Classification of Load Balancing in the Internet

Rafael Almeida[†] Ítalo Cunha[†] Renata Teixeira[‡] Darryl Veitch[⋆] Christophe Diot[♯]
[†]Universidade Federal de Minas Gerais, Brazil [‡]INRIA, Paris, France
[⋆]University of Technology Sydney, Australia [♯]Google, Mountain View, CA

Abstract—**Recent advances in programmable data planes, software-defined networking, and the adoption of IPv6 support novel, more complex load balancing strategies. We introduce the Multipath Classification Algorithm (MCA), a probing algorithm that extends traceroute to identify and classify load balancing in Internet routes. MCA extends existing formalism and techniques to consider that load balancers may use arbitrary combinations of bits in the packet header for load balancing. We propose optimizations to reduce probing cost that are applicable to MCA and existing load balancing measurement techniques. Through large-scale measurement campaigns, we characterize and study the evolution of load balancing on the IPv4 and IPv6 Internet with multiple transport protocols. Our results show that load balancing is more prevalent and that load balancing strategies are more mature than previous characterizations have found.**

I. Introduction

Internet traffic load balancing helps with increasing bandwidth, reliability, and reducing maximum link utilization. Load balancing can be configured in routers manually, or automatically by mechanisms such as Equal-Cost Multi-Path (ECMP). Routers that perform load balancing, which we call *load balancers*, compute which network link a packet should be forwarded to as a hash function of the packet's *flow identifier*, a subset of fields in the packet's headers (e.g., IP addresses and port numbers) [1].

The standard tool for measuring Internet routes is traceroute [2]. Classic traceroute uses the destination port to store probe identifiers, which often triggers load balancing and causes measurement artifacts that lead to incorrect inference of inexistent links, loops, and cycles [3]–[5]. Paris traceroute [1], [3] is an extension of traceroute that fixes flow identifiers to steer probes over a single link at load balancers, avoiding such measurement artifacts. The Multipath Detection Algorithm (MDA) extends Paris traceroute and systematically varies probes' flow identifiers to identify links used for load balancing at a router (if any) [6].

Recent advances in programmable data planes, software-defined networking, and the adoption of IPv6 lead to increasingly complex load balancing deployments [7]–[17]. Existing MDA implementations, however, only vary transport port numbers and the ICMP checksum [6], [18]. Previous work have applied ad-hoc extensions to MDA to classify load balancers as per-packet, per-flow, or per-destination [3], [19], [20], but these three predefined types are not comprehensive and the classification method does not generalize to other load balancer types. As a result, MDA may fail to identify new types of load balancing. Indeed, our results show that MDA deterministically returns incomplete route measurements for 2.6% of IPv4 and 1.8% of IPv6 routes. Also, applying the existing ad-hoc classification method leads to deterministic misclassification of 4.0% of IPv4 and 12.7% of IPv6 load balancers in our measurements (§VIII).

In this paper, we extend the existing formalism and router model [6] to allow more general load balancer behavior (§III). We introduce the Multipath Classification Algorithm (MCA), a probing strategy that identifies the set of bits in the packet header used by load balancers (§IV). We release an implementation of MCA and a route analysis tool (§V).

MCA increases the probing cost of route measurements [6], [21]. To balance this, we present optimizations that reduce the average number of probes sent on MCA measurements by 6% for IPv4 and 8% for IPv6, without any loss of accuracy (§VII). Overall, MCA is practical, with a probing cost only 34% higher than that of MDA.

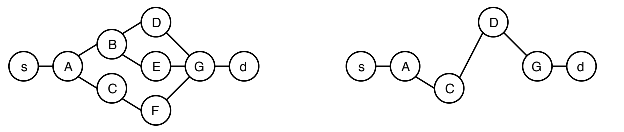
We conduct large-scale measurement campaigns using MCA, collecting measurements of IPv4 and IPv6 paths in the Internet using multiple transport protocols from a diverse set of vantage points (§VI). We use these to characterize the prevalence and behavior of load balancing in the Internet today, and reappraise previous characterizations [3], [20] (§VIII). Our results indicate that load balancing is more prevalent and their configurations more complex than previously reported. In particular, 74% of IPv4 and 56% of IPv6 routes traverse at least one load balancer, and 23% of IPv4 and 18% of IPv6 load balancers have three or more next hops.

