Multipath tracing with Paris traceroute

Brice Augustin, Timur Friedman, Renata Teixeira

Multipath tracing with Paris traceroute

Brice Augustin, Timur Friedman, Renata Teixeira
Université Pierre et Marie Curie, Laboratoire LIP6–CNRS

***Abstract*—Traceroute is a tool to report the route packets take between two internet hosts. However, with the deployment of load balancing, there is no longer a single route to a destination, hence classic traceroute systematically misses some of these paths. In this paper, we specify an adaptive, stochastic probing algorithm, called the Multipath detection algorithm, to report all paths towards a destination. We have deployed this algorithm, probing from a single source towards multiple destinations. In our results, we have found instances of load balancing with as many as 16 interfaces per hop. The algorithm also allows us to count load balancing routers, identify their locations, and characterize them by type.**

Categories and Subject Descriptors: C.2.3 [Computer Communication Networks]: Network Operations

General Terms: Measurement.

Keywords: traceroute, load balancing, multipath.

I. Introduction

Traceroute [1] is used to learn the path between two machines in the internet. Uses range from the diagnosis of network problems to the assemblage of internet maps. Unfortunately, traceroute measurements can be inaccurate and incomplete when the measured route traverses a load balancing router, or *load balancer*.

Consider the example in Fig. 1, which we will use throughout the paper. $L1$, $L2$ and $L3$ are load balancers. The left side represents the actual network topology. Routers are represented as circles. We also number each of their interfaces. Probes are represented by black squares, either above the topology if they traverse $L2$, or below if they traverse $L3$. The right side is a possible classic traceroute outcome. This example illustrates the two problems with classic traceroute under load balancing. First, the discovered path is inaccurate since it reports a false link (the link ($L2_0$,C_0) does not exist in the real topology). Second, it is incomplete since traceroute missed half of the nodes and their respective links. As a result, one may diagnose an incorrect path or build an incomplete map of the network.

Two types of load balancers cause problems with traceroute. Per-flow load balancers ascribe each packet to a flow defined by the header five-tuple, and each flow to an outgoing interface. Per-packet load balancers assign packets to interfaces regardless of flow. Traceroute cannot control the effects of per-packet load balancing. What is perhaps surprising is that we observe the same effects with per-flow load balacing. This is because traceroute varies some of the packet header fields that are used to define a flow.

[1]1-4244-1289-7/07/$25.00 2007 IEEE

In prior work [2], we proposed a new traceroute implementation called *Paris traceroute*[1], which maintains a constant flow identifier in all the probes it sends, and hence solves the problem of inaccuracy under per-flow load balancing. This control of the probe content helps traceroute avoid a significant portion of the measurement artifacts (loops, cycles, and diamonds) caused by load balancers.

The prior work described how to trace the path that a single flow takes from source to destination. However, in light of this new understanding, traceroute should be capable of describing the paths that all possible flows take from source to destination. The most straightforward way of finding these paths would be to repeatedly trace routes, each time with a different random flow identifier. But at what cost in probing overhead, and when should probing stop? In the context of internet cartography applications that trace from multiple sources to multiple destinations, repeated traces through the same set of load balanced paths will eventually lead to a complete topology discovery. For instance, successive traceroutes towards various destinations, but all traversing the network in Fig. 1, would discover all interfaces and links.

Nevertheless, we place our work in the context of tracing from a single source to a single destination, which is the most common way people use traceroute. The fact of widespread load balancing calls into question the traditional role of this tool, which is to trace a single path from source to destination. We suggest a new goal for route tracing: to find the entire set of load-balanced paths between source and destination. In this work, we describe modifications to Paris traceroute to solve the problem of path incompleteness. We show that the classic traceroute practice of sending three probes per hop is inadequate to have even a moderate level of confidence that one has discovered even the most basic load balancing at a given hop. We propose a stochastic probing algorithm called the *Multipath detection algorithm*, which adapts the number of probes to send on a hop by hop basis, in order to enumerate all interfaces and links at each hop. Our findings show that we need to send at least 6 probes per hop to rule out load balancing with a reasonable degree of confidence, and up to 96 probes to discover the largest set of interfaces present in our traces. We evaluate our solution by performing measurements towards a set of randomly-selected destinations, and use those experiments to characterize load balancing as seen from a single source. Because of the small-scale nature of our experiments, we present those results as a proof of concept

[1]Paris traceroute is freely available for download at http://www.paris-traceroute.net/.

