offer any procedure, nor investigate the usefulness or effectiveness of combining methods. In this section, we describe the procedure used by MIDAR to fully exploit four probing methods—TCP, UDP, ICMP, and a method we call *TTL-limited indirect probing*, or *Indirect* for short.

Table I summarizes the methods supported by MIDAR. The TCP, UDP, and ICMP methods are straightforward: send a probe packet to the target, and if the response is of the expected type, collect the IP ID value. Although UDP responses from a different address are often from a different interface on the same router, there is a risk that such responses are from a different router altogether, so we do not use them in MIDAR; interpreting these responses is more in the domain of the Mercator technique. The *Indirect* method imitates a traceroute measurement. Every intermediate address in a traceroute path responded with an ICMP *time exceeded* response, so in theory, we can elicit a *time exceeded* response again by reproducing the exact conditions of a traceroute measurement. For an address observed at hop h in a traceroute path, the *Indirect* method sends a probe with a TTL of h from the original vantage point to the original destination and obtains an IP ID sample from the *time exceeded* response. To maximize the chances of the probe taking the same route as the original traceroute packet and expiring at the target address, we maintain the same Paris-traceroute flow label (TOS, protocol, source and destination addresses, ICMP type and code, and ICMP checksum [21]) as the original traceroute measurement. Nevertheless, the route can still change, and we may face either (1) a new route to the destination that entirely bypasses the target address, or (2) a new route that still passes through the target address but at a different hop distance. MIDAR does not currently handle the first case—this is the greatest weakness of *Indirect* probing. MIDAR handles the second case by hunting for the target at nearby TTLs. If an *Indirect* probe with TTL $= T$ does not elicit the expected response, we send additional probes with TTL $= T \pm 1$. If we find the target at one of these TTLs, we use that new TTL as the expected TTL for subsequent probes. MIDAR performs this *TTL expansion* process only in the Estimation stage (see Sec. IV-A). In our experiments, TTL expansion increased the response rate to *Indirect* probing from 76.5% to 80.8%. Expanding further to $T \pm 2$ provided only a negligible increase in the response rate while significantly increasing the probing cost.

Appendix E describes the extent to which employing multiple probing methods increases usable time series for our dataset described in Sec. V. Using TCP alone resulted in only 34.6% of the addresses having usable time series, leaving nearly two-thirds completely unresolvable to IP ID based alias resolution. If we employ all four methods, 80.6% of addresses yield usable time series to at least one method.

The main concern with employing multiple probing methods is how consistently the interfaces on the same router behave. In the simplest case, either all or none of the interfaces of a single router respond with usable IP ID values to a given method. In this case, we can collect all IP ID samples with the same method, presenting no additional difficulty for alias inference. However, interfaces on a single router do not always behave consistently, perhaps due to different filtering on different routes to the various interfaces. In such cases, we can infer aliases only if we can meaningfully compare IP ID time series collected with different methods; that is, if a router uses the same IP ID counter to generate responses to different probing methods as well as for different interfaces. We expect that a router will use a shared counter on all interfaces when responding to TCP and UDP probes, since we expect the responses to come from a shared CPU that would execute (router-wide) services potentially reachable with these protocols. However, when we use ICMP or *Indirect* probing, the ICMP *echo reply* or *time exceeded* responses could be generated entirely on a line card (that is, on the fast path) [22], and a line card may have its own IP ID counter not shared with either the CPU or other line cards on the same router. Thus, there is a chance that responses to ICMP and *Indirect* probes may not share a counter with responses to TCP or UDP probes.

We can detect counter sharing across probing methods in the same way we identify shared counters across interfaces—we apply MBT to a pair of time series obtained from the same interface but with different probing methods. Note that these cross-method comparisons do not suffer from the high false positive rate of the cross-interface comparisons described in Appendix C, since only a single interface is involved. We observe a relatively high incidence of counter sharing for our dataset (see Sec. V), ranging from 88.9% to 97.4% of addresses per pair of methods (see Appendix F for full details).

The relatively high rate of counter sharing suggests that employing multiple methods may be a fruitful way of finding additional aliases. An obvious way to use multiple methods is to probe all targets with all methods and then perform alias inference on the resulting time series. This brute-force approach has the advantage of checking all possible combinations of addresses and methods for aliases, but it increases the difficulty of scaling up measurements, since we need overlapping time series from all methods. Furthermore, because of cross-method counter sharing, it can be redundant to probe a target with multiple methods; for example, if a target responds with a shared counter to TCP and UDP, then the time series produced by either method can be compared equally well to others for alias inference, so there is no need to collect both time series. Thus, we make a trade-off in MIDAR by probing with only one method *per* target but supporting multiple methods *across* targets. By using only one method per target, we need to send only one quarter of the probes, allowing us to probe four times more targets with the same resources, but we may not be able to find all potentially discoverable aliases if some routers do not share counters across methods, since we do not collect data using all combinations of addresses and methods.

To determine which methods are usable with each target, MIDAR probes all targets with all methods in the earliest stage (see Sec. IV-A). This process is inherently scalable since time series do not need to overlap across targets. When there are several usable methods for a given target, MIDAR selects a single method to use in subsequent stages based on the following method preferences. We prefer TCP over UDP because UDP is more often rate-limited and thus less reliable in eliciting responses. If UDP and ICMP do not appear to share a counter with each other, we prefer UDP, because ICMP is more likely to be generated on a line card using an ID counter that is not shared across interfaces. But if UDP and ICMP do share a counter, the choice of protocol does not affect the chances of cross-interface counter sharing, and we prefer ICMP because it is less likely to be rate limited than UDP. We assign *Indirect* the lowest preference because (1) it is more likely to be generated on a line card with an unshared counter, (2) it is more likely to be rate limited, (3) a routing change may prevent us from probing a target and (4) it is much more difficult to recover from the loss of a vantage point mid-run, since a target may be reachable with *Indirect* from only a limited number of vantage points (perhaps just one).

D. *Multiple vantage points*

MIDAR employs multiple vantage points to increase the aggregate probing rate, an obvious approach to scalability suggested but not implemented in [15]. Because MIDAR needs to compare time series collected by the different vantage points, their clocks must be synchronized, for example with NTP or RADclock [23], [24]. MIDAR does not require extraordinarily precise clock synchronization, but it does require an estimate of the maximum clock error ϵ across all vantage points during execution.[1] The lower the ϵ, the tighter the monotonic bounds become in the Monotonic Bounds Test (Sec. III-B), so we recommend minimizing ϵ where possible by, for example, deploying RADclock instead of NTP.

The higher probing rate achievable by multiple vantage points is not enough by itself to achieve true scalability as the number of targets increases to Internet-scale. We discuss another technique MIDAR employs for scalability in the next section.

