FlashRoute: Efficient Traceroute on a Massive Scale

Yuchen Huang
Dept. of Computer and Data Sciences
Case Western Reserve University
yuchen.huang2@case.edu

Michael Rabinovich
Dept. of Computer and Data Sciences
Case Western Reserve University
mchael.rabinovich@case.edu

Rami Al-Dalky
Dept. of Computer and Data Sciences
Case Western Reserve University
rami.al-dalky@case.edu

ABSTRACT

We propose a new traceroute tool, FlashRoute for efficient large-scale topology discovery. FlashRoute reduces the time required for tracerouting the entire /24 IPv4 address space by a factor of three and half compared to previous state of the art. Additionally, we present a new technique to measure hop-distance to a destination using a single probe and uncover a bias of the influential ISI Census hitlist [18] in topology discovery.

CCS CONCEPTS

• **Networks → Network measurement**.

KEYWORDS

Internet topology measurement, traceroute

ACM Reference Format:
Yuchen Huang, Michael Rabinovich, and Rami Al-Dalky. 2020. FlashRoute: Efficient Traceroute on a Massive Scale. In *Proceedings of IMC '20*. ACM, New York, NY, USA, 13 pages. https://doi.org/TBA

1 INTRODUCTION

Traceroute is one of the fundamental tools for Internet measurements, widely used to troubleshoot network connectivity issues and reveal the network topology. Among its common applications, traceroute is used to explore the routing topology to a large number of destinations or even the entire Internet [6, 12, 21, 22]. Doing so requires traceroute measurements on a massive scale, which means completing these measurements with as few probes and in as little time as possible. Shortening the time for topology measurements is especially critical because the shorter the time to complete the measurement the closer to a snapshot the results will be and the easier it is to understand the dynamics of Internet routing changes at fine time granularity.

The current state of the art in time efficiency of Internet-scale traceroutes is Yarrp [4, 5]. Its probing strategy is inspired by ZMap [9] and adopts ZMap's two main techniques to achieve high performance. First, Yarrp encodes all probing context (i.e., information needed for matching probes with responses and to time the response) in the probe's header, which makes the scanning process stateless, thus reducing memory footprint of the process, and decouples packet sending and receiving tasks, hence dramatically increasing the probing parallelism. Second, to avoid hitting routers' ICMP rate limits, Yarrp introduces a computationally efficient random permutation technique to generate all combinations of destination /24 prefixes and TTLs on the fly, without preloading all destinations. With Yarrp, tracerouting all /24 prefixes in IPv4 address space can be finished from a single vantage point in barely an hour at the probing speed of 100Kpps.

The drastic improvement in the tracerouting speed, however, comes at the price of increased number of probes, because for every target prefix, the vantage point blindly issues a probe to every hop within the maximum TTL, regardless of the actual hop-distance to the destination.

In this paper, we propose a new tool for massive tracerouting, FlashRoute, which further reduces the time required for tracerouting the entire /24 IPv4 address space by the factor of three and half, from one hour to 17 mins, at the probing speed of 100Kpps. And if given the same time budget as Yarrp, FlashRoute can run extra scans, using different source ports, to discover router interfaces on the alternative routes formed by load-balancing hops.

FlashRoute achieves this efficiency by reintroducing state into the probing process. The state is carefully designed to preserve high probing parallelism yet enable elaborate probing strategies that significantly reduce both the number of probes and the time required to complete the scan. At the same time, the state is compact enough, around 900MB for a complete scan of the /24 IPv4 address space, to be easily handled by modern computers.

FlashRoute's efficiency can be attributed to several factors. First, before the main scan, FlashRoute attempts to calculate the hop-distance to each destination using a single probe, and uses these distances to optimize the rest of probing. Second, similar to Yarrp, FlashRoute avoids online matching of probes with responses by encoding all information necessary to derive the measurement result into probe headers, which allows FlashRoute to probe in a highly parallel way. Third, FlashRoute utilizes a specially designed data structure to maintain some control states without hampering probing parallelism, which allows it to adjust the exploration process of individual routes based on the feedback from responses. In particular, this allows FlashRoute to explore routes in a backward direction and stop probing when detecting the route convergence, the idea originated in Doubletree [8] but abandoned by Yarrp in favor of randomized parallel probing at every TTL.

An interesting side-observation of our experiments, and another contribution of this paper, is that we find ISI Census Hitlist [18], an influential list of most responsive IP addresses from every /24 prefix, to be biased for the purpose of Internet topology exploration. Specifically, the IP addresses from the Hitlist tend to be at shorter hop-distances from our vantage point than random destinations.

IMC '20, October 27 – 29, 2020, Pittsburgh, PA, USA

ACM ISBN TBA…$TBA
https://doi.org/TBA

We present preliminary evidence that the Hitlist destinations are preferentially located at the entrance of the stub networks, shielding from discovery any other routers in the same prefixes that might be behind the interfaces provided by the Hitlist.

To recap, the main contributions of this work are:

- We present a new tool for massive-scale traceroutes, which uses less than 30% of probes, and correspondingly, runs the full /24 IPv4 scan in less than 30% of time, of the previous state of the art;
- We describe a new method of measuring hop-distance to destinations using a single probe;
- We uncover a non-obvious bias of the ISI Census hitlist [18] for the purpose of Internet topology exploration.

2 BACKGROUND AND RELATED WORK

Traceroute has been widely studied over the years, with a number of solutions proposed to improve its the performance. In particular, Doubletree [8] and its variants [7, 14] reduce probing redundancy based on the observation that the routes originated from a vantage point to the Internet form a tree-like structure, so new routes only need to be explored between the destinations and the branching points from the previously discovered routes. Augustin et al. proposed Paris traceroute to reduce the interference caused by load-balancers in route exploration and also invented the multipath detection algorithm (MDA) to explore alternative routes created by the load-balancers [2, 3]. Later, Vermeulen et al. introduced Multilevel MDA-Lite [23], reducing the probe usage of MDA. CAIDA has a long-running experiment to continuously measure the Internet topology from distributed vantage points using Scamper [15], which employs both Doubletree and Paris traceroute techniques [6]. While these solutions improve traceroute in various ways, the underlying probing technique remains the same as in the conventional traceroute: probes are issued sequentially one by one to every hop for each route, issued probes are remembered and matched to the responses, and expiration of pending probes is managed separately for each probe to handle missing responses.

Yarrp [4] introduced a new probing technique inspired by ZMap [9], which greatly increases the probing parallelism and thereby the probing rate. Later, Yarrp6 [5] extended Yarrp by adding the "fill mode". The fill mode reduces the maximum TTL within which hops are probed exhaustively by Yarrp and complements exhaustive probing within the maximum TTL with sequential probing of one extra hop upon arrival of the response from the farthest previously probed hop if this response indicates that the farthest hop is not the end-target. The fill-mode can reduce the number of probes sent in a scan, albeit at the cost of missing some interfaces because a single non-responsive router at a given hop-distance terminates the sequential forward probing. Both Yarrp and Yarrp6 use a stateless probing design, allowing the probing to occur with negligible amount of state. This design, however, results in an increase in the needed volume of probes. Indeed, as the state is removed, the scanner is unable to effectively utilize the feedback from the responses to adjust probing strategy. Therefore, both Yarrp and Yarrp-6 explore topology largely in an exhaustive manner by sending probes to every possible hops for every destination.

FlashRoute breaks the tradeoff between two types of optimizations of traceroute: the optimizations having low probing parallelism but low measurement traffic, and the optimizations having a high parallelism of probing but high measurement traffic. Consequently, FlashRoute can further significantly reduce the time for a full IPv4 address space scan.

As the probing speed increases, the ICMP rate-limiting becomes another important issue. [19] found pervasive ICMP rate-limiting in IPv4, with most routers allowing 500 or less ICMP replies per second. This observation suggests the network measurement tool not only needs to have a high speed but also low probing traffic that avoids creating hot spots. Otherwise, ICMP rate-limiting may be triggered, which would negatively affect the accuracy or coverage of the topology discovery. We discuss FlashRoute behavior with regard to ICMP rate limiting in §4.2.