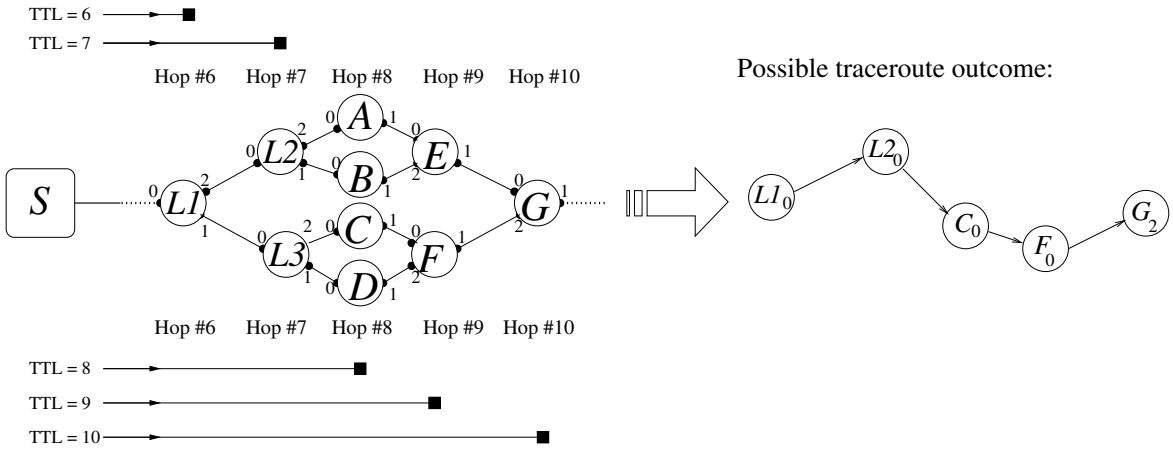


Fig. 1. Traceroute under load balancing

rather than a general study of load balancing in the internet.

This paper is structured as follows. Sec. II discusses related work. Sec. III describes the Multipath detection algorithm. Sec. IV describes our experimental setup. Sec. V characterizes load balancing and provides an evaluation of our probing algorithm. Sec. VI presents our conclusions and perspectives for future work.

II. Related work

Traceroute is the basis of many projects in internet measurements [3], [4], [5], [6], [7], [8]. They all use a classic traceroute probing algorithm for their experiments, which does not take into account the problems caused by load balancing. The problem of obtaining complete internet maps using traceroute is well represented in the litterature. The Rocketfuel work [9] tries to infer the topology of an Internet Service Provider network using multiple vantage points and a large set of destinations carefully selected to improve the network coverage. However, Teixeira et al. [10] reported that the infered topologies tend to comprise a significant number of false links. Lakhina et al. [11] showed that traceroute-like methods introduce biases in the measured topologies. Those papers tackle the completeness problem in the context of tracing from multiple sources to multiple destinations. Howerver, with load balancing, standard traceroute is not even able to discover a complete topology from a single source to a single destination.

Although there has been considerable work on the design of efficient load balancers [12], [13], [14], load balancing in the internet is still under-documented. Huffaker et al. [15] mention it as a potential problem, and it is likely the cause of what Paxson [16] calls *route fluttering*. Recently, TCP Sidecar [17] proposed to use the IP "Record Route" option to trace through load balancers. The use of Paris traceroute [2] showed that load balancing is the main cause of measurement artifacts like false diamonds, loops, and cycles in internet graphs. It also showed that Paris traceroute reports more precise paths by avoiding most of those anomalies. Our work is also close to prior work on path diversity [10], [18], as the Multipath detection algorithm measures (at least partially) the path diversity natively provided by the internet.

III. Enumerating paths

This section introduces the probing strategy we propose to trace the true paths to a single destination with traceroute, and discusses its key ideas. We first explain the desired features of a load-balancer-aware traceroute, then we detail our method to achieve this goal. An enhanced traceroute should find all the possible sequences of interfaces and links from source to destination. Formally, we define a *measured route* to be the vector $\mathbf{p}^k = (p_0^k, \ldots, p_{\ell_k}^k)$, listing all the interfaces, starting at the source, p_0^k, and going to the end of the trace at distance ℓ_k. $\mathbf{p}^k$ represents an individual measured route among the set of load-balanced routes to the destination. Let us call the set of all measured routes from source to destination P, and suppose there are $N = |P|$ such routes. Our goal is to discover all distinct measured routes $P = \{\mathbf{p}^1, \ldots \mathbf{p}^N\}$.

In addition to finding all measured routes, we also want to label each link in those routes that we know to be a true link. With per-packet load balancing, we do not know how to connect the interfaces found at one hop with the interfaces found at the next hop, but with per-flow load balancing we can link two consecutive interfaces found in a measured route and designate the pair as a true link. Therefore, we first need to classify the load balancing by type. We label each responding interface r in the paths with a Boolean π_r, where $\pi_r = 1$ if r is the interface of a per-packet load balancer. We then use a Boolean label, $t_i^k \in 0, 1$, where $t_r^k = 1$ means that the link (p_{i-1}^k, p_i^k) is a true link, and $t_i^k = 0$ means that we cannot tell whether link (p_{i-1}^k, r_i^k) is true or not. Let us define the vector $\mathbf{t}^k = (t_1^k, \ldots, t_{\ell_k}^k)$, of length $\ell_k - 1$, to be the *link label vector* for measured route k.

Having described the paths, we want to identify the flows that traverse them. This would permit a sender to select the path they desire for the packets they send. Let ϕ be a flow identifier, and Φ be the set of all possible flow identifiers. We are interested in dividing the universe Φ of flow identifiers into flow classes (where each flow of a given class is equivalent

in that it is treated the same way by per-flow load balancers), and identifying at least one representative flow from each flow class.

In this paper, we develop a traceroute that precisely reports and labels all routes between a source and a destination.