E. *Achieving probing scalability with sliding window*

For target addresses $A, B, C, \ldots$, the simplest approach to collecting overlapping time series is to take the samples $A_1, B_1, C_1, \ldots, A_2, B_2, C_2, \ldots$; that is, loop through the target list multiple times, probing the targets in order. If we probe N addresses at p packets per second (pps), then each pass through the target list will take $I = N/p$ seconds; that is, each target is sampled at an interval of I seconds. The resulting time series for each target is usable only if the sampling interval I is less than or equal to the maximum sampling interval I_{max} for that target (see Sec. III-A). I must be short enough to accommodate the highest velocity of the N targets. Suppose the highest velocity is 2000 ID/s. Then, from Eq. 1, we must have $I \leq 9.83\,\text{s}$. If $N = 2 \times 10^6$, then to achieve $I = 9.83\,\text{s}$, we must probe at 203 459 pps, which is at least 71.6 Mb/s of traffic with TCP probes. The brute force approach of probing from 1000 machines in parallel would reduce the probing rate to 203 pps per machine, but managing that many machines is problematic. Here, we present a more scalable technique that can achieve even smaller intervals for high velocity targets, at half the per-machine probing rate, using fewer than 40 machines.

MIDAR achieves probing scalability with a *sliding window* scheduling algorithm that exploits two observations. The first observation is that, when $N \geq 10^6$, the expected $O(N)$ number of aliases is significantly smaller than the $O(N^2)$ possible pairs of addresses (see Appendix C). Therefore, collecting overlapping time series for *all* possible pairs of addresses is largely unnecessary work. If addresses have very different velocities, they cannot share a counter, so we do not need to apply the MBT to them and thus do not need their time series to overlap. That is, we can use loose velocity similarity as a high sensitivity (but low PPV) shared counter test, to filter out many unshared counter pairs at an early stage. The second observation is that target velocities range widely from near zero ID/s to several thousand ID/s, but the vast majority of the targets we have observed have low velocities (see Appendix A), so we need to use a short sampling interval only for the minority of high velocity targets.

MIDAR incrementally probes the target list over multiple *rounds*. In each round, MIDAR sends one probe to each target in a *window* in sequence. A *window* is a contiguous subset of the target list defined by starting and ending target indexes. The width and position of the window changes over time. The width of the window determines the sampling interval

[1] To estimate ϵ, we used `ntpq`/`ntpdate` to determine the clock offset and delay of each vantage point during a MIDAR run.

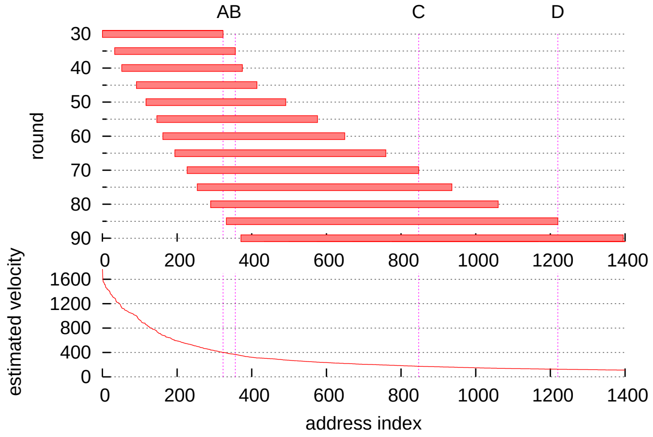

Fig. 5. Portion of sliding window schedule for a real large-scale run. The full schedule covers 272 rounds and 37 546 addresses per monitor, but for brevity, we show only every fifth round from 30 to 90 and only the first 1400 addresses. Addresses A and B have similar velocities, so are near each other in the velocity-sorted list, and thus share the window in rounds 35–83. Addresses A and C have less similar velocities, and share a window only in rounds 70–83. Addresses A and D have even less similar velocities, and so D does not enter the window until round 85, after A has exited.

for the targets within the window, since a target cannot be sampled more closely in time than it takes to probe a window sequentially. The position or coverage of the window determines which targets will have overlapping time series. MIDAR ensures that the window covers only likely shared-counter candidates by working with a target list sorted in descending order by target velocity, which puts addresses with similar velocities near each other (MIDAR obtains the target velocities in the Estimation stage, see Sec. IV-A). We can think of the simplest approach described at the beginning of this section as a degenerate case with a fixed window covering the entire target list, so that we collect one sample from all targets in each round and overlapping time series for all targets after multiple rounds of probing.

Fig. 5 illustrates the execution of the sliding window. In the upper subfigure, each dashed horizontal line represents the target list at a particular round of execution (so each vertical line represents the same target address over all rounds), and each solid bar represents the window. For brevity, the figure only shows every fifth round. The lower subfigure shows the target velocities in ID/s, with the target indexes matching up vertically between the two subfigures. For discussion, we have labeled four target addresses (A, B, C, and D) and highlighted their target indexes with vertical lines.

Observe first that the width of the window increases over time, from around 300 targets at round 30 to 1000 targets at round 90. The window must be narrow near the beginning of the list to ensure a sampling interval short enough for the highest velocity in the window. Because velocities vary widely in the beginning (dropping from 1600 to 400 ID/s in the first 300 targets), the narrow window includes all targets that could plausibly share a counter with the highest-velocity target, while excluding many targets that could not. The window is several times wider at round 90 than at round 30 because the target velocities at indexes 400–1400 are much lower with less variation, so the sampling interval can be longer, and more of the adjacent targets are shared-counter candidates based on their velocities.

Next, observe that the window gradually moves down the list over successive rounds. The more rounds two addresses stay together in the window, the more overlap there will be between their time series. We wish to (1) ensure sufficient overlap between targets with nearly the same velocities, (2) obtain at least *some* overlap between targets with moderately similar velocities, and (3) avoid wasting resources on obtaining overlap between targets with sufficiently different velocities. Targets with nearly the same velocities, such as A and B in Fig. 5, are near each other in the target list and thus will stay together in the window the longest and have the most overlap. Targets A and B have overlapping samples from round 35, when B first falls within the window, until round 83, when A last appears in the window. Targets with only similar velocities, such as A and C, are farther apart in the target list and thus will only stay together in the window for a limited number of rounds (rounds 70–83 for A and C), but collecting even a few overlapping samples is still useful for ruling out these unlikely shared-counter pairs with MBT. Finally, targets with sufficiently different velocities, such as A and D, never appear together in a window and have no overlap, but we presume they cannot share counters, so lacking overlap is a feature that improves efficiency.

In probing with the sliding window, MIDAR must balance two competing requirements in each round—the window must be narrow enough to ensure a sufficiently short sampling interval for the highest target velocity in the window, and the window must be wide enough to include all adjacent targets with velocities similar enough to share counters. We can quantify this trade-off with metrics that depend only on target velocities and use the metrics to guide the choice of the optimal window size. Let v_{high} be the highest target velocity in a window, and v_{low} the lowest. These are the velocities of the first and last targets in the window, because the target list is sorted by descending velocity. We define a *spacing* metric for the quality of a window's sampling interval from how much a counter with velocity v_{high} would advance between samples; specifically, we define a counter advancement of 16 384, or 1/4 of the ID space, to be one unit of *spacing*. If the counter advances 1/8 of the ID space, then the spacing will be 0.5. The lower the spacing, the more frequently we sample a target. We define a *similarity* metric for a window's inclusiveness of similar target velocities from the ratio $v_{\text{low}}/v_{\text{high}}$, with a value of 2/3 being one unit of *similarity*. A ratio of 1/3 would be a similarity of 0.5. The lower the similarity, the wider the range of velocities allowed as possible shared counter pairs. As a window becomes larger, the spacing metric increases (gets worse) and the similarity metric decreases (gets better). For each round, we choose the window's starting target index, and then choose the window size at which these two metric values are equal (or cross). If possible, we first advance the starting target index of the window past any targets in the beginning portion of the window that have already been probed at least 30 times while sharing a window with all targets of similar enough velocities to potentially share counters. In this way, the window eventually slides down the entirety of the target list over multiple rounds.