3 FLASHROUTE OVERVIEW

FlashRoute is optimized for tracerouting a large set of targets. It combines several techniques from existing traceroute solutions with novel ideas for increasing the aggregate efficiency of the overall scan. FlashRoute adopts the following existing techniques:

- Following the lead from Paris traceroute [2], FlashRoute uses the same source/destination ports and protocol ID in all probes that explore the route to a given target. This reduces the risk of exploring hops scattered across alternate routes caused by load-balancing routers.
- FlashRoute borrows the idea of packet encoding from Yarrp [4, 5] and, ultimately, ZMap [9], although our encoding details are different (see §3.1).
- FlashRoute utilizes the Doubletree [8] basic technique, also used by Scamper, to optimize the route exploration. Doubletree explores routes from an intermediate point, moving backward, towards the vantage point, and forward, towards the target. The backward probing stops once encountering a previously discovered interface, reducing redundant probing of previously discovered route sections. We refer to the intermediate point from which backward and forward probing commences as "split point" and the TTL that reaches this point as "split TTL".

FlashRoute's main innovation is the mechanism that retains Yarrp's advance of high parallelism through decoupling probe-sending and response-receiving tasks but at the same time preserves Doubletree's ability to reduce probing traffic by adjusting probing strategy based on feedback from prior responses. We thus prove, by construction, that high parallelism in massive traceroute scans does not have to come at the cost of high measurement traffic overhead. While it does introduce some state, resulting memory footprint is easily manageable by modern computers.

FlashRoute is written in C++ using 2.4k lines of code. It is compatible with a variety of Unix-based operating systems. We make it, along with the data collected in our study, publicly available at [10, 11].

3.1 Probe Encoding

Like Yarrp and Zmap, we encode all needed information to interpret the measurement into the probe packet header, which is then

returned in the ICMP response payload. In particular, we encode initial TTL of the probe into 5 bits of the IPID field and use the checksum of the destination IP address as the source port number to verify that the destination has not been changed by a middlebox en-route. We use another bit of IPID to distinguish probes sent during the preprobing phase (§3.3). Finally, we encode the timestamp of the probe but use a different approach from Yarrp to do it. Because Yarrp uses TCP-ACK probes by default, it encodes elapsed time from the start of the scan into the TCP sequence number field. We wanted to support UDP probes because, as previous studies shown [17] and we confirmed (§4.2), UDP probes produce more responses. For the UDP probes, Yarrp attempts to encode the entire elapsed time into the checksum and the packet length fields, which we found to be problematic (§4.2.1). Instead, we only use 10 remaining bits of IPID and 6 bits of the packet length fields to encode timestamp of the probe. With millisecond granularity, this leads to wrap-around in just over 65 seconds – less than the official maximum segment lifetime but more than enough to derive the round-trip time for all practical purposes.

3.2 Probing Strategy

FlashRoute explores routes to all destinations in a series of rounds. Each round cycles through all destinations and issues all its probes for all destinations back-to-back, without waiting for responses. Thus, the amount of parallelism in exploring the destinations participating in a round (initially all destinations) is only limited by how quickly FlashRoute is able (or allowed) to issue the probes. Sending probes and processing responses is decoupled (save for an unlikely mutex contention to access the control state for a destination, see §3.4) and is done through separate threads. To avoid creation of hot spots when probing topologically close destinations, FlashRoute shuffles the destinations into a random permutation before the rounds begin.

In each round, up to two probes per destination are issued: one to the next hop backward, towards the vantage point, and one to the next hop forward, towards the target. Forward probing towards the target will stop once the probes reach the target (more precisely, once the probes elicit a "host/port/protocol unreachable" response for the target address) or if the number of consecutive non-responsive hops reaches a configured *GapLimit* parameter. Backward probing for a target ends once a probe reaches the vantage point (or rather the hop at TTL 1 from the vantage point) or encounters a previously discovered hop. With both backward and forward probing completed, the route exploration for the destination completes and the destination is removed from further rounds.

The sending thread ensures that each round lasts at least one second, stalling if necessary before starting the next round. This allows at least one-second elapsed time between consecutive route exploration steps for a given destination, increasing the likelihood that enough responses will have arrived to adjust the probing strategy, i.e., to stop probing in either direction. One consequence of this strategy is that as the number of incomplete destinations decreases, the probing parallelism will at some point decrease. However, for 100 Kpps, this limit only comes into play at the very end of the process, when the number of unexplored destination drops to between 50K and 100K (given that an unexplored destination produces one or two probes in a round). Therefore, it does not significantly affect the overall time of the scan.

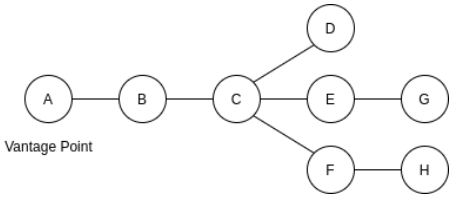

Figure 1: Tree-like router topology: routes merge into common sections close to the source.

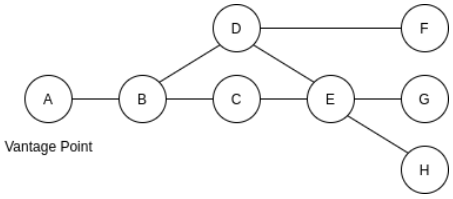

Figure 2: Alternative routes created by load balancer B are missed by backward probing.

3.2.1 Backward probing issues. The reasoning behind backward probing was originally explained and verified in the Doubletree study and has since become widely accepted; that is, routes from one vantage point to the rest of the Internet approximate a tree-like topology. Figure 1 provides an example of this tree-like topology, where the routes to targets D, G, and H reuse the same section A-B-C up to the convergence point at router C. Backward probing allows probing of the common section to occur only once.

At the same time, backward probing can miss some routes, when the network deviates from the tree topology, as shown in Figure 2. Here, router B balances traffic by forwarding packets to target H to router C, producing route B-C-E-H and forwarding packets to target G to router D, resulting in route B-D-E-G. By stopping at the convergence point E, backward probing will miss one of these routes. Yet another potential issue is that if the starting point is greater than the route length and the end-target is not responsive to probes, the backward probing will waste extra probes by exploring nonexistent hops one by one in the backward direction. Indeed, in this case all those extra non-existent hops will appear as non-responsive routers, and the backward-probing process has no alternative but to continue probing each hop until reaching the responsive section.

The first issue is alleviated to some extent by the fact that FlashRoute is designed to perform massive scans, and some route sections missed by one target can be discovered by another target. For instance, in Figure 2, even if router D is missed by backward probing from G and H, it would still be discovered by backward probing from F. The second issue compels systems utilizing backward probing to use the split point that is relatively close to the source and augment it with forward probing.

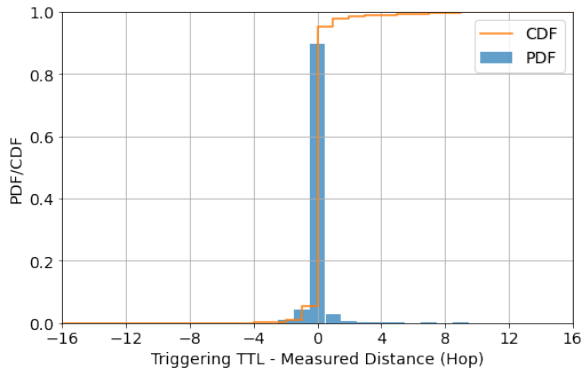

Figure 3: PDF and CDF of the difference between the triggering TTLs and the measured distances of targets.

In summary, setting the split point far from the vantage point explores most of the routes in a backward manner – this allows more aggressive elimination of redundant probes by detecting distant route convergence points but risks probing through long stretches of unresponsive routers at the tails of the routes. Starting too close has the opposite effect: it explores most of the route with forward probing, more likely stopping fruitless exploration of unresponsive route tails, but – by narrowing the scope of backward probing – could miss probes-saving opportunities that backward probing exploits. In this paper, we consider the performance of FlashRoute with the split point at TTL 32 (the maximum TTL explored by Yarrp, which very few paths exceed) and 16 (given the the Doubletree study finding that reuse of interfaces by routes to multiple destinations plunges at hop distances approaching this value).[1] In addition, FlashRoute offers an option to measure hop-distance to targets in a special *preprobing* phase and use this distance as the split point (§3.3). Starting backward probing at this point will prevent wasted probes if the distance is accurate.

3.3 Preprobing

FlashRoute performs preprobing using a new highly efficient mechanism for precise measurement of hop-distance to the destination with a single probe. We present this mechanism, assess its accuracy, and discuss its use in FlashRoute, below.

3.3.1 One-probe hop-distance measurement. The method to measure hop-distance with one probe is as follows. FlashRoute issues a probe with TTL 32 to every destination at port 33434. Since the destination port is set to 33434, which is a port reserved for traceroute purpose, the probe may trigger the host to return an ICMP port-unreachable response. As the response carries the original packet header of the probe, FlashRoute can calculate the hop distance of the destination using the residual TTL of the probe extracted from the response. The calculated distance reflects the length of the forward path from the vantage point to the destination.