Let R_h be the set of all interfaces reachable at hop h when tracing towards a given destination. For each interface $r \in R_h$, we wish to find its successor interfaces at hop $h+1$. If r is the interface of a load balancer, there will be multiple successors. We call the set of successors the *nexthops* of r, and denote it as S_r. The union of nexthops for R_h gives us R_{h+1}.

For example, in Fig. 1, suppose we have already discovered $R_7 = \{L2, L3\}$. We build R_8 by successively enumerating the nexthops of each interface in R_7. We find $\{A, B\}$ (respectively, $\{C, D\}$) as nexthops for interface $L2$ (respectively, $L3$). Below, we describe our method for enumerating nexthops with a given level of confidence.

The Multipath detection algorithm proceeds hop by hop, and explores the IP-level graph by enumerating the nexthop interfaces of each interface discovered, until probing reaches the destination. During the exploration, it attaches representative flows to each interface, when possible.

This exploration supposes that each load balancer evenly balances traffic along its d outgoing interfaces. For per-packet load balancing, this means that each interface receives $1/d$ of the forwared packets. For a per-flow load balancer, we assume that each interface carries $1/d$ of the flows. We also assume that per-flow load balancers use a static mapping of flows to interfaces. In practice, we found that these assumptions hold much of the time, though we have yet to produce precise measurements. We are also currently working on improved methods to relax these assumptions.

A. Multi-Interface Hops

Let us call the set of interfaces that may respond to traceroute probes towards a given destination, for a given hop, a *traceroute hop*. A traceroute hop generally contains a single interface, for example at hops 6 and 10 in Fig. 1. Let us call a traceroute hop that presents multiple interfaces, a *Multi-Interface Hop*, or MIH for short, and the total number of possible interfaces in a given MIH, the *MIH width*.

For instance, supposing that $L1$, $L2$ and $L3$ are load balancers, a traceroute performed in the topology presented in Fig. 1 has potentially three MIHs. Hop 7 can return any of the combinations of interfaces $L2$ and $L3$, thus creating an MIH of width 2. Similarly, hop 8 is also an MIH containing 4 interfaces, and hop 9 an MIH of width 2.

While MIHs are typically produced by load balancing, another possible explanation is a routing change that occurs during the probing of a hop. Yet, routing changes that affect a given end-to-end path are relatively infrequent [16]. Therefore, the chance that a routing change will fall within a given traceroute will be small. Even if a routing change does cause an MIH, it will not recur systematically at the same point in multiple traceroutes. Repeating the traceroute to the same destination would allow us to distinguish those cases from load balancing. This feature is not yet part of the algorithm.

B. Algorithm overview

Algorithm 1 presents our probing strategy to trace the paths from hop $h_{\min}$ to $h_{\max}$. We present a simplified version of our algorithm, as we consider that we receive a response for each probe we send. This means that there is neither packet loss nor routers that limit the number of ICMP packets they send. Unresponsive routers introduce probing subtleties which we are not fully able to handle. Our algorithm proceeds hop by hop. The probing of each hop h depends on the interfaces found at the previous hop $h-1$. We use a fake interface (which we denote with 0, or *null interface*) as an artifact for the algorithm initialization. For each interface r of $\hat{R}_{h-1}$, Algorithm 2 builds the set of its nexthops, $\hat{S}_r$. If $|\hat{S}_r| > 1$, then r is an interface of a load balancer. In that case we use Algorithm 3 to issue extra probes through r and determine the type of load balancing. Finally, we output informations related to r: IP address, type of load balancing, and list of flow identifiers that traverse r, which we note $F_{h-1,r}$ (Algorithm 2 built this set during a previous iteration). We use a delayed output because some informations on r (like the type of load balancing) are unknown before the probing of its nexthops. We repeat the previous procedure for each interface r that we previously discovered at hop $h-1$, in order to discover the full set of interfaces of h, R_h. Table I summarizes the symbols we use in the following discussion and algorithms.

Algorithm 1 Discover the paths from hops $h_{\min}$ to $h_{\max}$ to the $100(1-\alpha)\%$ confidence level

```
1:  procedure MultipathTraceroute(h_min, h_max, α_all)
2:      R̂_{h_min−1} ← {0}
3:      for h in h_min ... h_max + 1 do
4:          R̂_h ← ∅
5:          for each r in R̂_{h−1} do
6:              Ŝ_r ← NextHops(r, h, α_nexthops)
7:              R̂_h ← R̂_h ∪ Ŝ_r
8:              if |Ŝ_r| > 1 then
9:                  π_r ← PerPacket(r, h)
10:             end if
11:             Output(r, π_r, F_{h−1,r})
12:         end for
13:     end for
14: end procedure
```

Algorithm 2 formalizes our method to find all nexthops of a given interface r discovered at hop $h-1$. This algorithm takes as input r, the hop to probe, h, and the desired confidence level (expressed using a parameter α). When the algorithm terminates, $\hat{S}_r$, our estimate for S_r, will contain the set of nexthop interfaces discovered for r. The algorithm proceeds iteratively. At each iteration, it readjusts the number of probes to send, max, according to the number of interfaces it has already found, $|\hat{S}_r|$. For each probe to send, lines 8 through