MIDAR partitions the full target list across multiple vantage points and simultaneously probes with a sliding window from all locations. To ensure that all shared-counter candidates (that is, targets with similar velocities) have overlapping time series even across vantage points, MIDAR assigns, to the extent possible, both an equal number of targets and an equal distribution of target velocities to each vantage point, so that the windows of different vantage points cover the same range of velocities at the same time. Targets that can only be probed with the *Indirect* method can only be assigned to vantage points that saw that target in a traceroute path, but despite this hard constraint on target assignment, we are able to achieve nearly identical velocity distributions, due to the flexibility offered by a large number of targets that are usable with other methods. For example, in one MIDAR run, the number of targets assigned to each of 30 vantage points ranged from 43 375 to 43 938, a difference of only 1%.

A pre-calculated schedule drives the execution of the sliding window on each vantage point. We can calculate the position of the window for all rounds ahead of execution and compile this information into a schedule because the sliding window is dependent only on target velocities, which are known prior to execution. The schedule includes a delay in each round for any vantage point that was assigned less than its share of targets for that round, allowing us to finely synchronize the probing of a given velocity range across all vantage points.

The sliding window scales gracefully without manual parameter adjustment to varying numbers of targets and vantage points and to varying levels of overlap quality between time series. Using this approach, with a self-imposed limit of 100 pps per vantage point to minimize impact on the network, we were able to collect the required overlapping time series for 1.9 million addresses in 5.9 hours using 40 vantage points, with a worst case sampling interval of 15% of the wrap period. This aggregate probing rate of 4 000 pps is significantly lower than the 229 748 pps that would be needed by the brute force approach to achieve the maximum allowable worst case interval of 30% of the wrap period.

F. Further reducing false positives

For the millions of addresses typically discovered by Internet-scale mapping experiments, some of the trillions of possible pairs of addresses will have similar IP ID time series over a given measurement period out of sheer coincidence (see Appendix C). Thus, all alias tests will be susceptible to false positives at this scale. Fortunately, unrelated IP ID counters that were coincidentally similar during one period of time will eventually diverge under continued observation since even a tiny difference in velocity becomes magnified over time. We exploit this fact to substantially improve confidence in positive test results and to rule out false positives. We repeatedly test pairs that pass MBT, delaying hours or days between applications. The more MBT applications a pair passes, the higher our confidence of a shared counter, but a single test failure conclusively rules out a shared counter. Because of the virtually zero false negative rate of MBT, assuming the clock error is not underestimated and counter anomalies are detected (see Appendix I), we can repeatedly apply MBT with negligible risk of losing aliases.

We initially identify candidate alias pairs by probing all addresses with a sliding window and then applying MBT (Sec. IV-B). When we take the transitive closure of these candidate alias pairs, we obtain large, sparse *alias sets*, that is, sets of addresses that seem to share a counter. Because of the many inevitable false positives, these alias sets are typically composed of smaller cliques or near-cliques of true alias sets linked together by false alias pairs. We could repeat the sliding window probing to retest candidate alias pairs, but we undertake a more specialized probing approach in order to increase the effectiveness of MBT and thus our confidence in positive test results. We work in two stages. In the first stage, we perform focused probing and testing of every identified candidate alias pair in order to eliminate most false positives and to break up large alias sets into more realistic constituent sets. In the second stage, we probe each of these smaller constituent sets as a whole and apply MBT to both the pairs we have already discovered to share a counter and the pairs implied by transitive closure of those discovered pairs; that is, we check the internal consistency of an alias set, that every pair in a set shares a counter. In both stages, we probe with tighter probe spacing than can be achieved with the sliding window, which makes the MBT bounds tighter and helps to eliminate more false positives. We also repeat the second stage multiple times with delays between executions in order to allow IP ID counters to diverge. Under this regime, each candidate alias pair is tested multiple times over many hours, and the chances of a false positive remaining is extremely low.

IV. MIDAR Implementation

A complete execution of MIDAR is divided into four stages. In the *Estimation* stage, we determine the velocity and best probe method for each address for use in subsequent stages. In the *Discovery* stage, we probe all target addresses with a sliding window schedule that allows us to efficiently discover pairs that potentially share an IP ID counter. In the *Elimination* stage, we re-probe the potential alias pairs to rule out most false positives. Finally, in the *Corroboration* stage, we probe each candidate alias set as a whole to confirm them and to rule out remaining false positives. After completion of all probing stages, we infer reliable alias sets using all available data and results.

A. Estimation stage

In the *Estimation* stage, we ascertain two fundamental properties of each target. We first identify the best usable probing method for each target, according to the method preferences discussed in Sec. III-C. All subsequent MIDAR stages probe each target with only the target's best method. We next estimate the velocity of each target by applying Eq. 2 to the time series collected by a target's best method. The subsequent Discovery stage uses these estimated target velocities to calculate its $I_{\max}$ according to Eq. 1 and create the sliding window schedule.

We partition the target list across vantage points and probe every target with every probing method. Because we care only about the properties of individual targets, we do not need to collect overlapping time series across targets, so the probing procedure is inherently scalable to any number of targets. To avoid potential bias in selecting a probing method, we randomize the probing order of the methods for each target. We probe each target 30 times, with an average spacing of about 7.8 s between probes of a given method to the same target. This spacing is close enough to reliably sample targets with velocities up to 2 520 ID/s, according to Eq. 1.

B. Discovery stage

In the *Discovery* stage, we determine which address pairs appear to share a counter. We first generate a sliding window probing schedule using the velocities found in the Estimation stage and, following this schedule, probe each target with its best probing method. We then analyze the results of these Discovery probes, applying our shared counter tests to every pair of targets with overlapping time series. Our most important test for shared counters is the Monotonic Bounds Test (Sec. III-B). But before applying the MBT, we can sometimes rule out a shared counter with two simpler checks on IP ID byte order and precision (see Appendix I). We explicitly do not use tests based on hop distance between monitor and target, or on the inferred initial TTL set by the target in the response, because these tests are unnecessary and can yield a significant number of false negatives.

C. Elimination stage

In the *Elimination* stage, we perform focused probing and testing of every identified candidate alias pair in order to eliminate most false positives and to break up large candidate alias sets into more realistic constituent sets. Because the primary goal is to eliminate false positives, we wish to probe candidate alias pairs with the tightest probe spacing possible in order to minimize the ID bounds in the MBT. One way to achieve minimal probe spacing is to probe each alias pair separately; that is, obtain overlapping time series for just two addresses at a time. The main drawbacks of this approach are the high cost of probing a large number of alias pairs (6.8 million pairs in our experiment, see Sec. V) and the undesirability of repeatedly probing addresses that are involved in many alias pairs.