This work improves the *foundations* upon which we build tools to identify load balancing in the Internet. Our implementation of MCA provides more accurate information than existing tools, and can be useful in network characterization studies, particularly those concerned with traffic engineering and reliability to failures.

II. Definitions and Background

We follow the same notation as in previous work [22]. At any given time, the connectivity between a fixed source s and a destination d is realized by its *current route*. A *route* can be *simple*, consisting of a sequence of IP interfaces from s toward d, or *branched*, when one or more *load balancing* routers (LB) are present, giving rise to multiple overlapping sequences (called "multi-paths" in [6]). Thus a route is a directed graph with interface-labelled nodes. A route can be a sequence that terminates before reaching d due to routing changes or the absence of a complete route to the destination.

We define a *diamond* as the set of all interfaces between an LB and its *join point*, the interface where all the sub-branches



(a) Branched route (b) Possible traceroute output

Figure 1: Example branched route and traceroute output

originating from the LB first rejoin. An *outer* diamond is one that contains other *nested* LBs, and therefore their own diamonds. We define an *outermost diamond* as one which is not contained in some other diamond. Figure 1a shows an example branched route where the LB A defines an outermost diamond with join point G, containing a diamond defined by nested LB B, also with join point G. A branched route can have multiple (nonoverlapping) outermost diamonds.

By *hop-set* or *hop* we define the set of interfaces found at some fixed distance, or *radius*, from the source. We denote the hop-set at radius r in route p as $p[r]$. Conversely, for some valid hop-set h in route p, we let $p\langle h\rangle$ denote the radius of h. Thus $h = p[p\langle h\rangle]$. In Figure 1a, $p = [s, A, \{B, C\}, \{D, E, F\}, G, d]$ is a route with three branches. The first hop is $p[1] = \{A\}$, the second hop is $p[2] = \{B, C\}$, and hop $\{G\}$ is found at radius $p\langle\{G\}\rangle = 4$. Assuming loop-free routes, hops are unique, although individual interfaces can be found in multiple hops when route branches have different lengths.

A. Branched Routes and MDA: Key Idea and Limitations

Classic implementations of traceroute may miss routers and links as well as infer links that do not exist when measurements traverse load balancers [1], [6]. Figure 1 shows an example branched route (left), where A and B are load balancers, and a possible (incomplete) traceroute output (right).

The Multipath Detection Algorithm (MDA) is a probabilistic algorithm to map out the branched route between a given source s and destination d with statistical guarantees [6]. Assume an interface i at radius r in some branch of the route is known. MDA tests if i is a load balancer by sending multiple probes through i that expire at radius $r + 1$, using randomly varied packet headers to stimulate load balancing. Assuming the stimulation results in a uniformly distributed selection over the $K \geq 1$ next hops of i, MDA can specify the number of attempts that should be made, as a function of the number $1 \leq k \leq K$ of interfaces found so far, in order to provide statistical guarantees that all K have been found, and hence, when applied recursively, that all branches of the route have been discovered.

For example, to identify the branched route in Figure 1a with a confidence level $\alpha = 0.95$, MDA begins by sending 9 probes with varying flow identifiers to A's next hop. The probability that a second iterface will not be found (assuming a uniform LB), is $0.5^9 = 0.001953$. Once a second interface is seen, MDA labels A as an LB then sends an additional 8 probes in an attempt to find more. Since no more exist, MDA gives up and proceeds to the next hop after having expended 17 probes. This process repeats until MDA reaches the destination.

The common limitation of LB discovery approaches is the assumptions made about the nature of the LBs; in particular, the header fields on which they are assumed to base decisions on. Most commonly, port number fields are assumed, the so called *per-flow* LBs. However, it is well known that other kinds exist, for example *per-packet*, where branch selection is random, independent of packet headers.