TABLE I
SYMBOLS USED IN THIS PAPER

Symbol	Meaning
r, s	responding interface or *successor* of an interface
h	hop (TTL value)
α	degree of confidence
$\hat{R}_h$	set of interfaces discovered at distance h from the source
$\hat{S}_r$	set of nexthop interfaces of r
ϕ	flow identifier
$F_{h,r}$	set of flows traversing r at hop h
π_r	Boolean indicating if r belongs to a per-packet load balancer

11 select the flow identifier to use for the k^{th} probe, issue the probe, and wait for the discovered interface, s. The algorithm keeps sending probes until it does not find any new interface. At this stage it halts the repeat loop and returns the set of interfaces. Note that, at present, we only apply the confidence level α_{nexthops} to the task of completing each individual nexthop set. A simple extension will soon allow the algorithm to find the entire set of measured routes to a desired level of confidence α_{all}. Indeed, we have to find the value of α_{nexthops} such that $\alpha_{\text{all}} < 0.05$.

This high-level description of the algorithm simplifies some important aspects. We now discuss each of the main building blocks of this algorithm in more detail.

1) Nexthops enumeration (NEXTHOPS)*:* Let us focus on a particular interface r at hop $h-1$, and suppose that we want to enumerate all its nexthop interfaces.

Algorithm 2 Find the nexthops (at hop h) of interface r to the $100(1-\alpha)\%$ confidence level

```
1: function NEXTHOPS(r, h, α_nexthops)
2:     k ← 0
3:     Ŝ_r ← ∅
4:     repeat
5:         n ← |Ŝ_r| + 1
6:         max ← PROBESTOSEND(n, α_nexthops)
7:         for k in k+1 ... max do
8:             φ ← SELECTFLOW(k, h−1, r)
9:             s ← SENDPROBE(h, φ)
10:            Ŝ_r ← Ŝ_r ∪ {r}
11:            F_{h,s} ← F_{h,s} ∪ {φ}
12:        end for
13:    until |Ŝ_r| < n
14:    return Ŝ_r
15: end function
```

We model the probing of the nexthops of r as a sample from a random variable $X_{h,r}$ uniformly distributed over the set S_r of nexthop interfaces. At any point in our probing, having sent k probes, we will have seen some set of interfaces, which will be our estimate for S_r. We call this set $\hat{S}_r$.

We hypothesize that, having sent k probes, we have not discovered all of the interfaces in S_r. That is, $H_0 : |S_r| > |\hat{S}_r|$. If we can rule out this hypothesis with a high degree of confidence, we can stop probing. We want to rule it out to the $100(1-\alpha)\%$ level of confidence. To do this, we consider the worst case, which is $|S_r| = |\hat{S}_r| + 1$. (If there are more interfaces at the next hop, then it is more likely that we will discover one of them with a new probe.) Let us define $n = |\hat{S}_r| + 1$. We are interested in the probability of having seen, after k probes, only $n-1$ interfaces. We define a random variable, $Y_{h,k}$, that represents the number of nexthop interfaces of r seen at hop h after k probes. For $Y_{h,k} = n$, this probability is:

$$\Pr[Y_{h,k} = n] = 1 - \frac{\sum_{i=0}^{n-1} \binom{n}{i} i^k (-1)^{n-i-1}}{n^k} \qquad (1)$$

We derived (1) from basic axioms of probability detailed in [19]. It represents the sum of the probabilities to see exactly i interfaces among n, for i varying from 1 to $n-1$. This probability of the union of several events is equal to the probability to see exactly a single interface among n, minus the probability to see exactly 2 interfaces of n, plus the probability to see exactly 3 interfaces of n, and so on, up to $n-1$.

We can stop probing if we have sent k probes and $\Pr[Y_{h,k} = n] < \alpha$. In Algorithm 2, PROBESTOSEND calculates k in terms of n and α. We do not have a closed-form expression to calculate k but it is easy to compute with a binary search. For instance, we have found the values in Table II for the 95% confidence level ($\alpha = 0.05$).

TABLE II
NUMBER OF PROBES TO SEND (k) VERSUS NUMBER OF EXPECTED INTERFACES (n), FOR A 95% DEGREE OF CONFIDENCE

n	2	3	4	5	6	7	8	9	10
k	6	11	16	21	27	33	38	44	51
n	11	12	13	14	15	16	17		
k	57	63	70	76	83	90	96		

We are currently perfecting this stopping rule, to take into account a slight bias, that sometimes requires one additional packet to be sent in order to achieve the desired level of confidence.

2) Flow selection (SELECTFLOW)*:* In order to randomly sample the set of nexthop interfaces of r at hop h, we change the flow identifier of each probe. We use a hash function (SELECTFLOW) from the probe index number to the range of source port numbers (for UDP and TCP) or the ICMP *Sequence* number (for ICMP), which indirectly changes the flow by modifying the ICMP *Checksum* [2]. The hash function returns a random integer in the range 10,000 and 65,535. For UDP and TCP, we only vary the source port and maintain the destination port. Indeed, we found some routers which send ICMP errors only if the probes use the typical destination port range of classic traceroute (that is, 33,435 and following). Futhermore, maintaining the destination port helps traversing firewalls by emulating legitimate (typically, Web) traffic, as does tcptraceroute [20].

