Failure Control in Multipath Route Tracing

Darryl Veitch*, Brice Augustin†, Renata Teixeira†, and Timur Friedman†
*ARC Special Centre for Ultra-Broadband Information Networks (CUBIN), an affiliated program of National ICT Australia (NICTA), The University of Melbourne, Australia. dveitch@unimelb.edu.au
†UPMC Paris Universitas (Univ Paris 06) and CNRS, LIP6 Laboratory, Paris, France.
brice.augustin@lip6.fr, renata.teixeira@lip6.fr, timur.friedman@lip6.fr

***Abstract*—Traceroute is widely used to report the path packets take between two internet hosts, but the widespread deployment of load balancing routers breaks a basic assumption – that there is only a single such path. We specify an adaptive, stochastic probing algorithm, the Multipath Detection Algorithm (MDA), to report all paths that probes can follow between a source and a destination. We establish the foundations of, and show how to calculate, rigorous statistical guarantees for the discovery of the entire multipath route. We explore algorithm cost/guarantee tradeoffs in real experiments and show the inadequacy of the classic practice of sending three probes per hop.**

I. Introduction

Traceroute [10] is widely used for diagnosing network problems and to assemble internet maps. It discovers the route between two machines by issuing a series of probes with increasing time-to-live (TTL) values from a source to a destination. Unfortunately, traceroute measurements can be inaccurate and incomplete when the measured route traverses a load balancing router, or *load balancer* [1]. Consider the example of a multipath route between hosts S and T in Fig. 1, where L is a load balancer. If the traceroute probes with TTL=2 take the upper path and the probes with TTL=3 take the lower one then traceroute will report the existence of a false link between A and D. Alternatively, the probes might all follow the same path and traceroute would report only one path between S and T, when there are in fact two.

Two types of load balancers cause problems with traceroute. Per-flow load balancers are widespread [2]: they ascribe each packet to a flow defined by a *flow identifier*, which is the header five-tuple (i.e., IP source and destination addresses, source and destination ports, and protocol), and each flow to an outgoing interface. Per-packet load balancers are much rarer [2]: they assign packets to interfaces regardless of flow. A new traceroute implementation called *Paris traceroute* [1] maintains a constant flow identifier in all the probes it sends and hence solves the problem of inferring false links under per-flow load balancing. With Paris traceroute one can accurately trace the single path that a single flow takes from a source to a destination. However, to merely trace a single path from source to destination is to miss the coexistence of several paths.

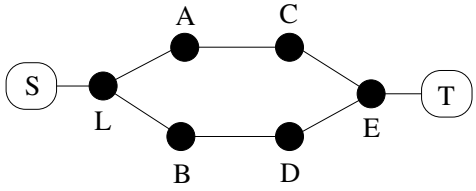

Fig. 1. A multipath route between S and T, created by a load balancer L.

Our work provides, for the first time, a tool capable of systematically finding the entire set of load-balanced paths that probes can follow in the presence of per-flow load balancing. We make a number of contributions:

(1) A formal model of multipath route discovery. Sec. II presents a graph-based model of the problem of multipath route discovery, which we use to analyze the probability that a discovery algorithm will find all nodes and links that probes can encounter along a multipath route. We provide a rigorous derivation of significance levels for finding an individual node's successor nodes, go beyond significance levels to the more important failure probabilities and how they may be controlled, and extend statistical node guarantees to path level guarantees.

(2) The Multipath Detection Algorithm (MDA). Sec. III provides an extension to Paris traceroute called the MDA, a stochastic probing algorithm that adapts the number of probes to send on a hop-by-hop basis in order to enumerate all reachable interfaces at each hop. The number of probes required by the MDA is a function of a tunable parameter, which is an upper bound on the probability of failing to discover the *entire* explorable multipath route, or *multipath*. Upon completion, the MDA yields a level of confidence in its result. The MDA also has mechanisms to deal with unresponsive routers and for identifying per-packet load balancers and routing changes.

(3) An experimental evaluation of the trade-off between strong guarantees and low probing overhead. Our statistical guarantees are based on very conservative, worst case assumptions, such as a load balancer at every hop. We perform experiments to explore the trade-off between overhead and actual success rates of MDA in a real world environment. Sec. IV describes experiments and Sec. V reports findings showing that even a 50% success probability bound is sufficient to find all the discoverable routes for more than 90% of multipath routes in our traces. On the other hand we find that the classic traceroute practice of sending three probes per hop falls well short of what is needed to obtain even modest confidence of finding the entire multipath.

In so doing, we correct and significantly extend a prelimi-

nary workshop paper that described an initial version of the MDA [3], as described in Sec. VI.

Our work puts traceroute on a solid analytic footing for the first time, showing how rigorous statistical guarantees can be engineered, and clearly demonstrating that standard traceroute practices are inadequate. By providing the MDA, an adaptive probing algorithm that regulates the number of probes according to the interfaces discovered at each hop, we give traceroute users more control and information to decide on the probing overhead they are willing to incur and the completeness of the multipath routes that will result.

II. Modeling Route Discovery

This section provides a formal graph-based model for route discovery, and analyzes its statistical properties. The model is based upon certain assumptions about ideal network behavior. Sec. III develops this into an algorithm that takes into account the constraints of conducting traceroute style route probing in a real-world network.

A. Formal Model

We model a multipath route from a source IP address s to a destination IP address d, $d \neq s$, as a directed graph $G = (V, E)$. V is a set of vertices, or nodes, consisting of s, d and the IP addresses of the ingress interfaces of the routers along each of the paths from s to d. E is a set of edges, where an edge $(v_1, v_2), v_1, v_2 \in V$, exists if and only if v_2 is encountered immediately following v_1 on at least one path from s to d. We assume about G: that the in-degree of s and the out-degree of d are both zero; and that for any node $v \in V \setminus \{s, d\}$ there exists at least one path from s to v and at least one path in G from v to d. G may contain loops.

In the above, we assumed that: **(1)** No routing changes during the discovery process.

G is unknown to the experimenter. The goal of our multipath route tracing is to discover G. We consider discovery to be a success only if all elements of V and E have been found (this analysis does not treat partial discovery).