Some MDA implementations [3], [19], while being primarily per-flow based, have included ad-hoc extensions to provide a degree of LB classification. For example: first send multiple probes with fixed port numbers (i.e., identical packet headers) and classify any identified LBs as per-packet. Then vary port numbers and classify any additional LBs as per-flow. Finally, vary the last bits of the destination IP address and classify any further ones as *per-destination*.

The above approaches are imperfect, and still rely on knowing the nature of deployed LBs. The landscape is made more complex by IPv6, which may rely on the IPv6-specific flow label field introduced to facilitate load balancing [23], and SDN, which enables network administrators to configure LBs over a large set of packet header fields [7]. There has been no systematic study of which fields are used for load balancing and how they vary across routers.

III. Load Balancer Model

Load balancing at a given router has two main aspects: (i) *when* load balancing is performed, and (ii) the *type* of load balancing when it is performed.

Aspect (i) is a function of whether multiple routes (through different next hops) to a destination prefix are known, the incoming link at the router, and the policy in place. To capture these dependencies, we test for LB behavior at the granularity of $\langle$interface IP, destination prefix, upstream IP$\rangle$ triples, where the interface and upstream IPs are those returned by traceroute, used here, as usual, as surrogates for the true, unknown interfaces.

Aspect (ii) is a function of the router's forwarding decision algorithm. We model this as an ideal hash function over a *hash domain* consisting of a subset, in general not contiguous, of packet header bits. Each possible bit-field value over the domain is mapped to one of the available next hops. We consider *hash domains* are LB-specific.

The goal of the above model is to capture very general LB behavior. For (i), it allows load balancing to be triggered on some paths but not others, and the load balancing type to be path dependent. For (ii), it includes the classic LB types mentioned earlier, but allows for much richer structure that router vendors or SDNs may be implementing today or in the future. Per-packet LBs are included as the special case of an empty hash domain.

The generality of arbitrary hash domains may seem intractable at a practical level. There are two important reasons why this is not the case which we exploit in the sections below:

1) *LB discovery:* By sending probes which vary all header bits (within some given target set), the presence of a LB can be detected *without needing to know* the exact hash

domain. Moreover, if we assume the hash function maps uniformly over the next hops, then the same stopping rules and statistical guarantees used for MDA [6] still apply *regardless* of the unknown hash domain.

2) *LB classification:* It is not necessary to fully determine the hash domain to gain useful results. For example, under the uniform hash assumption, if we vary a single field (e.g., origin port) uniformly, we will induce a uniform sampling of the next hops provided the hash domain and the field have a non-empty intersection. Thus, the involvement of fields in LB decisions can be determined without a full resolution at the bit level.

Hash functions are employed to approximate random selection of next hops, but are actually deterministic. This can result in undesirable *polarization* effects when nested LBs are of the same type. To describe this we define, for each interface i in hop $p[r]$ of route p, the set $\text{flows}(i, r)$ of *flow identifiers* (bit-field values in the packet header) with which $i \in p[r]$ can be reached. For example in Figure 1a, if A and B were polarized, then all flows that A sends to B would be sent on to D, that is $\text{flows}(B, 2) = \text{flows}(D, 3)$. It would then be *impossible* to observe interface E, regardless of the probing strategy used. Polarization has been observed in real routers and flagged as undesirable, as it prevents effective use of all available routes, and modern implementations include mechanisms to avoid it. For example, Cisco and Juniper hash a router-specific identifier together with the packet's flow identifier, while Arista and Cumulus allow configuration of a seed for the hash function.

Another challenge to identifying load balancing is dealing with non-uniform hash functions. These are sometimes deployed to better align link capacity and traffic volumes [8], [24]. For example, in Figure 1a, router A could send $2/3$ of the hash domain towards B and the remainder towards C. Our techniques can identify and classify non-uniform load balancers, however our bounds on the probability of error assume uniformity.

IV. Multipath Classification Algorithm

MCA receives as input the destination IP address d to trace; a confidence level α_d to bound the probability of detection errors, a confidence level α_c to bound the probability of classification errors; and a set $\mathcal{B}$ of *target* bits in the packet header defining the scope of LB detection and classification. We call the set of header fields that intersect $\mathcal{B}$ the *target* header fields, denoted $\mathcal{F}$.