SELECTFLOW also has to select flows which traverse r. For instance, the probing of the nexthops of $L2$ in Fig. 1 requires the selection of probes that will traverse $L2$, and the ignoring of the ones that traverse $L3$. We use the set $F_{h,r}$ to keep track of the flow identifiers traversing each nexthop interface r. In

practice, we already know a set of flow identifiers that traverse $L2$, which we discovered during a previous probing step. If this set is too small, we proceed with trial-and-error. We select a candidate flow identifier and send a probe to hop $h-1$. If r responds to the probe, we can use the identifier to probe its nexthops. If we receive a response from another interface, we reject it, and keep probing until we find a valid identifier. We can bound the maximum number of probes to send in order to find a valid candidate. Indeed, we are looking for a particular interface r, and already know the total number of interfaces that may respond, $|\hat{R}_{h-1}|$. In Fig. 1 for example, we may look for $L2$, knowing that we sample among 2 interfaces, $L2$ and $L3$. Thus, we have to find k such that $(\frac{|\hat{R}_{h-1}|-1}{|\hat{R}_{h-1}|})^k < \alpha$. We can give up probing if we do not receive a response from r after k successive probes sent.

3) Probe retransmission (SENDPROBE)*:* Unlike classic traceroute algorithms, in our case it is crucial to have responses for all the probes we have sent to reach the desired level of confidence. We use a very simple mechanism, which retransmits all the unresponsive probes as soon as a configured response timeout is triggered. We keep on trying until all the probes get a response, but stop if we do not receive new responses between two consecutive attempts. This simple technique allows us to handle the temporary outages (probe loss and ICMP-limiting routers). It also detects permanent cases of non responsiveness (for example, routers that do not send ICMP error messages), and abandons very quickly.

4) Load balancing classification (PERPACKET)*:* Once Algorithm 2 has terminated, and if r has multiple nexthop interfaces, we use the PERPACKET function (formalized in Algorithm 3) to categorize the type of load balancing. We first suppose that r does per-packet load balancing over its nexthop interfaces (that is, H_0: $\pi_r = 1$; recall that π_r is a Boolean indicating whether interface r belongs to a per-packet load balancer), and send k extra probes having the same flow identifier. In general, for a per-packet load balancer with d outgoing interfaces, the probability of detecting a single of its nexthop interfaces among d decreases as a function of d : $\frac{1}{d^{k-1}}$.

Algorithm 3 Test whether interface r (at hop $h-1$) belongs to a per-packet load balancer

```
1: function PERPACKET(r, h)
2:     φ ← SELECTFLOW(1, h − 1, r)
3:     X ← ∅
4:     for i in 1 . . . 6 do
5:         s ← SENDPROBE(h, φ)
6:         X ← X ∪ {s}
7:     end for
8:     return |X| ≠ 1
9: end function
```

Therefore, we can rule out H_0 to the degree of confidence $1-\alpha$ if a single interface responds to the k probes. For the worst case ($d = 2$) and a confidence level of 95% ($\alpha = 0.05$), we need to send $k = 6$ probes.

If we receive responses from a single interface, we can rule out H_0 and deduce that r belongs to a per-flow load balancer. Otherwise, either r belongs to a per-packet load balancer, or a routing change occured during the probing of its nexthop interfaces. For the moment we do not distinguish between these two cases. Repeating the probing and cross-checking the results would be a simple solution to this problem.

5) Output (OUTPUT)*:* We enrich the output of Paris traceroute with the list of all interfaces reachable at each hop, as well as the list of the flow identifiers $F_{h,r}$ associated with each interface r. We represent flow identifiers as small integers that we generate to uniquely identify a flow. In order to detect the presence of non-responsive routers in a load-balanced path we also print the list of flow identifiers for which we did not receive any response.

The following hypothetical and simplified (we removed the RTT information and only kept a few flow identifiers) extract of a trace illustrates how flow identifiers are used to track a flow. We print the flow identifiers after each interface address:

```
13  L.L.L.L
14  a.a.a.a:0,1,3   b.b.b.b:2,4,5
15  c.c.c.c:0,1,3   d.d.d.d:2,4,5
```

The single interface at hop 13 belongs to a per-flow load balancer that directs flows to two next-hop interfaces. For instance, the integer 0 uniquely identifies a flow that is associated to interface $a.a.a.a$ at hop 14, and to interface $c.c.c.c$ at hop 15. The same is true for flows 1 and 3. On the other hand, flows 2, 4 and 5 traverse the two other interfaces.

Under per-flow load balancing, we can easily build the link label vector from this information. For example, in the previous trace we would build two links: $a.a.a.a \rightarrow c.c.c.c$ and $b.b.b.b \rightarrow d.d.d.d$. Under per-packet load balancing, the flow identifiers are not relevant, thus they are not printed. We can build all the possible links between the interfaces of a given hop h and the interfaces of hop $h+1$, but we have no way to distinguish the true links from the false ones.