We model the discovery process as the incremental construction of a graph $\hat{G} = (\hat{V}, \hat{E})$ starting with the initial graph $\hat{G}_0 = (\{s\}, \emptyset)$. Discovery is guided by a set of nodes on the frontier of exploration, which we call the *open set* or Ω. Initially, $\Omega = \{s\}$, and nodes are added to and removed from Ω as described below. Discovery is successful if, upon reaching the stopping condition $\Omega = \emptyset$, $\hat{G} = G$.

Discovery proceeds by choosing a node from the open set, $v \in \Omega$, it does not matter which one, and employing a per-node discovery process described below to attempt to enumerate its set of *successor nodes*, $\sigma(v) = \{v' \in V : (v, v') \in E\}$. This returns a nonempty set $C_v \subseteq \sigma(v)$ of discovered successor nodes. For each discovered successor node $c \in C_v$, if it is new then we add it to the list of nodes in our discovered graph: $c \notin \hat{V} \Rightarrow \hat{V} \leftarrow \hat{V} \cup \{c\}$. If it is new and it is not the destination then we add it to the open set as well: $(c \notin \hat{V} \wedge c \neq d) \Rightarrow \Omega \leftarrow \Omega \cup \{c\}$. Whether or not c is new, since we only perform the per-node discovery process once per node, the edge (v, c) is necessarily new, so we add it to $\hat{E}$: $\hat{E} \leftarrow \hat{E} \cup \{(v, c)\}$. Having done this for each successor node $c \in C_v$, we remove node v from the open set: $\Omega \leftarrow \Omega \setminus \{v\}$. Discovery terminates when the open set is empty: $\Omega = \emptyset$.

As mentioned above, per-node discovery consists in attempting to enumerate the successor nodes $\sigma(v)$ for a node v. This is done through a series of experiments. Each trial involves a probe packet that we have determined will pass through v, and sending it one hop beyond v to see what is returned. We model the result x_i of each trial i as a sample from a random variable $X_{v,i}$ that is uniformly distributed across the successor nodes $c \in \sigma(v)$. The random variables $X_{v,i}$ are independent and identically distributed for all i.

In the above, we have exploited the following additional assumptions about the network: **(2)** There is no per-packet load balancing. (As a result, we can manipulate a probe packet's flow identifier to cause it to pass through a chosen node.) **(3)** Load balancing is uniform-at-random across successor nodes. **(4)** All probes receive a response. **(5)** The effect of sending one probe packet has no bearing on the result of any subsequent probe. In particular, load balancers act independently.

Per-node discovery terminates on the basis of a stopping rule that caps the number of trials that can be conducted at a node. This number depends upon the number k of successor nodes that have been discovered so far. For each possible value $k = 1, 2, \ldots$ (theoretically without bound, but under current network configuration practice bounded by $k = 16$), trials stop when $i = n_k$ where n_k is the integer-valued *stopping point* for k. Note that the stopping point cannot be smaller than the number of successor nodes already discovered: $n_k \geqslant k$. For self consistency it makes sense to assume $n_{k+1} \geqslant n_k$.

Based upon the model above it is easy to show that: route discovery is guaranteed to terminate; upon termination there will be at least one path in the graph $\hat{G}$ from the source s to the destination d; for any node $v \in \hat{V} \setminus \{s, d\}$ there exists at least one path from s to v and at least one path from v to d.

B. Success and Failure Probabilities

For any given fixed multipath route or graph, we can define the *discovery failure probability*, namely the probability $\Pr(\hat{G} \neq G)$ that at least one path from source to destination has been missed by the discovery algorithm. Naturally, discovery *success* corresponds to all paths being found. This section gives expressions for the failure probabilities both for full-graph and single-node discovery. These allow algorithm performance to be explored over different topologies, and conversely, it enables the design of parameter settings to guarantee a given target performance for any topology of particular interest, such as the very common ones.

Consider the failure probability β_{all} for graph discovery. The only parameter of a node v_i which influences its failure probability β_{K_i} is the number $K_i = |\sigma(v_i)|$ of its successor nodes. Since the algorithm works independently at each node, the graph discovery success probability is simply the product

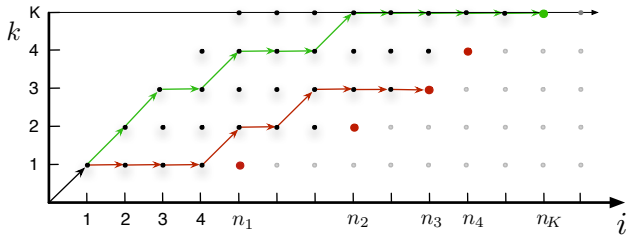

Fig. 2. State space for a node with K successor nodes. State (i,k) means k nodes discovered after i probes. Two algorithm paths are given: successful discovery stopping at (n_K, K), and discovery failure stopping at $(n_3, 3)$.

of success probabilities over all nodes, so that

$$\beta_{\text{all}} = 1 - \prod_i (1 - \beta_{K_i}). \tag{1}$$

We now consider the failure probability β_K for a single node (dropping the index i). Failure will occur whenever the algorithm concludes that there are less than K successors:

$$\begin{aligned} \beta_1 &= 0 \\ \beta_K &= \sum_{k=1}^{K-1} t_K^k, \quad K > 1 \end{aligned} \tag{2}$$

where t_K^k is the *stopping probability at level* k, the probability that the algorithm terminates having discovered k of the K successors. Since clearly $\sum_{k=1}^{K} t_K^k = 1$, the success probability is simply $t_K^K = 1 - \beta_K$. The stopping probabilities are a function of K and the set of stopping points $\{n_k\}$ only.

To better visualize the operation of the node discovery algorithm, Fig. 2 gives its state space, where state (i,k) corresponds to k successor nodes being known after i trials. Beginning at state $(0,0)$, paths move to the right with each probe, and upward as successor nodes are discovered, until reaching one of the K stopping states (n_k, k), $i \leqslant k \leqslant K$. Discovery success corresponds to paths which terminate at the state (n_K, K), and discovery failure to any of the other stopping states.

The probabilities governing state transitions are as follows. If $0 \leqslant k \leqslant K$ successor nodes have been discovered so far, then the probability that a new one will be found on the next attempt is just $q_k = (K-k)/K$, and $p_k = 1 - q_k = k/K$ is the probability of remaining at k. These transition probabilities are independent of the trial i, with the exception that transitions out of stopping states are of course disallowed, and the transition from $(0,0)$ to $(1,1)$ has probability one.