MCA measures a route p hop-by-hop in increasing radius order. At each radius r, MCA first executes a modified version of MDA to enumerate next hops of routers in $p[r]$ that perform load balancing involving bits in $\mathcal{B}$ (§§IV-A and IV-D). MCA then sends additional probes to classify the type of any LBs found (§IV-B). During both detection and classification phases, MCA employs optimizations to reduce the probing cost (§IV-C). After reaching the destination or stopping conditions (§IV-E), MCA outputs a directed graph representing the route discovered and the classification of each LB.

A. Load Balancer Discovery

MCA computes the *number* of probes to send through an interface to enumerate next hops following MDA (§II-A, [6]).

The novelty of MCA is the application of a broad definition of an LB search 'universe' based on $\mathcal{B}$, operating in the context of an arbitrary LB operating on the hash domain $\mathcal{H}$. In theory, if the *flow identifiers* (the value of the bits in $\mathcal{B}$) carried by probes are chosen uniformly, then so will be the bits in $\mathcal{H} \cap \mathcal{B}$, resulting in uniformity over the next hops. In practice, as $|\mathcal{H} \cap \mathcal{B}|$ may be small, it is important for flow identifiers to vary all bit combinations as much as possible, to ensure variety over $\mathcal{H} \cap \mathcal{B}$. To achieve this we proceed recursively as follows.

Let ϕ_1, ϕ_2, $\ldots, \phi_n$ be the flow identifiers generated for the first n probes. We generate the new flow identifier ϕ_{n+1} greedily bit-by-bit in random order. The value of each bit is set such that it maximizes the Shannon entropy of the distribution of values seen over the $n+1$ identifiers, restricted to the bits considered so far, with ties broken randomly. If the generated ϕ_{n+1} repeats an earlier identifier, bits are randomly flipped until uniqueness is obtained. This greedy heuristic generates each flow identifier with $O(|B|)$ operations.

Although MCA (and measurement tools in general) cannot control all bits in the source and destination addresses while performing a measurement, MCA still supports testing whether they are included in hash domains by allowing $\mathcal{B}$ to include the least significant bits of source and destination addresses.[1] Later we also vary the source IP at a higher level, via exploiting multiple vantage points.

B. Load Balancer Classification

For each LB identified in the previous step, our goal is to identify which header bits in $\mathcal{B}$ overlap its hash domain.

For each interface i identified as a LB, MCA chooses one flow identifier known from the detection phase to traverse i, and sends multiple probes with that *fixed* identifier to i's next hop. We compute the number of probes to send as in MDA [6]. MCA classifies i as a per-packet LB if responses arrive from multiple next hops.

If interface i is not classified as per-packet, MCA sends additional probes according to Algorithm 1 to infer which bits in $\mathcal{B}$ overlap with i's hash domain and classify the LB type (line 3). The number n of trials to perform for each bit is a function of a configurable confidence level α_c that determines the acceptable probability of classification error (line 2).

In each trial, MCA identifies a flow identifier ϕ that traverses i and a flow identifier ϕ' that also traverses i and is identical to ϕ except for the value of b (line 5, §IV-C). MCA sends probes with ϕ and ϕ' to i's next hop (if not yet sent) and waits for the responses (lines 6–7). This process is repeated until MCA identifies flow identifiers ϕ and ϕ' that i forwards to different interfaces (lines 8–10), or until enough trials are performed to infer that b does not overlap with i's hash domain at the configured confidence level (i.e., when $j = n$ in line 4).

[1]Varying the source IP address requires the measurement device to control multiple IP addresses (e.g., when allocated a /64 IPv6 address).