We also slightly modify the display of useful data reported by classic traceroute. First, the multiple probes sent by the algorithm generally gives us several RTT measurements for each responding interface. We print the minimum, maximum and average RTT, as well as the standard deviation of the delays to each interface we discover. Paris traceroute can also print the return TTL associated to each interface. We expect to see different return TTLs for the same interface, which may help infer the presence of load balancing on the return path. Finally, we print all the MPLS labels carried in the responses. Different MPLS labels associated to the same interface may help infer the presence of hidden load balancing over undetectable MPLS tunnels. We have not exploited those results yet.

IV. MEASUREMENT METHODOLOGY

This section describes our experimentation setup. We perform small-scale measurements to verify the ability of the

Multipath detection algorithm to find multiple paths between a source and a destination, compared to classic traceroute.

We ran our experiments from a single source located at the Université Pierre et Marie Curie, in Paris, France. Our source has only one link to the internet, via the French academic backbone, Renater. We built a destination list of 5,000 randomly-selected IP addresses that responded to a ping at the time of the construction of the list.

For each destination d in the list, we issue a Paris traceroute to d, using the Multipath detection algorithm, immediately followed by a classic traceroute with three probes per hop. This side-by-side measurement allows us to compare Paris traceroute and classic traceroute in terms of number of interfaces discovered. Both traceroutes use UDP. We set the maximum hop number to 36. To accelerate the measurements, both traceroutes stop after 3 consecutive unresponsive hops. We set the response timeout for each probe to 2,000 ms. For Paris traceroute, the interval between probes is 50 ms. Classic traceroute waits for a probe reply before sending the next probe. Finally, we use a 95% degree of confidence for the Multipath detection algorithm.

We conducted the experiments during the month of September 2006, and present results from a typical round.

V. Experiments

This section first provides a characterization of load balancing. Then, it evaluates the ability of the Multipath detection algorithm to find additional interfaces, compared to a classic probing algorithm.

A. Characterization of load balancing

This section shows the prevalence of load balancing as seen from our single source. Let us first address the problem of detecting and counting load balancers using the Multipath detection algorithm. We count as a load balancer each interface with multiple nexthop interfaces. Note that this method might overestimate the total number, since it maps each interface to a load balancer, whereas some interfaces may belong to a single load balancer. The routes to 1,525 of the 5,000 destinations in our measurements are affected by per-flow load balancing. This is 30%, whereas per-packet load balancing affects less than 2% of the routes. This is mostly due to the fact that the vast majority of load balancers observed use the per-flow technique. In a typical round, we find 327 distinct per-flow load balancers and only 55 per-packet load balancers.

There is a disparity between the small number of load balancers observed and the large number of paths affected by them. We found that the 20 most frequently-seen per-flow load balancers affect 78% of the paths that are subject to such load balancing. Indeed, the most frequently-seen load balancer affects 38% of such paths. This load balancer is a Level3[2] router that connects to Renater. In contrast, each per-packet load balancer tends to affect just a few paths.

We also study the partition of load balancers among ASes. Overall, we encountered load balancers in 89 ASes (7% of all the ASes our traces traverse), and load-balanced interfaces in 175 ASes. Fig. 1 helps us explain this difference. Suppose that hops up to #7 belong to AS1 and the following hops belong to AS2. Therefore, we have three load balancers and six load-balanced interfaces (A to F). We have found per-flow load balancers in eight out of nine tier-1 ISPs. For instance, we found 66 distinct load balancers in Level3 and 12 in the Savvis network.

Note that we also found a few per-packet load balancers in some tier-1 ISPs. They were all located at the edge of the network, suggesting that they are perhaps operated by customers and numbered using the tier-1's address space.

Although our results are biased by the connectivity of our vantage point, they do show that load balancers are becoming common in large ISP networks. These networks compose the core of the internet and most of the paths across the internet traverse them. Especially for these networks, which mostly use per-flow load balancing, the use of Paris traceroute greatly improves the precision and completeness of the measured routes.

B. Interface discovery

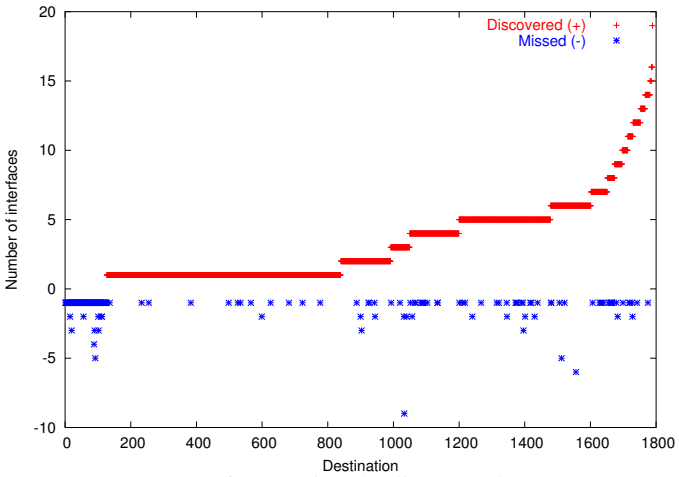
The prevalence of load-balanced paths in our traces has an impact on the completeness of the paths discovered by classic traceroute.