3.3.2 Accuracy of hop-distance measurement. The accuracy of this method can be affected if a middlebox resets TTL for a probe enroute to the destination (e.g., as suggested in [13]). In order to verify how often this case happens, we send probes with TTLs covering the entire 1 to 32 range and record the first TTL that triggers the ICMP port-unreachable, mimicking the traditional approach to distance measurement through traceroute. We thereby can compare the difference between the minimum TTL that triggers the ICMP port-unreachable responses, and our measured hop-distance to the destination. Note that this method will detect the inaccuracy but not necessarily the extent of it, because the traceroute method itself is not guaranteed to be accurate. In particular, if a middlebox resets TTL to a standard value, a probe will reach the destination as long as its initial TTL is sufficient to reach the middlebox regardless of the number of remaining hops. Typically, however, middleboxes sit at the entrance of the stub networks, and the number of the remaining hops is small, so we can assume the distance to the middlebox roughly approximates the distance to the destination.

As shown in Figure 3, for around 89.7% of routes, the measured distances and triggering TTLs are the same, indicating the measurements are accurate. For additional 7% of destinations, triggering TTLs are within 1 hop of the measured distances. Because it would take quite a coincidence for the middlebox to replace the current TTL of an incoming packet with such a close value, we speculate that many of these discrepancies are caused by route dynamicity; in any case, the discrepancy is small. This leaves only around 3.3% of destinations where our method differs from the traditional traceroute by more than 1 hop. We conclude that our proposed measurement of hop-distance provides generally accurate results with a single probe to the destination.

3.3.3 Hop-distance predictions. Our distance measurement technique requires the destinations to return ICMP port unreachable responses. Applied to FlashRoute, however, it produces few results because FlashRoute picks a random target address within each /24 block, which may not even be assigned to a host. To improve the recall of distance measurements, we observe that many stub networks advertise larger than /24 blocks (e.g., our campus advertises two /16 networks), in which case many adjacent /24 blocks will have similar hop-distance from the vantage point. Coupled with the fact that FlashRoute uses these distances as purely performance-improving hints, it can take the measured distance to one block to predict the distances to other blocks in close proximity in the address space. For example, if the address block 100.100.123/24 is 15 hops away from the vantage point, FlashRoute can predict that the hop distances to both the blocks of 100.100.122/24 and 100.100.124/24 are also around 15. The proximity span for prediction is controlled by a configuration parameter we refer to as "proximity span"; by default, FlashRoute uses proximity span of 5, which means it uses the measured distance to a given block to predict distances of its 5 preceding and 5 succeeding blocks.

3.3.4 Accuracy of hop-distance prediction. Since the distance prediction is applied to the destinations that do not return ICMP port-unreachable messages, the accuracy of the prediction can not be verified using the triggering TTLs. Fortunately, with the default proximity span of 5, around 89.5% of the address blocks with measured distances are in the proximity of other blocks with measured distances. We then take our measured distance to one of these blocks as the predicted distance to the other blocks within the first

[1] We leave a more careful exploration of other potential values of this parameter to future work.

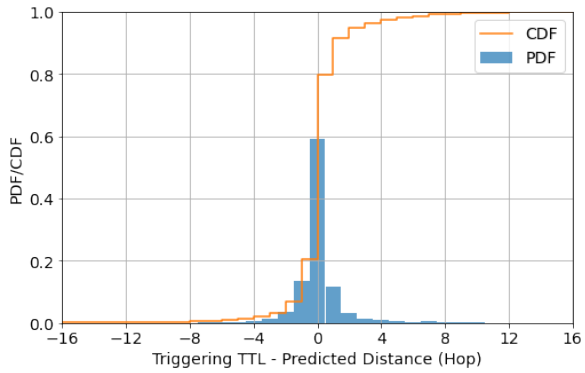

Figure 4: PDF and the CDF of the difference between measured and predicted distances of targets.

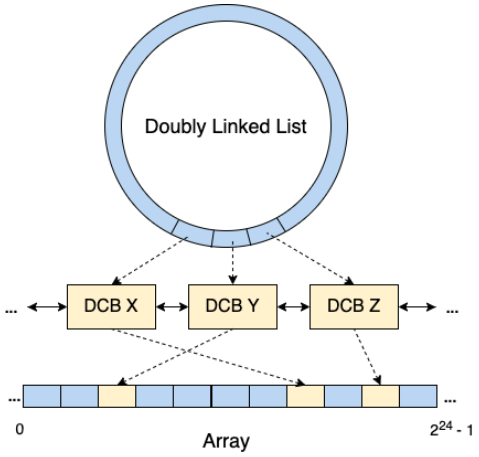

Figure 5: The control state structure.

block's proximity scope, and compare these predicted distances with the measured distances obtained through our technique.

Figure 4 provides the PDF and CDF of the differences between the predicted distances and the triggering TTLs that mimic traceroute-based measurements, for the same destinations. As can be seen, around 59.1% of predicted distances are equal to, and additional 25.4% are within one hop from, the traceroute distances.

Recall that FlashRoute's route exploration uses these predicted distances purely as an optimization hint; the accuracy of discovered routes does not depend on the accuracy of the prediction. Thus, and as long as our prediction is more often correct than wrong, the overall efficiency of route exploration improves. With over 84% of predictions being within 1 hop of traceroute-measured distances, we conclude that the predicted distances, calculated with the proximity span of 5, provide a useful hint when measured distances are absent.

3.3.5 Preprobing in FlashRoute. FlashRoute sets the split points of the routes using their measured or predicted hop-distances. For routes with unavailable distances, the split point remains the default 16 or a user-configured value.

FlashRoute has a special optimization when the default split-TTL of the main probing is set to 32, the same as the preprobing TTL. In this case, the preprobing can be taken as the first probing round of the main probing. This is because a response to preprobing is indistinguishable from the response to the same destination at TTL-32 during main probing. Therefore, when preprobing a destination at hop-32 produces no response or TTL-expired response, FlashRoute can directly take this outcome as the result of the first round and use it as part of the discovered topology. The main probing will therefore start at TTL 31 for this destination.

3.4 Control State

FlashRoute needs control state to support its probing logic. For each destination, it maintains a "destination control block (DCB)", shown in Listing 1, to keep track of the probing progress in both forward and backward directions as well as the "forward probing horizon", which limits the forward probing to at most *GapLimit* number of consecutive unresponsive hops. Variables *nextBackwardHop* and *nextForwardHop* represent, respectively, the TTL of the next backward and forward hop scheduled to be probed in the next round; these variables are accessed by the sending thread, and *nextBackwardHop* can also be updated by the receiving thread, which sets it to special value (zero) upon receiving the response that completes the backward scan (i.e., the response either from a hop at TTL 1 or from a previously discovered hop, indicating a convergence point). Variable *forwardHorizon* keeps value (*max_TTL*+*GapLimit*), where *max_TTL* is the TTL of the farthest hop from the source that has responded to a probe so far. It is updated by the receiving thread when processing responses and read by the sending thread. Thus, each DCB is protected by a mutex, but contention is highly unlikely as it only occurs when the sending thread happens to handle the destination at the same time when a response from a previous probe for this destination arrives. If *nextForwardHop* reaches *forwardHorizon*, it means the previous *GapLimit* number of hops have not responded and the forward probing stops.

FlashRoute's two threads place conflicting demands on the control state. The sending thread, in each round, must be able to efficiently move from DCB to DCB in the order determined by the random permutation, and also remove DCBs that finished probing from this sequence. The receiving thread, when a probe response arrives, must be able to quickly access the corresponding DCB to update its horizon or *nextBackwardHop*. Figure 5 illustrates our control state design to satisfy these requirements.

Like Yarrp, FlashRoute is designed to trace every /24 prefix. Consequently, FlashRoute keeps all DCBs in an array with 2^{24} slots, indexed by the /24 prefix of the destination IP address. Prefixes excluded from the scan still occupy their slots. This arrangement allows the receiving thread to immediately locate the DCB corresponding to an arrived response, based on the destination IP address extracted from the response.