We can now calculate the stopping probabilities. One method is to first initialize the 'visitation' probabilities over the state space as $\Pr((0,0)) = 1$ and otherwise zero, and then, beginning at $(0,0)$, to use the transition probabilities to recursively calculate them over all states. The visitation probabilities of the termination states are precisely the desired stopping probabilities. It is instructive however to explicitly enumerate the possible paths to each termination state.

There is only one path from $(1,1)$ to $(n_1, 1)$, and so

$$t_K^1 = p_1^{n_1 - 1}.$$

To reach $(n_2, 2)$ from $(1,1)$, we first make j_1 'horizontal' transitions at level $k = 1$, then discover a second successor node, then make the remaining transitions at $k = 2$:

$$t_K^2 = \sum_{j_1=0}^{n_1-2} p_1^{j_1} q_1 \, p_2^{n_2-2-j_1}.$$

For $(n_3, 3)$, we must also keep track of the number of transitions j_2 at level $k = 2$ (see the example path in Fig. 2 with $(j_1, j_2) = (3,1)$), yielding

$$t_K^k = \sum_{j_1=0}^{n_1-2} \sum_{j_2=0}^{n_2-2-j_1} p_1^{j_1} q_1 \, p_2^{j_2} q_2 \, p_3^{n_3-3-j_1-j_2}.$$

It is easy to see that the general form is

$$t_K^k = \sum_{j_1=0}^{n_1-2} \sum_{j_2=0}^{n_2-2-j_1} \cdots \sum_{j_{k-1}=0}^{n_{k-1}-2-J_{k-2}} \Big(\prod_{i=1}^{k-1} p_i^{j_i} q_i \Big) \, p_k^{n_k-k-J_{k-1}} \tag{3}$$

where $J_i = \sum_{k=1}^{i} j_k$.

The stopping probabilities t_K^k are unconditional, based on starting the algorithm at state $(0,0)$. However, it is natural to ask what they would become, conditional on j successors having already been discovered. This is easily calculated as:

$$\begin{aligned} t_{K,j}^k &= \Pr(\text{stop at level } k \mid \text{didn't stop at levels below } j) \\ &= \frac{\Pr(\text{stop at level } k)}{\Pr(\text{didn't stop at levels below } j)} = \frac{t_K^k}{\sum_{l=j}^{K} t_K^l} \end{aligned} \tag{4}$$

for $k \geqslant j$ and zero otherwise. Note that $t_{K,1}^k = t_K^k$ since finding at least one interface is certain. The corresponding conditional failure probability, $1 \leqslant j \leqslant K$, is

$$\begin{aligned} \beta_{K,K} &= 0 \\ \beta_{K,j} &= \sum_{k=1}^{K-1} t_{K,j}^k = \sum_{k=j}^{K-1} t_{K,j}^k = \frac{\sum_{k=j}^{K-1} t_K^k}{\sum_{l=j}^{K} t_K^l}, \; j < K, \end{aligned} \tag{5}$$

and naturally $\beta_{K,1} = \beta_K$.

C. Significance Levels

Since the failure probability is a function of the actual multipath graph, which of course is unknown to any discovery algorithm, it unfortunately cannot be calculated by the algorithm itself. However, it is clearly important that the algorithm attempt to control its error, by estimating or bounding its failure probability based on its own observations. A natural framework for this is that of hypothesis testing.

We first consider the case of node discovery. Assume that there are K successor nodes, of which k, $1 \leqslant k \leqslant K$, have been discovered so far. The algorithm's role is to give up the search for new ones only if it is very unlikely that there are more it has failed to find. This corresponds to controlling the significance level α_k of a test for additional interfaces. Namely, α_k gives the probability of stopping at the state (n_k, k) and thereby failing to find new successors (a type-I error), given that k have already been found, under the null hypothesis that there are in fact $K' > k$ of them.

The above hypothesis is *composite*, meaning it is a collection $\{H^{\emptyset}_{K'}, K' > k\}$ of separate *simple* null hypotheses, such as the hypothesis $H^{\emptyset}_{K'}$ that there are precisely K' successors. The significance level of a composite null hypothesis is the largest of the significance levels of each of its simple components [14]. Now let $p_{k,K'} = k/K'$ be the transition probability to stay at level k on the next trial based on $H^{\emptyset}_{K'}$. Because the probability of remaining within level k and moving toward stopping at (n_k, k) is controlled by $p_{k,K'}$, which is larger for smaller K', the largest simple significance level, and therefore α_k for the composite hypothesis itself, is equal to that of $H^{\emptyset}_{K'}$ with $K' = k+1$. By stopping at (n_k, k) we are rejecting $H^{\emptyset}_{K'}$ and accepting the alternate hypothesis that $K' = k$. This is a simple hypothesis which has in fact optimal power: the probability of a type-II error is zero, since if $k = K' = K$, it is not possible to find more successors.

We can now evaluate α_k. Under $H^{\emptyset}_{k+1}$, from (4) the conditional probability of stopping at level k is just

$$\alpha_k = t^k_{k+1,k} = \frac{t^k_{k+1}}{t^k_{k+1} + t^{k+1}_{k+1}}, \quad 1 \leqslant k \leqslant K. \tag{6}$$

In the special case where $k = K-1$ successors have been found, the algorithm uses $H^{\emptyset}_{K'}$ with $K' = K$, and therefore $\alpha_{K-1} = t^{K-1}_{K,K-1} = \beta_{K,K-1}$ from (5). That is, the algorithm's own bound on the probability of missing successors, α_k, is actually equal to the objective view of an oracle who knows the true conditional probability of failure, given that k successors have been discovered already. If $k = K$ then $\alpha_K > \beta_{K,K} = 0$ and the algorithm is conservative. However, if $k < K-1$, simple counter-examples can readily be found to show that the algorithm can underestimate the true error. For example take $K = 3$ and $k = 1$. If $(n_1, n_2, n_3) = (2, 30, 30)$ then $\alpha_1 > \beta_{3,1}$, but if $(n_1, n_2, n_3) = (2, 3, 4)$ we find $\alpha_1 < \beta_{3,1}$.