We can achieve far greater probing efficiency by exploiting the graph structure of alias sets, with addresses as nodes and candidate alias pairs as edges. Due to chance alignments in the Discovery stage, alias sets tend to be very sparse graphs with many smaller cliques or near-cliques linked together by relatively few edges. In Elimination, we decompose each large alias set into overlapping smaller subgraphs, ensuring each edge occurs in at least one subgraph. We try to extract subgraphs that are as close to a clique as possible, since we can efficiently collect overlapping time series between all pairs in a clique with minimal amount of probes, but if the resulting clique would cause ΔID to exceed 5% of the ID space, we choose a subgraph smaller than a clique to guarantee tight probe spacing for more effective elimination of false positives. To reduce repeated probing of addresses, we try to minimize the number of subgraphs that include any given address. In the experiment described in Sec. V, this subgraph-based probing generated only 15% as many probes as would have been needed by pair-wise probing.

We probe each subgraph for 10 rounds, where a round consists of a single probe to each member of the subgraph consecutively, which guarantees maximum overlap between the time series of the addresses. We send probes to members of the same subgraph no faster than once every 600 ms and no slower than once every second. The purpose of the lower bound is to avoid the appearance of an attack and to avoid rate limiting at the target, since at least one popular brand of router will by default rate-limit ICMP *unreachable* responses to one every 500 ms. For each subgraph $\mathcal{S}$, the round duration is typically a little over $|\mathcal{S}| \times 600\,\mathrm{ms}$ and at most $|\mathcal{S}| \times 1\,\mathrm{s}$.

To reduce total run time, we probe multiple subgraphs in parallel. The artificial delay within each round allows us to interleave rounds of different subgraphs without significantly increasing the duration of each round. We can control our aggregate probing rate by adjusting the number of subgraphs we probe in parallel.

D. Corroboration stage

In the *Corroboration* stage, we take the transitive closure of all candidate alias pairs that passed the Elimination stage to obtain candidate alias sets. We then probe each of the sets as a whole and apply MBT to both the pairs we have already discovered to apparently share a counter and the pairs implied by transitive closure of those discovered pairs.

Probing in the Corroboration stage is the same as in the Elimination stage. The only difference is in the input—the sets are smaller, but we want coverage of every possible transitive closure pair in each set, not just the previously discovered pairs. Although most sets are small, some are still large enough or have high enough velocity that they need to be broken into subgraphs as in Elimination. Compared to Elimination, more subgraphs are required to cover an alias set of a given size in Corroboration because we must probe every pair in the transitive closure. Minimizing the size of these sets by eliminating as many false positives as possible in Elimination allows Corroboration to work with reasonable efficiency.

The Corroboration stage can also be used as a standalone test for potential alias sets discovered by means other than a MIDAR Discovery run, such as with DNS name inference or other alias resolution techniques. Used this way, the Corroboration stage is more efficient and has better PPV and sensitivity than Ally or RadarGun.

E. Final alias inference

After all probing stages, we can finally infer reliable alias sets. First, we find all pairs that passed Elimination and were reconfirmed in Corroboration. Each of these pairs has passed the MBT at least two times, so we have fairly high confidence that they are actually shared counters. The transitive closure of these pairs yields the alias sets corresponding to routers.

TABLE II
CLASSIFICATION OF ADDRESS PAIRS IN OUR EXPERIMENT

stage/classification	pairs	fraction
total input to Discovery	1 753 713 330 078	100.00%
Discovery - unusable	110 318 572 353	6.29%
Discovery - not enough overlap	802 244 369 262	45.75%
Discovery - failed (unshared)	841 143 560 073	47.96%
Discovery - passed (shared)	6 828 390	0.000389%
Elimination - passed (shared)	3 168 049	0.000181%
Corroboration - passed (shared)	2 783 801	0.000159%

For each new pair created through transitive closure, we perform the MBT and other alias tests using any and all data collected in previous stages. Because the Corroboration stage was specifically designed to obtain overlapping time series for every pair in every alias set, we can perform at least one MBT on each of these transitive closure pairs, except when addresses were unresponsive. A *transitive closure conflict* occurs when addresses A and B appear to share a counter, B and C appear to share a counter, but A and C do not share a counter. Such a conflict cannot appear in an actual alias set, but can appear in experimental data due to a false positive or false negative, or an actual change in the underlying topology during data collection. Whatever the cause, we conservatively discard any alias sets with transitive closure conflicts; the sets that remain are MIDAR's final router alias sets.

V. EXPERIMENTAL RESULTS

In this section, we describe an Internet-scale experiment with MIDAR performed on CAIDA's Archipelago (Ark) [17] infrastructure. These results are summarized in Table II.

For input to MIDAR, we collected 2 323 682 addresses, primarily from intermediate (router) addresses in 189 million Paris-traceroute paths in the *IPv4 Routed /24 Topology Dataset* [25], which is an effort to systematically measure IP-level paths from Ark monitors to a dynamically generated list of IP addresses covering all /24 prefixes in routed IPv4 address space. Of these addresses, MIDAR's Estimation stage found that 1 872 813 (80.6%) were usable (Sec. III-A). For more detailed classification of Estimation responses, see Appendix D.

In the Discovery stage, we probed these usable addresses from 40 Ark monitors with a sliding window schedule. Of the $\binom{N}{2} = 1.75 \times 10^{12}$ address pairs, 6.8×10^6 (0.0004%) appeared to use a shared counter. The small fraction is not surprising because the number of shared pairs should be $O(N)$ whereas the number of total pairs is $O(N^2)$. The 45.8% of pairs that did not have enough overlap in their time series to apply the MBT does not mean that MIDAR missed 45.8% of potential aliases. Recall, the sliding window schedule is designed to not waste resources creating overlap between pairs with very different velocities that are thus very unlikely to share a counter. Analyzing all possible pairs in the Discovery stage is by far the most computationally expensive task in a large-scale MIDAR run; using a server with eight 3.0 GHz CPUs (hyperthreading provides a total of 16 logical cores), analysis of the 1.75 trillion pairs took 20 hours. Transitive closure of the apparent shared pairs resulted in 75 350 apparent alias sets containing a total of 1 033 759 addresses.

TABLE III
VALIDATION OF MIDAR ALIAS PAIRS AGAINST GROUND TRUTH. ("UNSHARED" ARE NOT FALSE NEGATIVE ALIASES BECAUSE WE DO NOT AUTOMATICALLY INFER NON-ALIAS FROM UNSHARED, SINCE SOME TRUE ALIASES MAY NOT USE A SHARED COUNTER.)

	validation set	
	R&E	Tier1
true pairs in MIDAR target list	17 930	66 875
true pairs that were monotonic in Estimation	8 061	38 250
MIDAR true positive shared	5 856	26 436
MIDAR false positive shared	0	11
MIDAR unshared	8	386

Of the 75 350 apparent alias sets found by the Discovery stage, 2 706 sets were large enough that we wanted to break them up before the Corroboration stage, which needs small sets to work efficiently. The remaining 72 644 sets were already small enough to be efficiently tested in the Corroboration stage, so we did not test them in the Elimination stage. The largest of the 2 706 large sets contained 618 877 addresses, but only 6 272 188 of the 191 504 061 126 possible pairs were actually classified as shared. This very sparse graph is consistent with our expectation of many smaller cliques or near-cliques of true alias sets being linked together by relatively few false alias pairs. These large sets were successfully broken up by eliminating pairs that failed the MBT or were untestable in the Elimination stage, leaving 174 075 sets containing 704 506 addresses, with the largest set containing 658 addresses.