In addition, we overlay a circular doubly linked list on the top of the array by having each DCB connect to successor and predecessor DCBs with pointers. Unlike the underlying physical array, in which probing states are ordered according to the prefixes of the destinations, the sequence of DCBs in the doubly linked list follows the random permutation order computed during the initialization phase. That means two consecutive DCBs on the linked list can be far from each other on the array, and a successor DCB in the list can

reside before its predecessor in the array. The circular list allows the sending thread to complete its rounds by moving from one destination to the next along the DCB links and also efficiently remove the destinations that finished tracing from the future rounds by simply removing the corresponding DCBs from the list.

The entire structure, along with the other overhead, such as mutexes, occupies around 900 MBytes memory for all /24 prefixes, the amount that is easily within the capacity of modern computers. There is a potential to reduce this footprint, most significantly by replacing general per-DCB mutexes with primitive atomic operations (such as a spinlock over the test-and-set instruction) but, given the overall memory consumption is modest, we opted for a more portable general approach.

Listing 1: Destination Control State

```
struct DestinationControlState {
    uint32_t destination;

    /* Probing progress information */
    uint8_t nextBackwardHop;
    uint8_t nextForwardHop;
    uint8_t forwardHorizon;

    /* Doubly Linked List Pointers */
    uint32_t nextIndex;
    uint32_t previousIndex;
    ...
};
```

This structure is initialized before the scan as follows. First, each DCB is initialized to include a destination IP address randomly generated from its /24 prefix (FlashRoute also has an option to load IP addresses from an exterior file instead but would still only use one address per /24 block.) Further, the variables to control the backward and forward probing are initialized using the default split point at TTL 16, or the value set by users, or the hop-distance obtained by preprobing (if this option is employed). Second, FlashRoute connects the DCBs into the circular doubly linked list to form a random permutation sequence of all /24 prefixes. Finally, all private, multicast, and reserved destinations, along with the destinations from the exclusion list, are removed from the doubly linked list before probing commences.

4 PERFORMANCE EVALUATION

We evaluate FlashRoute performance by studying the impact of its main design decisions and configuration parameters on its overall performance (§4.1) and by comparing its performance with representative existing alternatives, Yarrp, representing the state of the art in high-performance traceroute scans, and Scamper, an influential traceroute utility in current use for ongoing traceroute scans (§4.2). We consider the following metrics when comparing the tools. First, we conduct a full /24 scan with each tool to assess characteristics reflecting the tools' effectiveness for topology snapshot discovery: the number of interfaces discovered, the number of probes issued, and the scan time. Second, we compare the intrusiveness of the tools, by assessing how often their probing rate exceeds routers' ICMP response rate limit. Third, we compare the scalability of Yarrp and FlashRoute by considering the maximum scanning rate they can sustain.

Our testing environment is a Dell PowerEdge R610 server with Intel Xeon CPU E5620 @ 2.40GHz, 8 GB memory space, and 1 Gbps network connectivity. The capacity of this testbed can be easily met by most modern servers (or even laptops).

4.1 Impact of Features and Parameters

FlashRoute has three main features that may affect the performance: preprobing (including both hop-distance measurement of responsive targets and predicting the hop-distance of nearby unresponsive targets), backward probing, and forward probing. The preprobing helps FlashRoute decide where to place the split point on each route. The backward probing can increase the probing performance by removing the probing redundancy. The forward probing explores the route from the split point towards the target and allows conservative placement of the split point at an internal hop within the route. FlashRoute also has two configuration parameters, *GapLimit* and split TTL. This section assesses the impact of these features and parameters on the FlashRoute performance.

Table 1: Impact of redundancy elimination during backward probing.

Split-TTL	Redundancy removal	Interfaces	Probes	Scan time
32	On	805,472	164,882,469	27:54.19
32	Off	826,701	338,063,800	56:36.14
16	On	814,801	101,314,451	17:16.94
16	Off	817,509	257,983,117	43:33.55

4.1.1 Backward Probing. To quantify the impact of redundancy elimination during backward probing, we conduct full /24 scans with and without termination of backward probing at the route convergence points, when backward probing starts at TTL 16 and 32. The rest of configuration options are fixed to include preprobing with distance prediction based on proximity span 5 and forward probing with the gap limit 5.

Table 1 shows the performance metrics of these scans. Here and in the rest of the paper, the reported probes include the probes issued in preprobing phase if applicable, the scan time is the total time of the scan, again including any preprobing, and the number of interfaces reflects unique interfaces revealed in the scan. Redundancy elimination greatly reduces – by more than half – the number of probes and time required to conduct a scan. At the same time, while early termination of backward probing can miss interfaces on alternative routes (§3.2.1), the number of missed interfaces is very small, 2.5% and 0.3% for split-TTL 32 and 16, respectively. Thus, redundancy elimination brings substantial efficiency gains at a marginal cost in reduced function.

4.1.2 Forward probing and `GapLimit`. Forward probing of a route terminates after encountering the *GapLimit* number of successive silent hops. Using a large gap limit increases the chance to detect interfaces behind the silent stretch but increases the number of

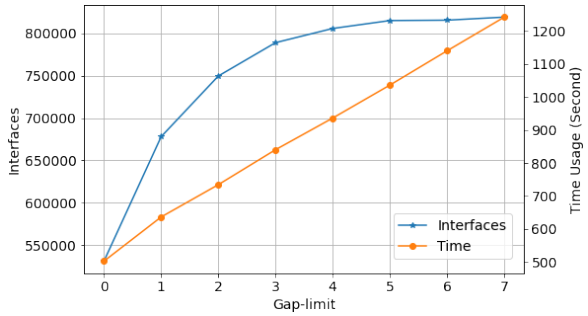

Figure 6: Discovered interfaces and scan time as a function of gap-limit.

unproductive probes and, consequently, the scan time. We quantify this trade-off below.

We test FlashRoute on full /24 scans varying the gap limit while keeping the rest of the configuration fixed (preprobing enabled with the prediction using proximity-span 5, backward probing with redundancy elimination enabled, split-TTL 16). When the gap-limit is 0, FlashRoute performs no forward probing. Figure 6 plots the number of discovered interfaces and scan time as the functions of the gap limit (the number of probes used correlates tightly with time and not shown). The scan time grows linearly with the increase of the gap limit but the number of discovered interfaces flattens considerably once the gap limit reaches 5. This suggests gap limit of 5 as a good default value (re-validating the default configuration of Scamper, which also uses this value).

4.1.3 Preprobing. We now assess the effect of preprobing on FlashRoute performance. With randomly picked representative IP addresses from each /24 block, only around 4.0% of the targets respond to our hop-distance measurement. Adding predictions for the neighboring prefixes, the distances to around 22.95% of the targets become available for FlashRoute's main probing phase. One way to increase the success rate of preprobing is to preprobe the destinations that are more likely to be responsive. The ISI Census Hitlist [18] maintains a list of addresses from each /24 block that are most responsive to ICMP pings. Using these IP addresses, preprobing is able to directly measure the distances to 10.0% of the targets and predict the distances to further 28.2% of the targets, for the total coverage of 38.2% of all prefixes.

We have found, however, a bias in the Census hitlist that affects the topology measurements (§5.1). Thus, in our experiment, we use the hitlist only for the distance measurement during preprobing, and use random addresses for each /24 block for the actual probing. Although the hitlist bias will result in slightly shorter hop-distances, they are still useful as performance hints in FlashRoute, while using random destinations for the actual probing will avoid the bias in the discovered topology. This arrangement, however, precludes the optimization from §3.3.5 as the result of preprobing is no longer identical to the first round.

To verify how the preprobing affects the performance of FlashRoute, we run six experiments. Three of them use split-TTL 16 and three use split-TTL 32. For either split-TTL value, we conduct a scan with preprobing based on the hitlist ("hitlist preprobing"), a scan with preprobing using randomly generated destinations ("random preprobing"), and a scan without preprobing. All scans use distance prediction with the proximity span of 5 whenever preprobing is employed, forward probing with gap limit 5, and redundancy elimination during backward probing.

Table 2: Effect of preprobing on FlashRoute performance. The number in the configuration column refers to the default split-TTL.