D. Bounding Failure Probabilities

Significance levels provide a rigorous way to evaluate the confidence one has in the decision to stop and declare that all interfaces have been found. However, we have just seen that $\alpha_k < \beta_{K,k}$ does not necessarily hold, which is clearly undesirable as it implies that the actual error probability ($\beta_{K,k}$) may be larger than the one believed by the algorithm and reported to the user (α_k). However, the above analysis was for a stopping set $\{n_k\}$ which was arbitrarily chosen. We now ask the question, can we show that $\alpha_k \geqslant \beta_{K,k}$ for all k provided the algorithm selects the $\{n_k\}$ appropriately? It is again easy to prove by counter-example ($K = 3$, $k = 1$, $\alpha = 0.16$, $(n_1, n_2, n_3) = (4, 7, 7)$) that this does not hold if we set $\alpha_k = \alpha$ for all k, the simplest choice. We now give a more nuanced choice that aims to achieve something different and more useful instead: a universal bound on the failure probability.

As described above, (6) was derived under $H^{\emptyset}_{K'}$ with $K' = k+1$ because that gave the largest α_k. Therefore, for $K' = K$

$$\alpha_k \geqslant t^k_{K,k} \geqslant t^k_K$$

since conditioning increases probability. Using this in the numerator of (5) we obtain the bound

$$\beta_{K,k} \leqslant \beta_{K,1} = \sum_{l=1}^{K-1} t^l_K \leqslant \sum_{l=1}^{K-1} \alpha_l. \tag{7}$$

where the first inequality follows from the fact that $f(x) = x/(a+x)$, a, $x > 0$, is increasing (set $a = t^K_K$). If for example all the significance levels were equal, then this bound grows linearly with K, which is not useful when K is unknown! To obtain a bound that is independent of K and so can be used by the algorithm, we set

$$\alpha_k = \alpha_1 r^{k-1} \tag{8}$$

where $0 < r < 1$, which geometrically reduces the significance levels as k increases. Continuing on from (7):

$$\beta_{K,k} \leqslant \sum_{l=1}^{K-1} \alpha_l < \frac{\alpha_1}{1-r}\ . \tag{9}$$

To adhere to some target failure probability β^*, we can for example fix a convenient α_1, and then select $r = 1 - \alpha_1/\beta^*$, or alternatively select r and set $\alpha_1 = (1-r)\beta^*$. Either way, the resulting *target* significance levels $\{\alpha_k\}$ will guarantee that $\beta_{K,k} < \beta^*$ for all k, regardless of the true (unknown) K.

We do not expect the above bound to be tight. For example whilst $\beta^* > \alpha_1$, we already know that in the case of nodes with $K = 2$ the algorithm will achieve the smaller failure probability of $\beta_{2,1} = \beta_2 = \alpha_1 = t^1_2 = 2^{-(n_1-1)}$. It is possible in fact to choose the $\{\alpha_k\}$ larger, however it remains an open question whether there exist schemes which can, for example, provide $\alpha_k \geqslant \beta_{K,k}$ for all k when $K > 2$. A related open question is how to optimize performance in terms of minimizing the number of trials for a fixed failure probability.

We can define a notion of graph level bounds on failure probability following the example of (1):

$$\beta^*_{\text{all}} = 1 - \prod_i (1 - \beta^*_{k_i}) \tag{10}$$

provided we know the total number of nodes. If we don't know but can bound this number, then the RHS of (10) becomes a bound on the graph level failure probability.

E. Summary of Statistical Guarantees

We summarize the above 'statistical guarantees' to path discovery, and compare them to those of our workshop paper [3].

Node significance levels. In the workshop paper [3],

$$\alpha_{\text{node}} = (k+1)^{-n_k} \sum_{j=0}^{k} \binom{k+1}{j} j^{n_k} (-1)^{k-j} \tag{11}$$

was used, an approximate significance level assuming $K' = k+1$ successors which ignores the stopping points below level k (i.e., is biased), and includes events at levels $k < K'$ rather than at $k = K' - 1$ only (is not conditional). In contrast, we now use (6), which is both conditional and exact.

Failure probabilities, and their control. In [3], the quality of a stopping decision was expressed in terms of α_{node} only, with no discussion of failure probabilities. We have introduced the conditional failure probabilities $\{\beta_{K,k}\}$ as the appropriate measure when the topology is known, and also shown how to bound them 'universally', that is when K is unknown.

Options for node statistical guarantees. The $\{n_k\}$ determine algorithm performance, and can be selected in one of two ways: significance levels, or failure probabilities. In the former case, a single significance level $\alpha_k = \alpha_{\text{node}}$ is used for each node and there is no ability to control or even report failure probabilities. In the latter case, actual failure probabilities can be specified and achieved to ensure performance for a given target K, or alternatively, a single parameter β^* can be used to bound all failure probabilities for arbitrary unknown K, which is more comprehensive but more expensive. When controlling failure probabilities, the significance levels are still meaningful, known, and can also be reported. However, they are no longer central, but merely a means to ensure the failure targets (and they now vary with k).

Graph level error reporting and control. In [3], stopping decisions only had guarantees at the node level. We have introduced a meaningful means of reporting a graph level guarantee via (1) when the topology is known, and via (10) when it is not (provided we bound the number of nodes).

III. Multipath Detection Algorithm

The previous section described an idealized model we used to build our path discovery algorithm and to understand its operation statistically. This section presents the MDA, a traceroute-like probing algorithm inspired by the formal model, capable of building the full set of discoverable multi-path routes between two internet hosts.

A. From Model to Real World

Although the network abstractions of the formal model are well motivated, they do not always hold in real networks at all nodes. Here we introduce the differences, and the adaptations we propose to circumvent them or to limit their impact.

Per-hop enumeration. The model allows the algorithm to tackle nodes on the open list in any order. However, traceroute works hop by hop, and prints real time output as it discovers new interfaces. The MDA hews to this mode of operation.