Of the 3 168 049 pairs in the input to the Corroboration stage, 2 790 570 were pairs that were tested in Elimination and passed, and 2 705 601 (97.0%) of those were reconfirmed as being shared in Corroboration, suggesting that Elimination had already removed the majority of Discovery's false positives among those pairs. The remaining 377 479 pairs belonged to the small Discovery sets that were not subjected to Elimination. Transitive closure of pairs that passed Corroboration yielded 125 497 sets containing 413 828 addresses. The largest set was the same 658-address set found by the Elimination stage. Of these sets, only 13 contained transitive closure conflicts. Treating the conflicted sets as untrustworthy leaves us with 125 484 sets containing 412 900 addresses and 2 490 702 total alias pairs. Only 1631 (0.07%) of the address pairs in 261 (0.2%) of the sets were untested by MBT, that is, inferred only via transitive closure. The high degree of internal consistency in the face of nearly complete full-mesh testing of every set is strong evidence that MIDAR's positive predictive value is extremely high (that is, it finds very few false positives).

VI. VALIDATION

For validation, we used two sets of ground truth data: *R&E*, a collection of known topologies provided by research and educational networks (CAnet [26], CENIC [27], GÉANT [28], I-Light [29], Internet2 [30], and NLR [31]); and *Tier1*, a known topology provided by a Tier 1 ISP.

The most direct validation we can do is test whether MIDAR and a validation set agree on the classification of alias pairs. Table III shows the result of this comparison for the two validation sets. For both sets, the number of false positives is a small fraction of the number of true positives, showing

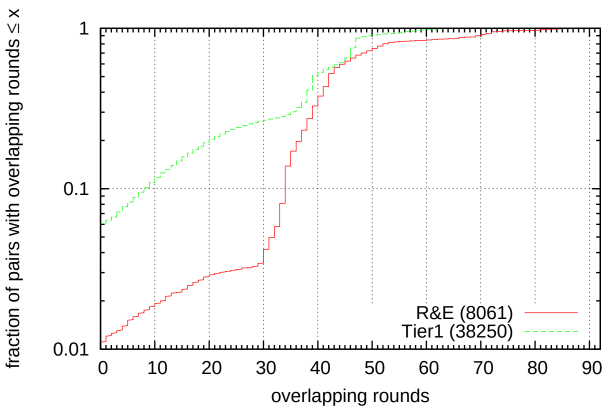

Fig. 6. Time series overlap between known alias pairs during sliding window.

that MIDAR and the validation sets largely agree. Note that disagreements may indicate not just errors in MIDAR, but also errors in the validation set or real changes in the network between collection of MIDAR data and validation sets.

The alias pairs not classified as shared or unshared are those that were inconclusive or discarded. The most significant cause of incompleteness is that interfaces without monotonic IP ID counters simply cannot be tested by MIDAR (nor any other IP ID based technique). Additionally, unresponsiveness can eliminate some interfaces from consideration.

To test the quality of the sliding window schedule, we examine how many rounds of overlap were achieved between the time series of pairs of known aliases. In Discovery, we require at least 5 sample points to pass the MBT; and if there are no discontinuities or unresponsive probes, it takes 3 rounds of overlap to obtain 5 testable points. Fig. 6 shows that the sliding window failed to achieve this 3 round minimum for fewer than 6.7% of known alias pairs in the Tier1 dataset, and fewer than 1.3% in the R&E dataset. Furthermore, these values are only upper bounds on sliding window failures, because some unknown fraction of these non-overlapping pairs are due to the pairs not actually using a shared counter, in which case no amount of overlap would allow them to pass the MBT.

VII. Conclusions

In this work, we described the design, implementation, experimental results, and validation of MIDAR, which extends recent work in IP ID-based IPv4 address alias resolution with new techniques that are higher in precision and sensitivity. MIDAR integrates multiple probing methods, multiple vantage points, an extremely accurate alias test based on IP ID monotonicity, and a novel sliding-window algorithm to increase scalability of IPv4 address alias resolution to millions of addresses. Our experiments show that MIDAR's approach is effective at minimizing false positives sufficiently to achieve a high positive predictive value at Internet scale.

We are currently using MIDAR to capture periodic router-level topology samples of the global IPv4 Internet over time, which we curate and share with researchers [32]. We plan to integrate MIDAR into our larger Multi-Approach Alias Resolution System (MAARS), which combines the strengths of other tools such as kapar and iffinder to improve overall accuracy and completeness.

References

[1] V. Jacobson, "traceroute tool," ftp://ftp.ee.lbl.gov/traceroute.tar.gz.
[2] k. claffy, T. Monk, and D. McRobb, "Internet tomography," in *Nature*, Jan. 1999.
[3] N. Spring, D. Wetherall, and T. Anderson, "Scriptroute: A public Internet measurement facility," in *4th USENIX Symposium on Internet Technologies and Systems*, 2002.
[4] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *ACM SIGCOMM*, 2002.
[5] Y. Shavitt and E. Shir, "DIMES: Let the Internet measure itself," in *ACM Computer Communications Review*, Oct. 2005.
[6] J.-J. Pansiot and D. Grad, "On routes and multicast trees in the Internet," in *ACM SIGCOMM*, 1998.
[7] M. H. Gunes and K. Sarac, "Importance of IP alias resolution in sampling internet topologies," in *IEEE Global Internet 2007 (GI 2007)*, May 2007.
[8] K. Keys, "IP alias resolution techniques," Tech. Rep., 2008, http://www.caida.org/publications/papers/2008/alias_resolution_techreport/.
[9] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *INFOCOM*, Mar. 2000.
[10] K. Keys, "iffinder tool," 2000, http://www.caida.org/tools/measurement/iffinder/.
[11] N. Spring, M. Dontcheva, M. Rodrig, and D. Wetherall, "How to resolve IP aliases," Tech. Rep., May 2004.
[12] M. H. Gunes and K. Sarac, "Analytical IP alias resolution," in *IEEE International Conference on Communications (ICC 2006)*, Jun. 2006.
[13] ——, "Resolving IP aliases in building traceroute-based internet maps," Tech. Rep., Dec. 2006.
[14] R. Sherwood, A. Bender, and N. Spring, "Discarte: A disjunctive Internet cartographer," in *ACM SIGCOMM*, 2008.
[15] A. Bender, R. Sherwood, and N. Spring, "Fixing Ally's growing pains with velocity modelling," in *IMC*, 2008.
[16] J. Sherry, E. Katz-Bassett, M. Pimenova, H. V. Madhyastha, A. Krishnamurthy, and T. Anderson, "Resolving IP aliases with prespecified timestamps," in *IMC*, 2010.
[17] Y. Hyun, "Archipelago measurement infrastructure," http://www.caida.org/projects/ark/.
[18] "Scriptroute source code," http://www.scriptroute.org/source/scriptroute-0.4.8.tar.gz.
[19] R. Elz and R. Bush, "Serial number arithmetic," RFC 1982, Aug. 1996.
[20] "RadarGun source code," http://www.cs.umd.edu/∼bender/radargun/radargun-0.3.tgz.
[21] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, and M. Latapy, "Avoiding traceroute anomalies with Paris traceroute," in *IMC*, Oct. 2006.
[22] J. Aweya, "IP router architectures: an overview," in *International Journal of Communication Systems*, 2001, pp. 447–475.
[23] D. Veitch, J. Ridoux, and S. B. Korada, "Robust Synchronization of Absolute and Difference Clocks over Networks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 2, pp. 417–430, April 2009.
[24] J. Ridoux and D. Veitch, "Principles of Robust Timing Over the Internet," *ACM Queue, Communications of the ACM*, vol. 53, no. 5, pp. 54–61, May 2010.
[25] http://www.caida.org/data/active/ipv4_routed_24_topology_dataset.xml.
[26] https://amidala.canet4.net/cgi-bin/reports.pl.
[27] D. Newcomb, CENIC, La Mirada, CA, private communication, May 2011.
[28] http://stats.geant2.net/lg/.
[29] http://routerproxy.grnoc.iu.edu/ilight/.
[30] http://routerproxy.grnoc.iu.edu/internet2/.
[31] http://routerproxy.grnoc.iu.edu/nlr/.
[32] http://www.caida.org/data/active/internet-topology-data-kit/.