Configuration	Interfaces	Probes	Scan Time
32/hitlist preprobing	807,588	159,185,459	27:31.85
32/random preprobing	805,472	164,882,469	27:54.19
32/no preprobing	799,562	181,757,638	30:48.48
16/hitlist preprobing	812,403	97,807,092	17:16.56
16/random preprobing	814,801	101,314,451	17:16.94
16/no preprobing	802,524	96,687,844	16:39.06

Table 2 shows the effect of preprobing on FlashRoute performance. For default split-TTL 32 with random preprobing, preprobing does not entail extra probes because these probes replace the first round of the main probing phase. Thus, it leads to pure savings in terms of the number of probes and the scan time. With this, random preprobing saves 10% of the probes and 10% of the scan time, as shown in table. While folding preprobing into the main probing cannot be applied for hitlist preprobing, hitlist preprobing produces more estimates for split-TTL, which overcomes the extra overhead. Thus, hitlist preprobing brings even slightly higher savings, cutting 12% of the probes and 11% of the scan time. With the default split-TTL 16, when destinations fail to produce the hop-distance estimate, the probes send in the preprobing phase are wasted for both random and hitlist preprobing. The end-result is that the overhead outweighs the improvement.

The number of discovered interfaces is affected by preprobing because preprobing can change split-TTL for destinations and therefore the balance between backward and forward probing. Backward and forward probing miss interfaces in different ways: backward probing can miss alternative routes preceding convergence points (as in Figure 2) while forward probing can miss interfaces behind a silent stretch. On the balance, preprobing seems to slightly increase interface discovery.

Finally, the default split-TTL of 16 is significantly better overall than 32, with or without preprobing, in terms of the number of discovered interfaces, the number of probes, and the scanning time. The reason is that for the destinations that fail to produce hop-distance estimates during preprobing, TTL-32 has to probe backwards through long stretches of non-responsive routers, especially when routes to these destinations have a long unresponsive tail, whereas forward probing in TTL-16 stops exploration after encountering GapLimit of non-responsive routers in a row.

Overall, considering all the factors above, including scan time, probe volume, and interface discovery, default split-TTL of 16 with preprobing based on hitlist appears to be the most beneficial configuration for FlashRoute (but see §4.2.2 for further considerations).

4.2 Comparison with Exiting Solutions

We compare performance of FlashRoute with Yarrp and Scamper. We note that Scamper was designed with different goals from Yarrp and FlashRoute. While Yarrp and FlashRoute are designed for massive fast scans from a single vantage point, Scamper is geared for long-term network measurement held from multiple vantage points to draw a full picture of the topology of the Internet. For this reason, Scamper optimizes the scan to reduce probing redundancy not only from the local vantage point but also across vantage-points, using the Doubletree algorithm. We include Scamper in our study as it has been an influential tool, long representing the state of the art even for the massive scans from a single vantage point before Yarrp emerged.

4.2.1 Scan Efficiency. We run full /24 scans with each tool using the following configurations. Given the results from §4.1, we consider two configurations for FlashRoute, one with the default split-TTL 16, gap limit 5, redundancy removal enabled, and preprobing using destinations from the hitlist [18] (referred to FlashRoute-16) and the other with the default split-TTL 32 and the rest of configuration the same as FlashRoute-16 (FlashRoute-32). We use split-TTL 16 ("first-TTL" in Scamper's terminology), maximum TTL 32, and the default gap limit 5 for Scamper but restrict its retry on failure to ensure it behaves similarly to FlashRoute and Yarrp, issuing one probe per hop. We also consider two configurations of Yarrp, one using the split-TTL 16 with fill-mode enabled up to the maximum distance 32 (recommended configuration in the Yarrp6 study [5], denoted Yarrp-16), and the other with the maximum TTL 32 (as proposed in the Yarrp original paper [4], denoted Yarrp-32). The probing speed of both Yarrp and FlashRoute is set to 100Kpps, the limit used by Yarrp's study, which we also were able to negotiate with our network administrators. Since Scamper limits the maximum probing speed to 10 Kpps, we test it at its maximum speed.

Finally, both Scamper and FlashRoute use the Paris-UDP probes. Yarrp, on the other hand, uses Paris-TCP-ACK probes by default and experiences an error when using the UDP probes[2]. Therefore, we use the default Paris-TCP-ACK probes for Yarrp, but in addition, we simulate the Yarrp-32 scan with Paris-UDP using FlashRoute without preprobing or forward probing and without terminating backward probing at convergence points. In this mode, FlashRoute issues one probe to every hop from 1 to 32 for all destinations just as Yarrp-32 would. Given that both the set of probes and the sending rate are the same in both cases, and both tools harvest interfaces from the responses to the same set of probes, we take FlashRoute's scan results in this mode for Yarrp-UDP[3].

Table 3 provides the results. As can be seen, FlashRoute-16 uses the least time and fewest probes to finish the scan. FlashRoute-32 uses around 61% more time and probes than FlashRoute-16 but is still faster than the existing tools. Yarrp-16 uses almost twice as much time as FlashRoute-16 to complete its scan, yet it can only discover less than half the number of interfaces obtained by the other tools, so it has limited utility as a topology discovery tool. We believe this behavior is mainly caused by premature termination of forward probing in Yarrp's fill mode. Yarrp's stateless design entails an inherent gap limit of 1 in forward probing because the next probe forward is triggered by the arrival of a TTL-expired response from the hop at the current maximum distance. As a result, Yarrp's forward probing stops after encountering a single silent interface, missing any interfaces beyond. Yarrp-32 with either Paris-TCP-ACK probes or (simulated) Paris-UDP probes finishes the scan using around one hour, the same as reported in the Yarrp original paper [4]. Not surprisingly, Scamper is the slowest, taking more than 3.5 hours to complete the scan at its maximum speed.

Table 3: Performance of FlashRoute, Yarrp, and Scamper on a full /24 traceroute scan.

Tool	Interfaces	Probes	Scan Time
FlashRoute-16	812,403	97,807,092	17:16.56
FlashRoute-32	807,588	159,185,459	27:31.85
Yarrp-16	393,433	177,851,221	30:14.71
Yarrp-32	801,455	355,702,000	1:00:15.21
Scamper-16	819,149	131,833,846	3:43:27.56
Yarrp-32-UDP (Simulation)	829,387	355,701,952	59:58.40

Turning to the number of discovered interfaces, all tools except Yarrp-16 discover between 801,455 and 829,387 interfaces during a complete /24 IPv4 scan. The differences in the numbers of discovered interfaces can be attributed to two main reasons.

First, according to a previous study [16], the probe type will affect the interface discovery, and UDP probes can generally detect more interfaces than probing with TCP packets. This can explain why Yarrp-32 discovers fewer interfaces than the other tools. In particular, Scamper discovers 2.1% more interfaces than Yarrp-32 in our experiments, a slightly greater difference than observed in the original Yarrp study, which compared this metric with both tools using TCP-ACK probes.

Second, removing probing redundancy through backward probing with termination at convergence points reduces the number of discovered interfaces by missing alternative routes (§3.2.1). Indeed, both FlashRoute-16 and FlashRoute-32 discover fewer interfaces (by, resp., 2.0% and 2.6%) than Yarrp-32-UDP simulation because FlashRoute employs this optimization while Yarrp-32-UDP does not.

Interestingly, Scamper-16 discovers around 0.8% more interfaces than FlashRoute-16 even though both Scamper-16 and FlashRoute-16 use the same split-TTLs and presumably adopt the same redundancy removal optimization from Doubletree. While the difference is small, we wanted to understand the reason behind it, and we found that, even though Scamper claims it uses the Doubletree backward probing, its implementation is actually different. Figure 7 provides the distribution of the number of targets whose routes are explored by Scamper and FlashRoute-16 at different TTLs. As the figure shows, FlashRoute-16 progressively terminates route

[2]With UDP, Yarrp encodes part of the elapsed time in the packet length header field, quickly outgrowing the maximum packet size allowed by the system, resulting in "Network API error: Message too long".

[3]Note that while the set of probes is the same, the *order* in which they are issued is different. In particular, due to Yarrp's randomized order, nearby routers are less likely to exceed ICMP rate limit in Yarrp than in FlashRoute in this mode. However, an over-limit interface would omit responses only after responding to probes that arrive within the limit and thus would highly likely be discovered in FlashRoute as well.

exploration at TTLs below 16 due to redundancy elimination. Scamper, however, starts removing redundancy one hop later, and then preserves a certain level of probing redundancy until the TTL reduces to 6, as the blue curve turns flat from 14 to 6. The curve suddenly plunges at TTL 6, converging to that of FlashRoute, indicating return to full redundancy elimination. As a result, Scamper sends more probes than FlashRoute in the backward probing and thereby has a higher chance to discover router interfaces on alternative routes. The overall effect is that Scamper discovers 0.8% more interfaces at the cost of 34.7% more probes than FlashRoute-16.