Fig. 3 illustrates the technique and introduces some notation. Let R_h be the set of all interfaces reachable at hop h when tracing towards a given destination. Suppose we have already built $\hat{R}_{h-1}$, our estimate for the set of interfaces at the previous hop. To build $\hat{R}_h$, we enumerate the successors c of each interface $v \in \hat{R}_{h-1}$. If v is the interface of a load balancer, there will be multiple successors. We call the set of successors of v the *nexthops* of v and denote this as C_v. The union of nexthops for R_{h-1} is R_h. Below, we describe our method for enumerating nexthops with statistical guarantees.

The MDA proceeds hop by hop, and explores the IP-level graph by enumerating the nexthop interfaces of each interface

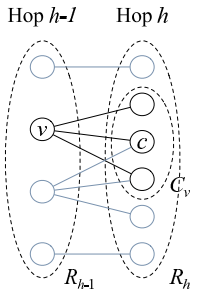

Fig. 3. Per-hop enumeration.

discovered, until probing reaches the destination along all paths. During the exploration, it labels each interface with its representative flows, whenever possible. This hop-by-hop exploration is equivalent to the one used in the model, the only differences being that we specify a certain order in which to visit the nodes and we also may search for nexthops from a node more than once if that node is found at more than one hop count (in each case, we will be visiting that node using a different set of flow identifiers). We can easily derive the graph from the flow-labeled nodes, by creating a link (v, c) for every pair of interfaces v and c that are at consecutive hop counts and that share a common flow identifier. In so doing, we go beyond the basic goal of discovering the graph, to labeling the nodes with flow information, as well as load balancing type.

Node access. Another difference with the model is the way we access a node whose successors we wish to enumerate. With traceroute-like measurements, this requires an additional effort in order to ensure that the packets with a given flow identifier actually go through the target node. Let us illustrate this problem with Fig. 3. If one wishes to discover the successors of v, one must first generate a sufficient number of flow identifiers that cause probes to reach v at hop $h-1$. The only way to do so is by trial and error, generating flow identifiers with port numbers chosen at random. In so doing, we discover flow identifiers that cause probes to reach other nodes in R_{h-1}, which may or may not be useful.

Non-responses. The failure of routers to respond to probe packets is another network characteristic our model ignores. Unlike classic traceroute probing, which does not try to address this problem, in our case it is crucial to continue to obtain responses since the stopping points $\{n_k\}$ have been calibrated to achieve particular statistical guarantees. There are various causes for non-responses. Some, such as anonymous routers [19] have no countermeasure. The MDA handles other causes of non-responses, such as ICMP rate limiting and packet loss, with a timeout-based retransmission mechanism.

Per-packet or non-uniform load balancing. Our model assumes per-flow load balancing, performed uniformly across successor nodes. Although per-packet and non-uniform load balancing are considerably less common, they also exist in the internet [2], Under per-packet load balancing, we have as yet no way to control the path that packets take. The algorithm must therefore identify cases of per-packet load balancing where they exist, which requires additional probing overhead.

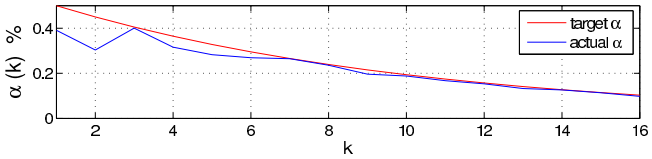

Fig. 4. Significance levels corresponding to $\{n_k\}$ from Table I.

Under non-uniform load balancers, the MDA may miss links in the paths that packets take less often.

Node revisiting. Under our model, we do not attempt to enumerate the successors of any given node more than once. However, given the hop-by-hop approach, and the fact that the same node might be reached at different hop counts with different flow identifiers, the MDA attempts to enumerate the successors to every node encountered at a given hop count, whether they have been visited before or not. The consequence for (1) is that a second attempt at finding its successors improves the chance of success, leading to

$$\beta_{\text{all}} \leqslant 1 - \prod_i (1 - \beta_{K_i}). \tag{12}$$

In this case, global effects boost the efforts of local action.

Topology changes. Our model also supposes that routes remain stable during measurement. However, the probing of the entire multipath to a destination can take dozens of seconds, sometimes more, increasing the risk of topology change.

B. Implementing Statistical Guarantees

Since the algorithm's operation (both now and in [3]) is based on the stopping points $\{n_k\}$, at the node level implementing the statistical guarantees reduces simply to implementing different choices for how these are calculated. Given the input parameter controlling the statistical guarantee, α_{node} or β^*, the $\{n_k\}$ can easily be precalculated in advance of sending probes. Note that because the $\{n_k\}$ take discrete values, in practice instead of the target significance levels $\{\alpha_k\}$ we end up with the set $\{\underline{\alpha}_k\}$ of *actual* significance levels. These are 'safer,' i.e., the largest possible where $\underline{\alpha}_k \leq \alpha_k$.

Whereas in [3], the $\{n_k\}$ were chosen to ensure that $\underline{\alpha}_k \leq \alpha_{\text{node}}$ for each k, here we use (8) to first select a set $\{\alpha_k\}$ according to a choice of β^*, and then select the smallest $\{n_k\}$ which can achieve them. This is straightforward to do recursively since $\underline{\alpha}_k$ is a function of the $\{n_j, j = 1, \ldots k+1\}$ only. Table I gives an example based on $\beta^* = 0.05$ (with $r = 0.9$), generated from target significance levels given in Fig. 4. Note that the largest conditional failure probability over all k for each $K \leq 16$ using these $\{n_k\}$ is 0.0041, which is below $\beta^* = 0.05$ as claimed, and well below, as expected.

k	1	2	3	4	5	6	7	8	9
n_k	9	17	24	33	42	51	60	70	81
k	10	11	12	13	14	15	16		
n_k	91	102	113	125	136	148	161		

TABLE I
NUMBER OF PROBES n_k NEEDED TO BOUND ALL (CONDITIONAL) FAILURE PROBABILITIES FOR ANY K AT A NODE BY $\beta^* = 0.05$.

Finally, when using failure control via β^* the implementation now takes an option to control this via a graph level parameter β^*_{all}, calculated as described in Sec. II-D. We used a bound of 30 for the node count of the multipath route. Based on real traces [2], this holds for 99.8% of multipath routes.

IV. MEASUREMENT METHODOLOGY

The preceding section describes how to determine the number of probes the MDA must send at each hop in order to bound its failure probability. This bound is based on minimal assumptions about the actual topology of a multipath route. Since the network might diverge from these assumptions, we ask how tight the bound is in reality. This section describes the experiments we designed to obtain answers to this and other questions about the MDA's performance.