Appendix A
Velocity Distribution

Fig. 7 shows the distribution of velocities for usable time series (Sec. III-A) collected by the Estimation stage (Sec. IV-A). The figure shows separate distributions for each of the four supported MIDAR probing methods (Sec. III-C), with the key

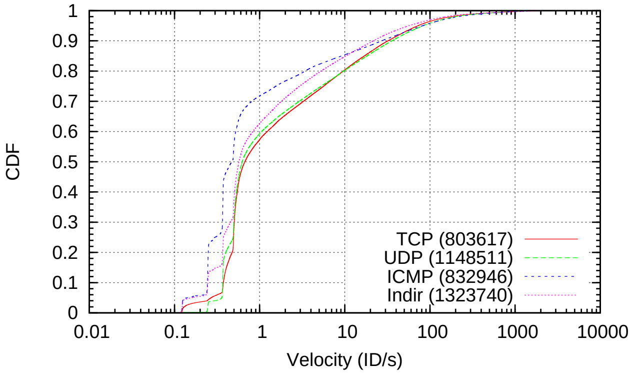

Fig. 7. Distribution of IP ID velocities for usable time series collected by the Estimation stage using four probing methods (the key includes the count of time series for each method). The distribution is heavily skewed toward low velocities and tapers off long before reaching the maximum discernible velocity of 2 520 ID/s for the given sampling interval.

TABLE IV
RELATIONSHIPS BETWEEN BINARY CLASSIFICATION TERMS.

		Actual value		
		Positive	Negative	
Test result	Positive	TP	FP	→ PPV
	Negative	FN	TN	→ NPV
		↓ Sensitivity	↓ Specificity	↓ → Accuracy

giving a count of the included time series for each method. The upper bound on the plot is approximately 2 520 ID/s, the maximum we could detect with our chosen sampling interval. The CDF tapers off long before reaching this upper bound, suggesting there are not many actual interfaces using monotonic IP ID counters with velocities higher than this bound; that is, any apparent velocities higher than this bound are likely due to randomly generated IP ID values.

APPENDIX B
BINARY CLASSIFICATION

To aid in discussion of alias resolution tests, it is useful to review some terminology commonly used in epidemiology and other fields. Some of these terms and their relationships are illustrated in table IV.

- *positive*: having the condition in question (e.g., a pair of addresses sharing an IP ID counter, or being aliases)
- *negative*: not having the condition in question
- *actual positives* (AP): number of cases that actually have the condition in question
- *actual negatives* (AN): number of cases that actually do not have the condition in question
- *prevalence*: fraction of cases that actually have the condition, $\mathrm{AP}/(\mathrm{AP}+\mathrm{AN})$
- *true positives* (TP): actual positives that test as positive
- *true negatives* (TN): actual negatives that test as negative
- *false positives* (FP): actual negatives that test as positive
- *false negatives* (FN): actual positives that test as negative
- *sensitivity* or *true positive rate* (TPR) or *recall*: fraction of actual positives that test as positive, TP/AP
- *specificity* or *true negative rate* (TNR): fraction of actual negatives that test as negative, TN/AN
- *positive predictive value* (PPV) or *precision*: fraction of positive tests that are correct, $\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})$
- *negative predictive value* (NPV): fraction of negative tests that are correct, $\mathrm{TN}/(\mathrm{TN}+\mathrm{FN})$ (the value $1-\mathrm{NPV}$ was called "false negative (rate)" in [11])
- *accuracy*: fraction of tests that are correct, $(\mathrm{TP}+\mathrm{TN})/(\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN})$
- *false positive rate* (FPR or α): fraction of actual negatives that test positive, FP/AN
- *false discovery rate* (FDR): fraction of positive tests that are incorrect, $\mathrm{FP}/(\mathrm{TP}+\mathrm{FP}) = 1-\mathrm{PPV}$ (called "false positive (rate)" in [11])

If there are no inconclusive test results or we treat inconclusive results as negative, then the following identities hold:

- $\mathrm{AP} = \mathrm{TP}+\mathrm{FN}$
- $\mathrm{AN} = \mathrm{TN}+\mathrm{FP}$
- $\alpha = 1 - \text{specificity}$

Accuracy alone is not a good measure of the quality of a test. When prevalence is low, as in the case of large scale alias resolution, a test that mostly gives negative results will have a large number of true negatives and thus high accuracy, but might still have poor sensitivity and PPV.

APPENDIX C
FALSE POSITIVES

The potential for false positives is very high when using IP ID time series for alias resolution at Internet scale.

According to the well-known "birthday problem," in a group of just 23 or more randomly chosen people, there is greater than 50% probability that at least one pair of people will have the same birthday. Similarly, given that the IP ID space has $2^{16} = 65\,536$ distinct values, it takes a group of just 302 IP addresses to have a 50% probability of some pair of addresses having the same IP ID value at any given time, and just 777 addresses for a 99% probability. When the number of targets N passes the number of possible values H, collisions are guaranteed by the pigeonhole principle. Even worse, if our IP ID test allows a range of nearby values instead of just equal values, the frequency of collisions increases by an amount that can be approximated by dividing H by a factor proportional to the size of the range. This would be the case if it were possible to probe all N targets instantaneously with Ally. In the context of the birthday problem, this requirement would be like requiring a pair of people in a group to have birthdays within 4 days of each other (which happens with 50% probability in a group of just 9 people).