We note that Yarrp has an optimization to reduce the probes sent to routers near the source called neighborhood protection. This optimization stops issuing probes to the nearby routers at a given hop-distance from the source if no new interfaces are discovered for the default of 30 sec. To examine the effect of this optimization, we run two extra scans with the same configuration as Yarrp-32 but with neighborhood protection enabled, protecting routers within, respectively, three and six hops from the vantage point. The result shows that 3-hop neighborhood protection reduces probe volume by 6.3% (around 22,410,163 probes) and scan time to 57 minutes while 6-hop neighborhood protection reduces probe volume by 15.7% (around 55,846,057 probes) and scan time to 51 minutes. This improvement, however, comes at cost of impairing the completeness of interface discovery within the neighborhood. We find that 3-hop neighborhood protection misses 20% (5 out of 25), and 6-hop neighborhood protection misses 35.6% (98 out of 275) of interfaces in their respective neighborhoods.

In summary, FlashRoute conducts scans three and a half times faster, and finds marginally more interfaces, than the best operational configuration of Yarrp, which represents the prior state of the art in massive traceroutes. Scamper finds 0.8% more interfaces but takes over 13 times longer, and uses 34.7% more probes, to complete its scan. Thus, FlashRoute offers experimenters significant new capabilities in taking Internet topology snapshots at finer time-granularity. Moreover, since most missed interfaces are from the alternative paths, FlashRoute can use some of the time saved to run multiple probing cycles with different source port numbers to explore alternative paths (see §5.2).

4.2.2 Scan Intrusiveness. Both Yarrp and FlashRoute have a high probing rate. As more probes are sent per unit of time, the risk to trigger the ICMP rate limiting also rises – the occurrence we refer to as "router overprobing". Higher frequency of router overprobing makes a scan more intrusive to regular network operation and less precise in its topology discovery as affected routes will miss otherwise responsive hops. Yarrp sends its probes in totally random order to reduce the risk of overprobing a router – this makes Yarrp less likely to send too many probes to the same router in a quick succession. FlashRoute, on the other hand, probes routes deterministically backward and forward from the split point but reduces redundant probing of the same routers through early termination of backward probing at convergence points. This section compares the extent of router overprobing by the two tools during a full /24 scan.

Unfortunately, we cannot directly observe incidents of router overprobing during the scans because we cannot know the reason why a hop has not responded to a probe. Therefore, we run a simulation using the probes issued by FlashRoute and Yarrp in real scans at 100Kpps but mapping them to the topology discovered by Scamper at the probing speed of 10Kpps. Specifically, we assume that a probe specifying a given destination IP address and a given TTL, issued by either tool, will experience the "TTL expired" event at the hop discovered by Scamper for the same destination address at the same TTL distance. We then consider real timing of each probe issuance by each tool and consider mapped hops from the Scamper topology to be overprobed if they are issued more probes than the ICMP rate limit in each one-second interval of the tool's scan duration. In this analysis, we assume the ICMP rate limit is 500 pps for each router interface, which is the upper bound of ICMP rate-limiting for most devices [19]. We consider both FlashRoute-16 and FlashRoute-32 and compare them with Yarrp-32 and its two variants, with 3-hop and 6-hop neighborhood protection (denoted Yarrp-32 3-hop protection and Yarrp-32 6-hop protection, resp., for brevity). We do not consider Yarrp-16 given the scan effectiveness results from §4.2.1.

The results of this experiment are summarized in Table 4. Table 4 shows that both Flashroute configurations are significantly less intrusive than Yarrp-32. While FlashRoute-16 lowers the number of overprobed interfaces compared to Yarrp-32 by 41.9%, it also lowers the severity of overprobing when it does happen, as the total number of dropped probes in FlashRoute-16 is just 27% of those in Yarrp-32. FlashRoute-32 further drastically decreases overprobing

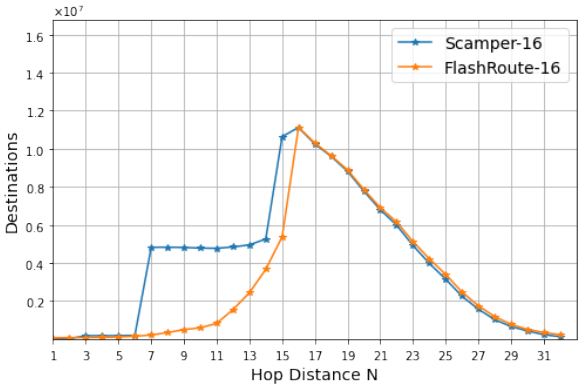

Figure 7: The distribution of targets with routes probed at a given TTL.

Table 4: Interface overprobing.

Tool and Configuration	Overprobed Interfaces	Dropped Probes
FlashRoute-16	5,746	14,569,275
FlashRoute-32	3,091	8,312,385
Yarrp-32	9,895	53,813,793
Yarrp-32 3-hop protection	9,903	53,792,883
Yarrp-32 6-hop protection	9,886	53,364,491

Table 5: Non-throttled scan speed.

Tool and Configuration	Scan Speed (Kpps)	Estimated Scan Time
FlashRoute-32	302.826/228.929	11:23.4
FlashRoute-16	302.826/215.573	6:55.38
Yarrp-32	239.088	24:47.74
Yarrp-16	189.704	15:37.51

and is by far the least intrusive of the five configurations. In particular, it overprobes 3.2 times fewer routes, and loses 6.4 times fewer probes, than Yarrp-32. Thus, the deterministic route exploration of FlashRoute is more than compensated by early termination of backward probing in terms of the amount of overprobing, and the aggressive application of early termination by FlashRoute-32 produces especially strong effect in this regard.

Interestingly, we find that Yarrp's neighborhood protection does not help in reducing the overall scan intrusiveness of Yarrp. This can be attributed to two reasons. First, the number of interfaces in the protected neighborhood is small and extensively probed; as a result, they are already overprobed before the protection is activated. Second, because the overall probing rate is fixed at 100Kpps, neighborhood protection can intensify probing of non-neighboring hops, potentially exacerbating the severity of their overprobing. We note that our analysis does not capture a positive effect that Yarrp's neighborhood protection can reduce the severity and duration of neighborhood overprobing. However, because this effect comes at the cost of missing a significant fraction of the neighborhood interfaces (§4.2.1), we use Yarrp-32 for the rest of the paper.

The above results also uncover an interesting trade-off between FlashRoute-16 and FlashRoute-32. On the one hand, FlashRoute-16 completes a scan considerably faster and discovers virtually the same number of interfaces with considerably fewer probes. On the other hand, FlashRoute-32 is much less intrusive and loses many fewer probes; hence, while both configurations find the same total number of *interfaces*, the *routes* discovered by FlashRoute-32 will have fewer holes. Thus, an experimenter may choose the configuration based on the measurement priorities: if the goal is to approximate a snapshot in time as closely as possible, they should use FlashRoute-16; if the main concern is to obtain the most complete routes to destinations, FlashRoute-32 with preprobing seems a better choice.

4.2.3 Tool scalability. All our scans so far were conducted at 100K probes per second, the limit we have negotiated with our network administrators, which is also the rate used in the Yarrp study. We have seen that both tools can sustain this rate. We now assess the *potential* efficiency of both tools, for someone who might be able to negotiate a higher probing rate. To this end, we received a permission to run each tool for a limited time without artificial throttling. Thus, we start a full /24 scan with each tool at full speed, run each scan for the allowed time (5 minutes), and measure the probing rate each tool exhibits.

Table 5 lists the probing rate observed for different tools during these limited runs, as well as the estimated time for a full scan completion at this rate, given the number of probes each tool issues (see Table 3). The two numbers for FlashRoute-32 and FlashRoute-16 reflect probing rates during preprobing and main probing phases. The results show that Yarrp-32 can support slightly higher probing rate than FlashRoute, 11% higher than FlashRoute-16 and 4% higher than FlashRoute-32 in the main phase. This improvement, however, cannot compensate for the fewer probes FlashRoute issues, and it has only negligible effect on FlashRoute's advantage in total scan time. In particular, FlashRoute-16 can complete a full /24 IPv4 scan in less than 7 minutes. For completeness, we include the results of Yarrp-16; its probing rate is actually lower than the rest of the tools, presumably due to the slow fill-mode.