To determine if the MDA has failed for a given route requires a 'ground truth'. This is a multipath route that we know, to a high degree of certainty, represents everything that traceroute could possibly discover when probing from the route's source to its destination. Ideally, we would obtain this ground truth from network operators themselves. However, it is impossible to obtain all such topologies, and even then there would be no guarantee that some exotic router behavior would go unreported. We set a different standard for failure: the MDA only fails if additional probing could have yielded more information about a multipath route. We do not consider it a failure, for instance, if the MDA leaves a link unrevealed that is not actively being used for routing. Similarly, we do not consider it a failure if the MDA leaves unrevealed a series of links that are hidden by an MPLS tunnel. Our concern here is with the capabilities of traceroute-style probing. Therefore, we obtain a 'baseline' topology using the MDA itself, by setting an extremely strict bound on the failure probability (we used $\beta^*_{\text{all}} = 2^{-7}$) that makes it very unlikely that there exist undiscovered paths. Although not perfect, this approach provides well defined performance calibration for failure probabilities above the baseline threshold.

Having established the ground truth by sending the very high numbers of packets required by a very strict bound, we want to compare the outcomes for more reasonable bounds. We can do so without reprobing. Our initial trace serves as a reference, from which we emulate the MDA outcome for a smaller number of probes by sub-sampling. We also compare the outcome for classic traceroute's standard of three probes per hop by replaying just the first three probes for each hop in the reference trace. In our experiments, we emulated the outcomes for failure probability bounds of $\beta^*_{\text{all}} = 0.01, 0.05, 0.25$, and 0.50, as well as for the classic three probes per hop.

We launch all measurements to the same destination close in time, to minimize the chances of routing changes between measurements. We use UDP probes for all our experiments and set the maximum TTL to 36. We set the retransmission timeout to 2 seconds and the interval between probes to 50 ms and stop tracing a destination after 3 consecutive unresponsive hops, to accelerate the measurements.

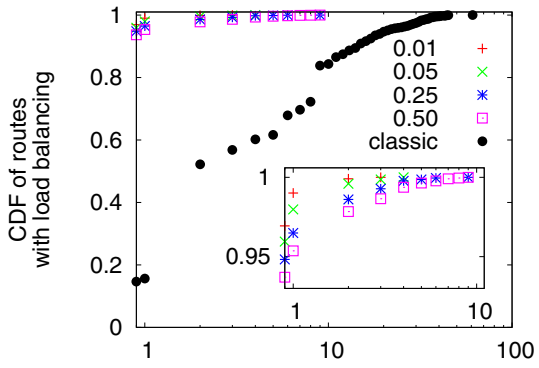

Fig. 5. Links undiscovered by the MDA (log scale), with various bounds β^*_{all} (zoom in the bottom right corner), and by classic traceroute.

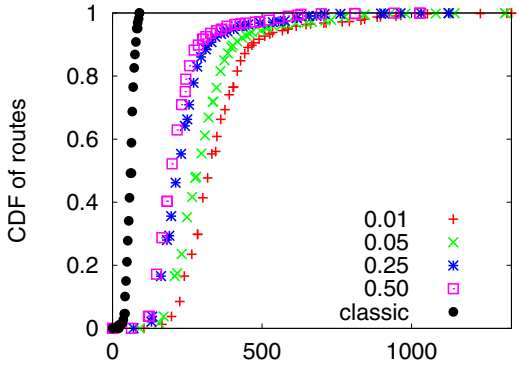

Fig. 6. Packet overhead for the MDA, with various bounds β^*_{all}, and for classic traceroute.

V. Evaluation

This section studies the MDA's performance on 5,000 routes, located between a source at UPMC Paris Universitas and 5,000 randomly-selected IP addresses that respond to pings. Since they are limited to one source, these experiments do not provide universal guidelines for trading-off strict failure bounds against overhead. Load balancing affects each location to a different degree. In a previous measurement study [2], the portion of source-destination pairs affected by per-flow load balancing varied between 23% and 80%, depending upon which of 15 sources in the US and Europe we used. We chose the Paris source because its portion lies in the middle (varying between 38% and 44% from round to round). We repeated each round of measurements four times during a three-week period. The results are equivalent across rounds; hence we report the results for one round.

These experiments provide a procedure that others can easily follow to calibrate their particular usage of the MDA. In addition to experiencing their own location-dependent conditions, each user will have their own particular tolerance for overhead, and intolerance to multipath route discovery failure.

A. Links Undiscovered

Fig. 5 gives us a rich picture of multipath route discovery failure by looking at the extent of failure when it occurs. The graph is a CDF of the number of links (shown in log scale) that the discovery algorithm fails to find. There is one curve for each failure probability bound β^*_{all} and one curve for the classic traceroute approach of sending three probes per hop. Of the 5,000 routes traced, only 2,205 (44%) provide opportunities for failure because they traverse a load balancer, and so Fig. 5 is based only on those routes.

Here we see the *clear inadequacy of the classic three-probes-per-hop approach*: it misses at least 1 link for 84% of the multipath routes, more than 10 links in 25% of cases, and in nine extreme cases, misses more than 40 links! In contrast, adaptive probing with the MDA provides routes that are close to complete even when the failure bounds are not strict. For instance, with $\beta^*_{\text{all}} = 0.50$ it finds all links 94% of the time.

This leads us to our second conclusion, which is that the *failure bounds are quite loose*. There is a high rate of complete success, which can be explained by the conservative assumptions that go into calculating the bounds: that a multipath consists of 30 nodes, and that every node represents an opportunity for failure. In our experience [2], there tend to be just one or two divergence points on paths that encounter load balancing, and so just one or two true opportunities for failure.

Even when there is failure, the MDA tends to miss only a small number of links. This can be explained by the fact that most instances of load balancing split packets across just two or three paths and load-balanced paths tend to have lengths of just two or three hops [2], so there tend to be few links to miss. The advantage of the MDA's adaptive approach to probing becomes abundantly clear in those rare cases where there are many load-balanced links to miss. By continuing to send more and more packets to a given hop as more an more interfaces are discovered, the MDA keeps the number of undiscovered links low.