Given N target addresses, and an average of d addresses per router, the number of alias pairs (actual positives) is approximately the number of interface pairs per router times the number of routers:

$$\mathrm{AP} \approx \binom{d}{2} \times \frac{N}{d} = \frac{N(d-1)}{2} \tag{3}$$

and the number of non-alias pairs (actual negatives) is the total number of pairs that must be compared minus the actual positives:

$$\mathrm{AN} = \binom{N}{2} - \mathrm{AP} \approx \frac{N(N-d)}{2} \tag{4}$$

TABLE V

CLASSIFICATION OF IP ID BEHAVIOR OF ADDRESSES PROBED WITH VARIOUS METHODS. PERCENTAGES ARE RELATIVE TO ADDRESSES PROBED.

	TCP		UDP		ICMP		indirect	
addresses probed	2 323 641	100.00%	2 323 641	100.00%	2 323 641	100.00%	1 832 771	100.00%
insufficient responses	905 267	38.96%	1 151 476	49.55%	482 399	20.76%	352 537	19.24%
— mostly unresponsive	865 498	37.25%	1 014 227	43.65%	459 545	19.78%	322 991	17.62%
— mostly unexpected	39 741	1.71%	136 863	5.89%	22 723	0.98%	28 454	1.55%
non-counter ID values	137 363	5.91%	17 164	0.74%	999 677	43.02%	138 553	7.56%
— mostly zero	130 744	5.63%	15 044	0.65%	1 293	0.06%	110 849	6.05%
— mostly repeat	100	0.00%	568	0.02%	236	0.01%	985	0.05%
— mostly reflect	6 516	0.28%	1 349	0.06%	998 077	42.95%	26 270	1.43%
not monotonic	477 394	20.55%	6 490	0.28%	8 619	0.37%	17 941	0.98%
monotonic	803 617	34.58%	1 148 511	49.43%	832 946	35.85%	1 323 740	72.23%

The prevalence is then $(d-1)/(N-1)$. Some fraction α of the tests on actual negatives will give false positive results when counters belonging to unrelated addresses are coincidentally synchronized to within the tolerance of the test. Then, the total number of false positives will be

$$\text{FP} = \alpha \times \text{AN} \approx \alpha \times \frac{N(N-d)}{2} \tag{5}$$

For alias resolution results to have a useful *positive predictive value*, the number of false positives must be much smaller than the number of actual positives. Comparing Eq. 5 and Eq. 3, and solving for α, gives us an upper bound on useful values of α:

$$\alpha \ll \frac{d-1}{N-d} \tag{6}$$

Thus, when $N \gg d$, the maximum acceptable false positive rate of the test is inversely proportional to the number of target addresses.

To decrease α, RadarGun and MIDAR compare tens of sample points in time series, as opposed to just two points in Ally. However, the decrease is not as much as one might expect, for two reasons. First, the samples in a single series are not independent, but are related by an underlying counter that increments with a somewhat regular rate. From this perspective, we can view the test as requiring that two counters have similar *initial* ID values and similar *velocity* (rate of ID change). Second, because the velocity distribution of real ID time series is heavily skewed towards low velocities as seen in Fig. 7, many pairs of counters will have a low velocity difference. Two unrelated counters that start with a similar ID value and have a low velocity difference will take a long time to diverge.

Furthermore, note that the alias relationship is transitive. That is, if addresses A and B are aliases, and B and C are aliases, we must infer that A and C are also aliases; all three addresses belong to the same router. Even a small set of false positives, interpreted at face value, could lead us to incorrectly merge many distinct routers into one. The topology distortion caused by false positives is thus amplified by transitive closure.

APPENDIX D
RESPONSE RATE AND IP ID CHARACTERISTICS

To study the usefulness of the probing methods, we analyze our Estimation run, in which we probed 2 323 641 addresses with all available probing methods. The Indirect method could not be used with non-Ark addresses, because we do not have the necessary traceroute information for them. Table V shows the results.

We count a target as having *insufficient responses* if fewer than 75% of the probes to the target elicit the expected response. The subcategories enumerate the most common reasons. *Mostly unresponsive* means more than 75% of probes did not elicit *any* response. During Indirect probing, a sequence of TTL expansion that does not elicit any response from the target is counted as a single non-response. We count a target as *mostly unexpected* if more than 75% of the probes elicit a response of an unexpected type. For most targets, either all or none of the responses are unexpected. Most of the unexpected responses are ICMP *destination unreachable* messages from non-target addresses.

The main cause of unresponsiveness for the Indirect method appears to be network changes during the delay between the traceroutes and our experimental probes. When the delay is shorter, the response rate is higher. For example, in a different Indirect probing run to 3 000 targets from a single monitor, the response rate was 98.8% for addresses gathered from traceroutes taken only 3–4 hours earlier. The traceroutes collected for Table V were taken up to 18 days before the Estimation run, showing that Indirect probes can still be useful even after a moderate delay. However, we do see significant variability between monitors in the response rate to Indirect probing, suggesting different levels of route instability and per-packet load balancing near each location.

We classify a target as having *non-counter ID values* if it had sufficient responses but 25% or more of the ID values were zero, some other constant value, or the value used in the probe packet. Such ID values are not useful to us because they do not reveal the state of an underlying shared counter. The subcategories enumerate targets for which more than 75% of IDs had the same type of non-counter value. Nearly half of the targets respond to ICMP *echo request* by echoing the ID, and a significant fraction of targets respond to TCP and Indirect with zero-valued IDs.

Finally, any target that passed all of the above tests is classified as *monotonic* if its response IDs can be modeled as a monotonic counter, otherwise *nonmonotonic*. TCP is the only method for which a significant fraction of targets passed the earlier criteria only to be classified as nonmonotonic.

TABLE VI
UTILITY OF COMBINING MULTIPLE PROBING METHODS. PERCENTAGES ARE RELATIVE TO 2 323 641 TOTAL ADDRESSES.

combination of methods				responsive	usable
tcp				62.75%	34.58%
	udp			56.35%	49.43%
		icmp		80.22%	35.85%
			indir	64.97%	56.97%
	udp	icmp	indir	88.27%	77.39%
tcp		icmp	indir	89.11%	76.02%
tcp	udp		indir	85.72%	77.28%
tcp	udp	icmp		82.66%	68.91%
tcp	udp	icmp	indir	89.25%	80.60%

TABLE VII
CROSS-METHOD COUNTER SHARING FOR ADDRESSES THAT YIELD USABLE TIME SERIES TO MULTIPLE METHODS.

methods	addresses	shared	
TCP:UDP	595 465	562 582	94.48%
TCP:ICMP	383 712	341 247	88.93%
TCP:indir	511 111	456 523	89.32%
UDP:ICMP	523 710	509 951	97.37%
UDP:indir	774 993	745 913	96.25%
ICMP:indir	545 585	525 224	96.27%

APPENDIX E
UTILITY OF MULTIPLE PROBING METHODS

Table VI shows the increase in target responsiveness and usable time series achievable by employing multiple probing methods for our dataset. Individually, ICMP has the highest responsiveness but the lowest amount of usable time series due to many addresses echoing the IP ID of the probe in the response. UDP and Indir have the highest amount of usable time series of any single method despite being more susceptible to rate limiting. If we employ all four methods, 89.2% of addresses respond to at least one method, and 80.6% yield usable time series to at least one method. This improved coverage will make alias resolution much more complete.