The above results are meant to assess the maximum scalability of the tools. Thus, they were obtained using the most performant regime in each tool. FlashRoute offers an option to exclude response logging, relegating this task to an external sniffer, while Yarrp's internal logging is built in. Consequently, the results in Table 5 are obtained without logging for FlashRoute and with logging for Yarrp. We also tested FlashRoute with internal logging, to make it directly comparable to Yarrp. While the preprobing rate drops significantly (to 219.432 Kpps for both FlashRoute configurations), the main probing phases achieve roughly the same probing rate (234.791 Kpps for FlashRoute-32 and 214.751 Kpps for FlashRoute-16). Because preprobing is a very small part of the overall scan, the overall estimated scan time remains similar, 11:21.2 for FlashRoute-32 and 7:34.2 for FlashRoute-16.

4.2.4 Summary of Comparative Evaluation. Our results show that FlashRoute cuts the time to complete a full /24 Internet traceroute scan by over 3 times, saves a similar amount of probes, and is less intrusive to networks under measurement than Yarrp, the current state of the art in massive traceroutes. While FlashRoute does reintroduce control state to achieve these results, having to maintain this state has negligible effect on its overall performance: running on a low-capacity 8 year old server[4], it can sustain over 200K probes per second during its scans, just 4-11% lower than Yarrp, which requires many more probes to complete the same scans.

5 DISCUSSION

We made some interesting observations in the course of our experiments with FlashRoute. We discuss some of these observations, as well as our plans for future work, below.

5.1 On Census Hitlist Bias

The Census Hitlist [18] is generated by a long-running experiment which uses ICMP pings to detect the most responsive destination in each routable IPv4 /24 prefix. When we used the addresses from the hitlist as representatives for each /24 prefix in our scans, we observed that the number of interfaces discovered during a scan decreases significantly compared to scanning random representatives. In particular, the exhaustive probing of every hop from 1 to 32 of each destinations discovers 829,338 interfaces when each /24 prefix is represented by a random destination (referred to as the "random

[4]The one server within our campus DMZ from which we are able to conduct massive scans.

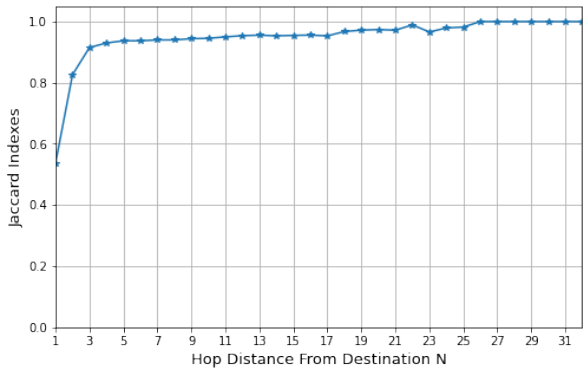

Figure 8: Jaccard index of the interface sets at a given hop-distance from the destinations, produced by the scans using the hitlist and random targets.

scan" below) and only 759,961 interfaces when representatives from the hitlist are used (the "hitlist scan").

In an attempt to understand the reason behind the difference, we compare the similarity between the sets of interfaces discovered by the two scans at each hop-distance from the destinations. Figure 8 shows the Jaccard index of the two sets of interfaces at each hop distance from the destination. Jaccard index measures the similarity between sets of elements, ranging from 1 for identical sets to 0 for totally different sets. As shown in Figure 8, the main difference in the composition of discovered interfaces between two scans is at the two hops next to the destinations.

We further observe that the lengths of the discovered routes tend to be shorter in the hitlist scan than in the random scan: 1,515,626 routes to random destinations are longer than routes to hitlist destinations from the same /24 prefixes, while the reverse is true for only 1,349,814 routes. We hypothesized that this difference of route lengths is the main cause to the difference in the numbers of discovered interfaces. Therefore, we compare the unique interfaces on the extra route parts to the same prefix in the two scans. The result shows that the random scan discovers 57,532 more unique interfaces on extra parts of its longer routes than the hitlist scan, roughly matching the difference in the total number of discovered interfaces between two scans (69,377).

As a potential factor contributing to this phenomenon, we speculate that the hitlist tends to prefer gateway appliances (routers, firewalls, NAT boxes, etc.) in stub networks as they are more likely to respond to pings than internal hosts. Therefore, tracerouting hitlist destinations may not explore the routes in the stub networks interior. Random destinations, on the other hand, are equally likely to select any address within a address block and do not exhibit this bias.

To find evidence for this explanation, we consider how many hitlist destinations appear as intermediate hops on the routes to random destinations within the same /24 prefixes, and, reversely, how many random destinations appear as intermediate hops on the routes to hitlist destinations. We find 27,203 hitlist addresses en-route to random destinations, and only 6,421 random destinations on the paths to hitlist destinations. There are 1,273,230 hitlist destinations and 540,060 random destinations that respond to preprobing. Thus, while responsive random destinations constitute 42% of responsive hitlist destinations, the number of random destinations appearing on the routes to hitlist destinations represents only 24% of the hitlist destinations appearing on the routes to random destinations. Thus, hitlist destinations are indeed more likely to be routers on address block periphery although their absolute number appears to be too small to fully explain our finding.

An alternative explanation for the observed bias could be that, since the hitlist destinations are by design more likely than random ones to exist, they are less prone to a potential network configuration bug, where the stub network forwards packets to non-existent addresses along a default route back towards its ISP (in the same way it forwards packets to external destinations). This could result in a loop, when the ISP routers return the packets back to the stub network, or in dropped packets, if the ISP employs ingress address filtering, but the end result in both cases would be a longer route measured for non-existent destinations.

To assess this possibility, we test if the observed bias would still be present if we only considered /24 prefixes in which *both* hitlist and random destinations were responsive to our probes, i.e., both existed. Our dataset contains 294,123 /24 prefixes where both the hitlist and random targets were responsive. In 64,279 of these prefixes, random targets have a longer hop distances than hitlist targets, and only in 34,057 prefixes the paths to the hitlist targets are longer. Thus, the bias we observed for the full set of prefixes remains even for the prefixes in which both hitlist and random targets respond to our probes. Further, as a side observation, we directly assess the prevalence of the loops in the paths leading to unresponsive random targets. We have 971,113 prefixes in our dataset with responsive hitlist and unresponsive random targets. Of these, in 16,549 prefixes, or 1.7%, routes to (unresponsive) random targets include a loop. Thus, these loops do exist but are rare.

In summary, one must be cognizant of the Census Hitlist bias when using it for Internet topology explorations. In particular, as mentioned earlier, FlashRoute only uses this hitlist for preprobing and reverts to random destinations for the main probing.

5.2 Discovery-Optimized FlashRoute

FlashRoute cuts the amount of time required to conduct a full /24 IPv4 traceroute scan to 7 minutes at maximum probing speed, likely less on a more capable server. It can optionally use some of the saved time to discover load-balanced alternative routes by issuing extra probes with different source port numbers. We refer to this option as the discovery-optimized mode.

In this mode, FlashRoute begins by conducting a normal FlashRoute-32 scan. Using split-TTL 32 explores most parts of the routes by backward probing; as a result, the "stop set" of potential convergence points used for backward probing termination contains the majority of discovered interfaces. After that, FlashRoute runs a given number of extra scans, using backward probing only, starting from the initial TTL randomly picked from 1 to 32 for each destination. For i-th extra scan, probes for a given destination use source port number $P' = P + i$, where P is the source port used by the initial regular scan for this destination (recall that it is checksum

of the destination IP address). A different port number prompts per-flow load-balancers to route probes through different interfaces in each extra scan. Random initial TTLs allow the discovery of load-balanced path sections affecting early (i.e., near the vantage point) parts of the routes. Since the stop set is shared across main scan and extra scans, extra scans explore only previously undiscovered routes as they stop once encountering observed interfaces. As a result, extra scans take much less time than the normal scan but have a high chance to discover alternative routes.

We run a scan using the discovery-optimized mode of FlashRoute with three extra scans. The entire scan takes 56 minutes to finish at 100Kpps. FlashRoute discovers 865,339 interfaces, or 35,952 more than what simulated Yarrp-32-UDP can discover in the same amount of time, or 63,884 more than real Yarrp-32 with the default configuration detects.

5.3 In-flight Address Modification

Some middleboxes modify destination address of packets on their way to the target. An earlier study reported this affected around 2% of the probes in their experiment [17]. Since FlashRoute uses the checksum of the destination address as probe's source port, we can check whether or not the destination address of the target has been changed in-flight by observing a mismatch between the checksum and the destination address from the quoted probe header in the response, and drop the responses exhibiting mismatches. The number of mismatches we observe varies by scans, ranging from 0.007% to 0.054%, but is significantly lower than in [17]. Besides possible Internet changes in the time between the two studies, another reason could be the difference in the methodology: while [17] used only around 84k IP addresses of web servers as the target set and TCP-SYN probes, our experiments probe destinations representing every /24 block using UDP packets. Thus, we believe our results reflect more current and general behavior.