The data show one case in which the bound is not tight: for $\beta^*_{\text{all}} = 0.01$, the MDA succeeded completely in only 96.9% of cases, instead of $100 \cdot (1 - 0.01)\% = 99\%$ or better. We believe this apparent contradiction to the statistical guarantee to be the result of a very small sample size, allowing statistical variation about the average value of β^*_{all} to become visible.

B. Packet Overhead

Classic traceroute sends a fixed three probes per hop, which helps cap overhead. With adaptive probing, the number of probes sent is less predictable. It depends on the numbers of paths, load balancers, and non-responses, not only path length. Fig. 6 gives the CDF of the total number of packets sent to trace each route, including those without load balancing. This includes the probes sent during the interface enumeration phase, the classification phase, and all necessary retransmissions. We plot these results for the MDA with various bounds and for classic traceroute. As expected, classic traceroute sends a relatively low number of packets, with a mean of 61 (for a route with 20 hops) and a maximum of 90 packets. For the MDA, even with an unstrict bound ($\beta^*_{\text{all}} = 0.50$), the mean

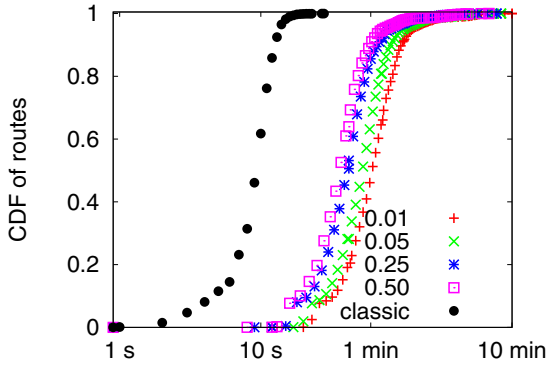

Fig. 7. Time overhead for the MDA (log scale), with various bounds β^*_{all}, and for classic traceroute.

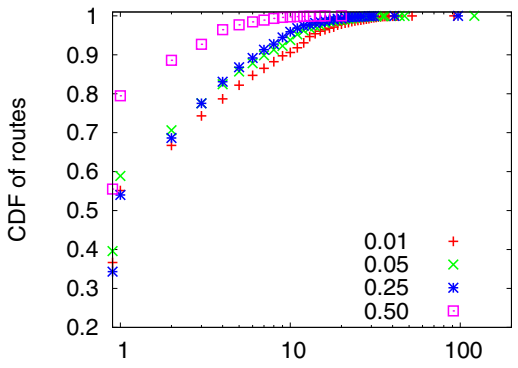

Fig. 8. Number of retransmissions by the MDA (log scale), with various bounds β^*_{all}.

nearly quadruples to 216 packets and the maximum is 1,027. With a relatively strict bound ($\beta^*_{\text{all}} = 0.01$) the mean becomes 348 and the maximum 1,334. However, this maximum number is rare. The MDA traces 90% of routes with fewer than 468 packets for $\beta^*_{\text{all}} = 0.01$ (290 for $\beta^*_{\text{all}} = 0.50$).

We conclude that *traceroute overhead increases significantly with the introduction of the MDA*. This is not to say, though, that individual traceroutes present a congestion concern for traditional wired links. A back-of-the-envelope calculation for $\beta^*_{\text{all}} = 0.05$ indicates bandwidths of 880 bps for probes and 1,288 bps for replies, which is minuscule compared to today's link capacities. However, peaks of probe emissions might cause problems because of rate limiting by routers. We discuss this concern further in Sec. V-D. Scenarios might also arise in which overhead becomes a concern because of either unusually low link capacities or unusually high numbers of simultaneous traceroutes.

C. Time Overhead

A long route trace duration increases the risk that a routing change will cause an inaccurate inference. Also, long traces are a problem for individual users who use traceroute for troubleshooting and expect results in real time.

Fig. 7 plots the CDF of tracing time (in log scale) for all multipath routes, for the MDA with various failure bounds and for classic traceroute. Classic traceroute takes less than 10s for half of the routes and 23s in the worst case (a forwarding loop). It terminates quickly because the time depends only on the RTT to each hop, the path length, and the number of non-responsive hops (requiring a timeout expiry before sending the next probe). The MDA takes more time to complete. For $\beta^*_{\text{all}} = 0.01$, it takes more than one minute to trace more than one half of the routes, and we encountered a maximum probing time of over ten minutes! Even with relaxed bounds, the MDA takes considerably longer than classic traceroute.

Delays can be shortened to a single RTT for classic traceroute by using an a priori maximum hop count and sending all probes in parallel. While the MDA does lend itself to such improvement through parallelism, its adaptive nature requires feedback from initial probes, or timeouts, to determine hop counts and flow identifiers for subsequent probes. As a result, some of these delays are compressible only if one is willing to incur large increases in probing overhead to anticipate a wide range of possible probing outcomes with high probability. Our results indicate that *further work is required on reducing delays in multipath route probing.*

D. Retransmissions

Retransmissions after a timeout are an important component of both overhead and delay for the MDA. Fig. 8 shows the CDF for the number of retransmissions. Classic traceroute does not retransmit, so it does not appear in the plot.

Retransmissions are fairly common, being triggered for 45% of routes when $\beta^*_{\text{all}} = 0.50$, and this portion increases as the bounds get stricter. The increase is doubtless due to larger numbers of probes resulting in greater ICMP rate limiting by routers. It is natural, therefore, to ask whether these retransmissions help.

To answer this question, we compared the failure rates for obtaining responses with and without retransmission. With retransmissions, of 106,162 hops probed, no reply was obtained from 14.3% of them. Disabling retransmissions, the portion of non-responses only increased a small amount, to 17.4%. This indicates that most non-responses are the result of anonymous routers [19], which drop all traceroute probes. The lesson we draw is that *retransmissions have a limited benefit.* In situations that are sensitive to probing or delay overhead, it may be best to avoid them, at a small cost in completeness.

VI. Related work

Traceroute is the basis of many projects in internet mapping [5], [8], [12], [16]. All of these studies aim to have internet maps as complete as possible by tracing from multiple sources to multiple destinations. There is also a non-peer-reviewed work that looks at the trade-off between completeness and low overhead when tracing from a single source to multiple destinations [6]. In contrast, our work addresses the completeness of the topology between a single source and a single destination, specifically under load balancing.