APPENDIX F
CROSS-METHOD IP ID COUNTER SHARING

Table VII shows the frequency of cross-method counter sharing for our dataset. For each pair of methods, Table VII lists the number of addresses that responded to both methods with usable IP ID values and then the count and percentage of those addresses that had a shared counter. Overall, there is a high incidence of counter sharing, ranging from 88.9% to 97.4%. As expected, TCP and UDP share often at 94.5%. The sharing rates of the remaining pairs seem to be correlated with the response type; that is, counters seem more likely to be shared when two probing methods elicit a similar type of response. For instance, the sharing rate of TCP with either ICMP or *Indirect* is comparatively low perhaps because TCP rarely elicits an ICMP response. In contrast, UDP always elicits an ICMP response, and we thus see comparatively greater counter sharing between UDP, ICMP, and *Indirect*.

APPENDIX G
NEGATIVE DELTA RATE OF RANDOM TIME SERIES

A *random time series* is produced from random IP ID values rather than from a monotonic counter. In random time series, the average probability of an *individual* delta being negative is 50%, regardless of the sampling rate. Therefore, the expected number of negative deltas appearing in a random time series of n values is given by the binomial distribution for $n-1$ trials and $p = 0.5$. This distribution is a bell-shaped curve with mode at $(n-1)/2$.

For a time series of 30 samples (29 deltas), we would allow a maximum of $\lfloor 0.3 \times 29 \rfloor = 8$ negative deltas before classifying the time series as unusable based on our 30% threshold on negative deltas (Sec. III-A). The probability of getting 8 or fewer negative deltas out of 29 random deltas is just 0.012, so 98.8% of random time series will be correctly identified as unusable. We do not need to detect all random time series because of the extra testing MIDAR performs to eliminate false positives (Sec. III-F), but we can reduce work by detecting and eliminating targets that produce random time series.

APPENDIX H
DISCONTINUITIES IN TIME SERIES

An IP ID time series that appears mostly monotonic may have an occasional *discontinuity*, a local region of uncertainty where we cannot be confident that a counter remained monotonic between individual samples.

There are two types of discontinuity. First, there is a discontinuity if the time gap between samples is too large, or more precisely, if Δt_i is greater than 3.5 times the median Δt of the same time series. This means that we lost three or more consecutive samples (assuming a regular spacing of probes) due to rate limited responses or packet loss. Recall that our definition of a usable time series required at least three samples between counter wraps, so if we have lost three or more samples, the gap may hide one or more counter wraps.

The second type of discontinuity occurs when the counter advances too quickly between samples, which could be due to a burst of router traffic causing high velocity monotonic ID advancement, but could also be due to the router's counter being reset, causing a non-monotonic ID change. Let $\hat{v}$ be the median segment velocity for a given time series. If either the actual counter advancement ΔID_i or the expected counter advancement $\hat{v}\Delta t_i$ of a segment is greater than 30% of the ID space, we mark that segment as a discontinuity.

We take discontinuities into account in all our analyses, allowing us to use time series that would otherwise introduce errors or be unusable. For example, we exclude discontinuities when computing $\bar{v}$ in Eq. 2; that is, for a discontinuity between samples i and $i{+}1$, we exclude ΔID_i and Δt_i, thus improving the robustness of $\bar{v}$ to atypical or transient counter behavior. We observed a discontinuity in approximately 0.8% of the usable time series we collected.

APPENDIX I
ANOMALIES IN MONOTONIC COUNTERS

We observe several types of anomalies in IP ID values. MIDAR detects and accounts for these anomalies in order to maximize its sensitivity and positive predictive value,

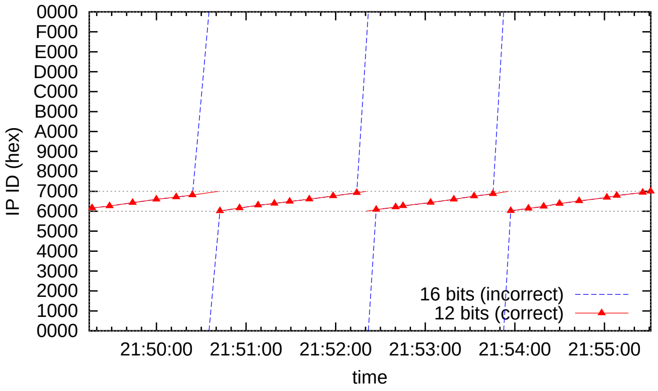

Fig. 8. An example of the limited-precision counter anomaly. The naïve interpretation shows a much larger unwrapped delta (dashed line) than the correct interpretation.

Most routers transmit ID values in big-endian order (network byte order), but some use little-endian order. If ID values from a low-velocity counter are interpreted in the wrong byte order, then the counter will appear to have a velocity about 256 times greater than its true velocity. On the other hand, if a high-velocity counter is interpreted with the wrong byte order, it will be indistinguishable from random. We developed an inexpensive test to detect the correct byte order, and found that approximately 0.6% of usable time series were little endian. We can also use byte order as an additional criterion for ruling out aliases, assuming that every interface of a router would use the same byte order for a given probe method (Sec. III-C) (but we do not assume that every router uses the same byte order for different probe methods).

The second type of anomaly is caused by routers that do not use all 16 bits of the IP ID field. Such a *limited precision* counter will wrap around its smaller ID space more frequently than a full precision counter with the same velocity, as shown in Fig. 8. To identify a b-bit limited precision counter, we require not only that the $16 - b$ high bits are constant, but also that there is at least one wrap, and that every wrapped segment, after being unwrapped, has a velocity similar to that of the non-wrapped segments. Failing to identify limited precision counters would not directly lead to false results, but it would lead us to unnecessarily mark their their wrapped segments as discontinuities. We can also use limited precision counters to rule out shared counters: two time series with different ranges cannot share a counter. We found about 0.39% of usable time series had limited precision, most commonly using 12 bits with values between 0x6000 and 0x6FFF.

The final type of anomaly we observed appears to be due to a race condition in which the two bytes of the ID counter are incremented asynchronously. For example, we may see a sequence of (hexadecimal) ID values like 11F7, 11FB, 1100, 1204, 1209, where it appears the third value *should* be 1200, but the high byte has not yet been incremented. We also saw cases in the opposite order, e.g. 11F7, 11FB, 12FF, 1204, 1209, where it appears that the low byte of the third value has not yet changed from FF to 00, even though the high byte has changed from 11 to 12. We call these anomalies *XX00 and XXFF outliers*. We take a conservative approach and discard these questionable samples from the time series. Although we found these anomalies in only 0.01% of 3.2 million observed monotonic time series, failing to account for them could result in false negatives in the MBT, decreasing the overall sensitivity.