In Fig. 2(a) we plot, for each destination, the number of interfaces that Paris traceroute uniquely discovered (positive y-axis) or missed (negative y-axis). We do not plot the destinations where both traceroutes found the same set of interfaces. In 1,005 cases, Paris traceroute found at least 2 more interfaces than classic traceroute. There were 598 cases where a classic traceroute missed at least 5 interfaces.

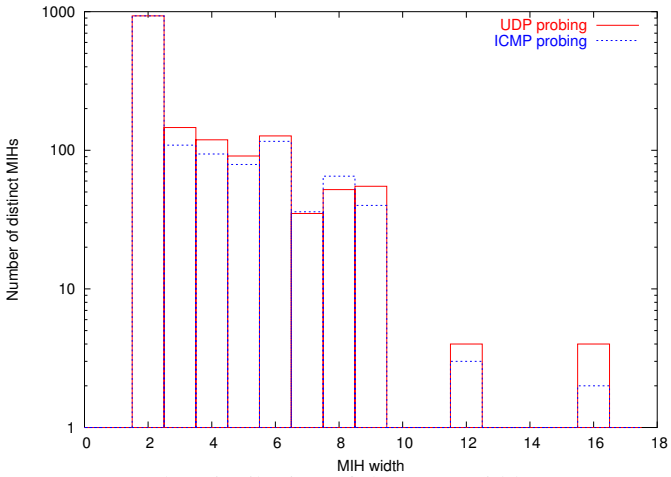
Nevertheless, in 201 cases classic traceroute also found interfaces that Paris traceroute missed, which reveals the difficulty one have to make precise internet paths measurements. In most of the cases, the cause of the miss is a temporary outage that our simplistic retransmission algorithm cannot handle. We also observed cases of a routing change between the measurements of Paris traceroute and classic traceroute. (Since we do not launch Paris traceroute and classic traceroute towards a destination at the same time, there is a delay of up to 30 seconds between the two measurements.) In those cases, the effects were either a modification of a portion of the path, or the quick appearance/disappearance of load balanced paths in some core networks.

In one round, the Multipath detection algorithm discovered 15,926 distinct IP addresses. Interestingly, classic traceroute discovered almost all of those interfaces (15,427 that is to say 96.9%). This is because many paths among the 5,000 paths traverse the same load balancers. Thus, even if a first classic traceroute only discovers a subset of an MIH, the following classic traceoutes traversing the same load balancers are likely to discover another subset of interfaces. Thus, with measurements to a sufficient number of destinations, classic traceroute will eventually discover the same set of interfaces as Paris traceroute.

[2]Level3 is a tier-1 ISP and is one of Renater's providers.



(a) Interfaces uniquely discovered (+) or missed (-) by Paris traceroute



(b) Distribution of the MIH width (y-axis in logscale)

Fig. 2. Evaluation of the interface discovery

Despite the fact that probing to multiple destinations will tend, over time, to fully reveal the topology close to the source, Paris traceroute demonstrates a clear benefit concerning the completeness of route discovery towards an individual destination. Thus, a network operator should find our tool particularly useful to diagnose a load-balanced path to a given destination. Our goal is not to discover most of the topology (as obtained with repeated classic traceroutes), but to discover all of it, to a given level of confidence, and with the smallest number of probes possible.

Fig. 2(b) plots the distribution of MIH widths. On the x-axis, we have the number of interfaces found in an MIH. The y-axis is the number of MIHs in our measurements with this number of interfaces. Note that we count each distinct set of interfaces only once. We also show the size of the MIHs found with UDP and ICMP probes. For UDP probing, we observe over 1,664 distinct MIHs. 933 MIHs (59%) have only 2 interfaces, and 489 (31%) present more than 3 interfaces. The maximum MIH width we found is 16. It is caused by a per-flow load balancer located in a peering point between a tier-1 and a Brazilian AS. Perhaps 16 is a built-in limit. For instance, one can configure at most 16 load balanced paths in a Juniper router [21].

Interestingly, we found fewer MIHs with ICMP probing. The majority of the per-flow load balancers use the five-tuple and some other fields (like the IP *Type of Service* and the first four bytes in the ICMP header) to define a flow. But some of them ignore ICMP packets contents to define a flow. As a result, such a load balancer will create an MIH with UDP probing, but not with ICMP probing. In our traces, about 20% of the per-flow load balancers found with UDP probing were undetectable with ICMP probing.

Finally, we quantify the overhead of probing, which is significantly higher than the three probes per hop of classic traceroute. For instance, for the confidence level we used (95%), the Multipath detection algorithm has to send at least 6 probes in presence of a single nexthop interface. We need 33 probes to discover a nexthop set of 6 interfaces, and 96 probes to discover the largest nexthop set in our traces. We need to add to this the overhead caused by the trial-and-error discovery process that we use to make sure our probes go through a particular interface. We are presently at work on a full accounting of the comparative overhead.