Although there has been considerable work on the design of efficient load balancers [7], [15], load balancing in the

internet is still under-documented. Huffaker et al. [9] mention it as a potential problem, and it is likely the cause of what Paxson [11] calls *route fluttering*. Recently, TCP Sidecar [13] proposed to use the IP "Record Route" option to trace through load balancers. The work on load-balanced paths is also close to prior work on path diversity [17], [18], as the MDA measures (at least partially) the path diversity natively provided by the internet.

This paper grows out of our earlier work on Paris traceroute [1]–[3]. The original Paris traceroute paper [1] introduced the idea, which we use here, of fixing the flow identifier for all probes of a path, in order to report precise routes under per-flow load balancing. We developed the MDA for a measurement paper that followed [2], in order to study the prevalence of load-balanced paths in the internet. That paper made use of the MDA as defined in a workshop paper [3] that provided the first algorithmic statement and analysis of the MDA. That paper presented the first version of the model described in Sec. II, which this paper completes in three ways: providing a more rigorous derivation of significance levels for finding an individual node's successor nodes, going beyond significance levels to the more important failure probabilities and how they may be controlled, and extending statistical guarantees from the node level to the path level. The MDA, as described in Sec. III, is updated from the workshop version in light of the above improvements. Its basic structure remains the same, but the number of probes sent can now be calibrated according to the new statistical criteria, driven by a new input parameter: a bound of the probability of failing to discover the *entire* multipath route. We have also added mechanisms to deal with unresponsive routers, and for identifying per-packet load balancers and routing changes.

VII. Conclusion

Our study represents a major advance for traceroute users. Instead of blindly using the default policy of sending three probes per hop, users will now be able to configure their probing algorithm with the knowledge of how to trade off the completeness of multipath routes against low probing overhead. Our model of multipath route discovery allows us to compute the number of probes to send to find all successors of a node with rigorous statistical guarantees. Our significance level analysis linked to a bounded failure probability provides tunable auto-calibration so the algorithm can provide meaningful measures of confidence, and guarantees, conditional on what it experiences. We proposed the Multipath Detection Algorithm (MDA), a traceroute-based probing algorithm that brings the idealized discovery algorithm from the model to reality. Our experiments showed that classic traceroute misses at least one link for more than 80% of multipath routes, whereas the MDA's adaptive probing, even with a maximum failure probability of 50%, misses links in only 6% of cases. This greater route knowledge comes at the cost of higher probing overhead. In extreme cases, the MDA may send more than a thousand probes to find all links of the multipath route when failure is bounded at 1%.

We see a number of possible extensions to the MDA. Better knowledge of the hashing functions used by per-flow load balancers [4] should make it possible to reduce probing overhead, with the possibility to revert to the MDA in cases where no predictable pattern can be identified with high certainty. There is considerable scope for improving the current (loose) failure bound, thereby saving probes, and both the significance level and failure analyses open up new avenues for trading off and optimizing performance against probe budget. Another open question is how to accurately trace multipath routes past per-packet load balancers.

Acknowledgments

The Paris traceroute tool was developed with financial support from the French CNRS, as part of its contribution to the European Commission-sponsored *OneLab* project. We are grateful to Xavier Cuvellier, Fabien Viger, Matthieu Latapy, and to Clémence Magnien for their contributions to the Paris traceroute project.

References

[1] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira. Avoiding traceroute anomalies with Paris traceroute. In *Proc. ACM SIGCOMM Internet Measurement Conference, IMC*, October 2006.

[2] B. Augustin, T. Friedman, and R. Teixeira. Measuring Load-balanced Paths in the Internet. In *Proc. ACM SIGCOMM Internet Measurement Conference, IMC*, October 2007.

[3] B. Augustin, T. Friedman, and R. Teixeira. Multipath Tracing with Paris Traceroute. In *Proc. IEEE Workshop on End-to-End Monitoring, E2EMON*, May 2007.

[4] Z. Cao, Z. Wang, and E. W. Zegura. Performance of hashing-based schemes for internet load balancing. In *Proc. IEEE Infocom*, 2000.

[5] Cooperative Association for Internet Data Analysis. Skitter. http://www.caida.org/tools/measurement/skitter/, January 2000.

[6] B. Donnet, P. Raoult, and T. Friedman. Efficient route tracing from a single source. *arXiv preprint cs.NI/0605133 v1*, May 2006.

[7] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proc. IEEE Infocom*, 2001.

[8] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *Proc. IEEE Infocom*, March 2000.

[9] B. Huffaker, D. Plummer, D. Moore, and k claffy. Topology discovery by active probing. In *Proc. Symposium on Applications and the Internet*, Jan. 2002.

[10] V. Jacobson. traceroute, February 1989.

[11] V. Paxson. End-to-end internet packet dynamics. *IEEE/ACM Trans. Networking*, 7(3):277–292, June 1999.

[12] Y. Shavitt and E. Shir. DIMES: Let the internet measure itself. *ACM SIGCOMM Computer Communication Review*, 35(5):71 – 74, October 2005.

[13] R. Sherwood and N. Spring. Touring the Internet in a TCP Sidecar. In *Proc. ACM SIGCOMM Internet Measurement Conference, IMC*, October 2006.

[14] S. Silvey. *Statistical Inference*. Chapman & Hall, 1975.

[15] S. Sinha, S. Kandula, and D. Katabi. Harnessing TCPs Burstiness using Flowlet Switching. In *Proc. SIGCOMM Workshop on Hot Topics in Networking, HotNets*, November 2004.

[16] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. In *Proc. ACM SIGCOMM*, August 2002.

[17] R. Teixeira, K. Marzullo, S. Savage, and G. M. Voelker. In Search of Path Diversity in ISP Networks. In *Proc. ACM SIGCOMM Internet Measurement Conference, IMC*, October 2003.

[18] X. Yang and D. Wetherall. Source Selectable Path Diversity via Routing Deflections. In *Proc. ACM SIGCOMM*, August 2006.

[19] B. Yao, R. Viswanathan, F. Chang, and D. Waddington. Topology inference in the presence of anonymous routers. In *Proc. IEEE Infocom*, April 2003.