5.4 Limitations and Future Work

A current FlashRoute limitation is that it probes only one IP address from each /24 block. We chose to base FlashRoute on this assumption because (a) it's the common approach for topology scans, which FlashRoute aims to improve (e.g., CAIDA's Skitter measurements follow this approach, and Yarrp is also built on this assumption), and (b) most BGP routers don't accept advertisements for longer than /24 prefixes [1]; so, from the perspective of external routes to the destination, /24 seems sufficient granularity. However, the autonomous system holding a given /24 subnet may in principle use different internal paths for different addresses within this /24, which would be left undiscovered by probing only one address. There are two ways FlashRoute could use to mitigate this limitation.

First, FlashRoute could scan at finer granularity with still feasible memory consumption. Since its memory footprint is overwhelmingly determined by our main control data structure, which grows exponentially with the prefix length, scanning one address per up to /28 block would still be entirely feasible as it would only require < 15GB memory. Going beyond that, however, quickly makes memory demands impractical, up to 230GB for a complete /32 scan.

Second, discovering these additional internal paths could also be done within the "Discovery-Optimized Mode" (Sec. 5.2), that could vary not just the source port number but also the destination address within each block in the additional scans. Which approach is more productive for finding those additional internal paths (i.e., extending the initial targets to one per /28 or discovery-optimized mode with varying target addresses) is an interesting question for future work.

In the future, we plan to extend FlashRoute to tracerouting IPv6 address space. This will require a redesign of the control state: as the allocated IPv6 addresses are sparsely assigned [20], FlashRoute will no longer be able to keep DCBs in an array indexed by address prefixes. There are also several further avenues for optimization that we would like to explore. First, our current choice of the default value for proximity span is rather arbitrary and may not be the optimal choice. We plan additional experiments to find a substantiated recommended value, which can potentially increase the coverage of distance prediction and hence further improve the tool efficiency. Second, the current discovery-optimized mode randomizes the starting point of backward probing within 32 hops for extra scans. We will explore more purposeful heuristics to guide the starting point selection. For example, if we find the route length in the main scan to be 20, our random starting point in the extra scans could be selected from the range of 1 through, say, 25 instead of 32, since alternative routes may not drastically change the route length – saving seven backward probes.

6 CONCLUSION

In this paper, we propose FlashRoute, a new tool for large-scale traceroute scans, which cuts the time needed to complete a full /24 Internet scan by over three times from the current state of the art. Furthermore, FlashRoute completes its scans with fewer probes and is less intrusive to networks under measurement. If allowed to scan at full speed, FlashRoute can obtain a snapshot of the Internet routes from a given vantage point in just seven minutes, when running on a low-capacity server. Other contributions of this study include a new simple method of measuring hop-distance to a destination with a single probe, and a finding of bias in a popular hitlist of Internet destinations, which can affect topology-related studies. We will make FlashRoute prototype and all the data from our experiments public after the double-blind review.

ACKNOWLEDGMENTS

This work was supported by NSF under Grant CNS-1647145. We would like to thank the anonymous reviewers and our shepherd, Neil Spring, for helpful comments and suggestions, which resulted in significant paper improvements. We are grateful to Case Western Reserve University's ITS department for their support for our experiments. We in particular would like to acknowledge Debbie Andrews for monitoring the network during our numerous scans in the wee hours.

REFERENCES

[1] Emile Aben and Colin Petrie. 2014. Propagation of Longer-than-/24 IPv4 Prefixes. RIPE Labs; available at https://labs.ripe.net/Members/emileaben/propagation-of-longer-than-24-ipv4-prefixes.

[2] Brice Augustin, Xavier Cuvellier, Benjamin Orgogozo, Fabien Viger, Timur Friedman, Matthieu Latapy, Clémence Magnien, and Renata Teixeira. 2006. Avoiding traceroute anomalies with Paris traceroute. In *Proceedings of the 6th ACM SIGCOMM Internet Measurement Conference*. 153–158.
[3] Brice Augustin, Timur Friedman, and Renata Teixeira. 2007. Multipath tracing with Paris traceroute. In *Workshop on End-to-End Monitoring Techniques and Services*. IEEE, 1–8.
[4] Robert Beverly. 2016. Yarrp'ing the Internet: Randomized high-speed active topology discovery. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*. 413–420.
[5] Robert Beverly, Ramakrishnan Durairajan, David Plonka, and Justin P Rohrer. 2018. In the IP of the beholder: Strategies for active IPv6 topology discovery. In *Proceedings of the Internet Measurement Conference*. 308–321.
[6] UCSD CAIDA. 2007–2020. The IPv4 Routed/24 Topology Dataset.
[7] Benoit Donnet, Bradley Huffaker, Timur Friedman, and Kimberly C Claffy. 2007. Increasing the coverage of a cooperative Internet topology discovery algorithm. In *International Conference on Research in Networking*. Springer, 738–748.
[8] Benoit Donnet, Philippe Raoult, Timur Friedman, and Mark Crovella. 2005. Efficient algorithms for large-scale topology discovery. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. 327–338.
[9] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. ZMap: Fast Internet-wide scanning and its security applications. In *{USENIX} Security Symposium ({USENIX} Security 13)*. 605–620.
[10] FlashRoute. Sept 17 2020. FlashRoute Data Download. Available upon request due to large size.
[11] FlashRoute. Sept 17 2020. FlashRoute Github Page. https://osf.io/kw5jr/.
[12] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. 2019. Zooming in on wide-area latencies to a global cloud provider. In *Proceedings of the ACM Special Interest Group on Data Communication*. 104–116.
[13] Christian Kreibich, Mark Handley, and V Paxson. 2001. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *USENIX Security Symposium*, Vol. 2001.
[14] Bo Li, Jingsha He, and Henghua Shi. 2008. Improving the efficiency of network topology discovery. In *The 3rd International Conference on Grid and Pervasive Computing-Workshops*. IEEE, 189–194.
[15] Matthew Luckie. 2010. Scamper: a scalable and extensible packet prober for active measurement of the Internet. In *Proceedings of the 10th ACM SIGCOMM Internet Measurement Conference*. 239–245.
[16] Matthew Luckie, Young Hyun, and Bradley Huffaker. 2008. Traceroute probe method and forward IP path inference. In *Proceedings of the 8th ACM SIGCOMM Internet Measurement Conference*. 311–324.
[17] David Malone and Matthew Luckie. 2007. Analysis of ICMP quotations. In *International Conference on Passive and Active Network Measurement*. Springer, 228–232.
[18] USC/LANDER project. Jan 30 2019. Internet Addresses Hitlist Dataset. USC-LANDER/internet_address_hitlist_it84w-20190130/rev10351.
[19] Riccardo Ravaioli, Guillaume Urvoy-Keller, and Chadi Barakat. 2015. Characterizing ICMP rate limitation on routers. In *IEEE International Conference on Communications (ICC)*. IEEE, 6043–6049.
[20] Justin P Rohrer, Blake LaFever, and Robert Beverly. 2016. Empirical study of router IPv6 interface address distributions. *IEEE Internet Computing* 20, 4 (2016), 36–45.
[21] Yuval Shavitt and Eran Shir. 2005. DIMES: Let the Internet measure itself. *ACM SIGCOMM Computer Communication Review* 35, 5 (2005), 71–74.
[22] Neil Spring, Ratul Mahajan, David Wetherall, and Thomas Anderson. 2004. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Transactions on Networking* 12, 1 (2004), 2–16.
[23] Kevin Vermeulen, Stephen D Strowes, Olivier Fourmaux, and Timur Friedman. 2018. Multilevel MDA-lite Paris traceroute. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference*. 29–42.

APPENDIX: ETHICS

FlashRoute scans the Internet at a high speed. We employ best practices of good Internet citizenship to minimize the intrusiveness of our experiments. First, we coordinate our experiments with our ITS department, so they can respond quickly to any complaints. Second, we include a text message in probe payload to disclose our identity and contact information as well as research intention. Third, the server from which we run the scans has a reverse DNS record, which can be used to obtain the email address of the ITS department from the Whois database. Any complaints regarding our experiments submitted to this email are also forwarded to us. Fourth, we randomize the probing sequence to reduce risk of overprobing a given network. Throughout the development and testing, we received 7 complaints in total, all finding us from the Whois lookup. We promptly added the involved addresses to our exclusion list thus removing them from all future experiments.