VI. Conclusion

In this paper, we introduce a new definition of a "route" between a source and a destination that takes load balancing into account. We propose the Multipath detection algorithm, the aim of which is to enumerate all load-balanced paths with a high degree of confidence. From the probing source used in our experiments, the routes to 30% of the destinations followed multiple paths. If our experience is at all typical, it points out the importance of considering load balancing when tracing routes. This study is based on very small measurements from a single vantage point, but we are currently conducting broader experiments from multiple vantage points. Preliminary results show that the results presented in this early study are far from anecdotal.

Most of the research on internet topology measurement focus on broad experiments from multiple sources to multiple destinations. However, load balancing points out interesting questions to solve in the base case of probing from a single source to a single destination. Futhermore, it also has impact on broad internet map discovery, since repeated traceroutes are likely to miss paths that are far from the sources, causing a poor coverage of some areas. Widely-deployed load balancing also calls into question the large-scale algorithms such as DoubleTree [7], which assume that the routes have a tree-like structure. Even more, since our method relies on the knowledge of previous hops to probe a given hop, designing a

backward probing algorithm that takes multipaths into account does not seem straightforward.

Although this algorithm represents an important first step, it is clearly not the final word. We plan to improve on it in a number of ways. We believe it might be possible to send fewer probes, given better knowledge of the hash algorithms that per-flow load balancers use [22]. We also believe it is possible to trace accurately past per-packet load balancers. In our observations, they tend to use a round-robin forwarding algorithm [23]. We plan to use packet trains to help us determine the real paths packets take. We also observed rare cases of uneven load balancing. We have not conducted any systematic measurements to detect it, but we believe it is a very simple extension to the current algorithm. For instance, imagine a case where a router splits traffic over three links, which respectively receive $\frac{1}{2}$, $\frac{1}{4}$ and $\frac{1}{4}$ of the flows. We can detect the three nexthop interfaces with the same degree of confidence by reducing this case to a typical case where the Multipath detection algorithm expects 4 interfaces instead of only 3.

This work should inspire research in many directions. First, we plan to revisit previous studies based on classic traceroute measurements. For instance, our view of routing dynamics might be significantly biased by the measurement artifacts that arise when using classic traceroute in load-balanced paths. Furthermore, the Multipath detection algorithm will also allow us to build more complete and accurate maps of the internet. It is clear that our probing technique does not significantly improves the coverage of the internet, in terms of number of discovered interfaces. However, it adds valuable information on load balancing, by grouping a set of paths, which seem independant in our traditional view of the internet map, into a multipath that can indifferently be selected by an end-user to reach a destination. Finally, the study of the path diversity that the network natively provides might lead to innovations in the optimization of end-to-end connections.

ACKNOWLEDGMENTS

We are grateful to Fabien Viger, to Matthieu Latapy, and to Clémence Magnien for their thoughtful comments. 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. The research using the tool was conducted with financial support from the French MNRT ACI *Sécurité et Informatique* 2004 grant, through the *METROSEC* project.

REFERENCES

[1] V. Jacobson, "traceroute," February 1989, the most recent version is available at: ftp://ftp.ee.lbl.gov/traceroute.tar.gz.

[2] 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*, October 2006.

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

[4] R. Govindan and H. Tangmunarunkit, "Heuristics for internet map discovery," in *Proc. IEEE INFOCOM*, March 2000.

[5] Y. Shavitt and E. Shir, "DIMES: Let the internet measure itself," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 71 – 74, October 2005.

[6] Z. M. Mao, D. Johnson, J. Rexford, J. Wang, and R. H. Katz, "Scalable and accurate identification of AS-level forwarding paths," in *Proc. IEEE INFOCOM*, March 2004.

[7] B. Donnet, B. Huffaker, T. Friedman, and kc claffy, "Evaluation of a large-scale topology discovery algorithm," October 2006.

[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM*, 1999, pp. 251–262.

[9] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, 2002.

[10] 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*, October 2003.

[11] A. Lakhina, J. W. Byers, M. Crovella, and P. Xie, "Sampling biases in IP topology measurements," in *INFOCOM*, 2003.

[12] S. Sinha, S. Kandula, and D. Katabi, "Harnessing TCPs Burstiness using Flowlet Switching," in *3rd ACM SIGCOMM Workshop on Hot Topics in Networks (HotNets)*, San Diego, CA, November 2004.

[13] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *INFOCOM*, 2001.

[14] C. Villamizar, "Ospf optimized multipath (ospf-omp)," Internet Draft, 1999.

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

[16] V. Paxson, "End-to-end internet packet dynamics," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, 1999.

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

[18] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *Proc. ACM SIGCOMM*, August 2006.

[19] S. Ross, *A First Course in Probability*. Prentice Hall, 2005, ch. 2, Axioms of probability.

[20] M. Toren, "tcptraceroute," April 2001, see http://michael.toren.net/code/tcptraceroute/.

[21] Juniper, "Configuring load-balance per-packet action," from the JUNOS 7.1 Policy Framework Configuration Guideline.

[22] Z. Cao, Z. Wang, and E. W. Zegura, "Performance of hashing-based schemes for internet load balancing," in *INFOCOM*, 2000.

[23] Cisco, "Configuring cisco express forwarding," from the Cisco CEF Documentation, see section "Configuring Per-packet Load balancing".