Algorithm 1: Classify load balancer interface i at hop h

```
    Input  : Load balancer interface $i \in h$ with set of next
             hops $\mathcal{N}$, known not to be a per-packet load
             balancer; target bit set $\mathcal{B}$; confidence level $\alpha_c$
    Output: Interface $i$'s hash domain $\mathcal{H}_i \subseteq \mathcal{B}$
 1  $\mathcal{H}_i \leftarrow \emptyset$
 2  $n \leftarrow$ NumTrials($|\mathcal{B}|, |\mathcal{N}|, \alpha_c$)                Eq. (1)
 3  foreach bit $b \in \mathcal{B}$ do
 4      for $j \leftarrow 1$ to $n$ do
 5          $\phi, \phi' \leftarrow$ SelectFlows($i, h, b$)              §IV-C
 6          reply $\leftarrow$ ProbeReply($\phi, p\langle h\rangle + 1$)
 7          reply' $\leftarrow$ ProbeReply($\phi', p\langle h\rangle + 1$)
 8          if reply $\neq$ reply' then
 9              $\mathcal{H}_i \leftarrow \mathcal{H}_i \cup b$
10              break
11  return $\mathcal{H}_i$
```

MCA computes the number of trials to send when classifying an LB as a function of the confidence level α_c. We define the classification of an LB interface i as correct when *all* target bits in $\mathcal{B}$ are correctly labelled as overlapping or not with i's hash domain $\mathcal{H}_i$. We assume LBs distribute flow identifiers uniformly over next hops. Violation of this assumption increases the misclassification probability.

We denote the probability that an interface i forwards traffic to its next hop j as $P_{\text{next}}(i,j)$. Using the uniformity assumption across the set of next hops $\mathcal{N}_i$, we have $P_{\text{next}}(i,j) = 1/|\mathcal{N}_i|$ for all $j \in \mathcal{N}_i$.

Mislabelling can only occur for a header bit $b \in \mathcal{B}$ that overlaps $\mathcal{H}_i$. We compute the number of trials considering the worst case scenario where $\mathcal{B} \cap \mathcal{H}_i = \mathcal{B}$; if $\mathcal{B}$ is a proper subset of $\mathcal{H}_i$, then we overestimate the number of trials and the probability of error will be smaller than α_c. Mislabelling b when performing n trials varying b happens when b is in i's hash domain, and probes on all n trials are forwarded to the same next hop, which happens with probability $(1/|\mathcal{N}_i|)^{n-1}$. The probability of misclassification is then given by the probability that we mislabel any bit in $\mathcal{B}$:

$$P_{\text{miss}}(i,n,\mathcal{B}) = 1 - \left[1 - \left(\frac{1}{|\mathcal{N}_i|}\right)^{n-1}\right]^{|\mathcal{B}|}.$$

To bound $P_{\text{miss}}(i,n,\mathcal{B})$ below $1-\alpha_c$ we set

$$n = \left\lceil -\log_{|\mathcal{N}_i|}\left(1 - \alpha_c^{\frac{1}{|\mathcal{B}|}}\right)\right\rceil + 1, \quad (1)$$

To classify an interface i with confidence $\alpha_c = 0.95$ when $|\mathcal{B}| = 52$ and $|\mathcal{N}_i| = 2$, MCA first sends up to 9 probes to check if i is a per-packet load balancer and, if not, performs up to 10 trials (up to 20 probes) for each bit $b \in \mathcal{B}$.

C. Optimizations for Searching Flow Identifiers

In this subsection, we discuss the existing approach (baseline) for searching for new flow identifiers [19] and optimizations to reduce probing cost.

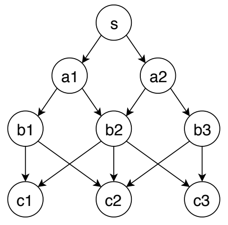


Figure 2: Example toplogy.

a) Problem: MDA and MCA need to generate a number of probes with varying flow identifiers through each interface i in a route to explore i's next hops and, if load balancing is identified, i's hash domain. Consider the MDA probing process for the route in Figure 2. MDA will send 17 probes[2] varying flow identifiers through s to identify s's next hops a_1 and a_2 (assuming both are found). To send a probe through i requires that MDA first check that the probe's flow identifier reaches i. MDA will also send 17 probes varying flow identifiers through a_1 to identify a_1's next hops b_1 and b_2. Any flow identifier MDA knows will reach a_1 from previous probes can be reused. However, MDA will need to send additional *search probes* to obtain a total of 17 identifiers that actually reach a_1.

b) Randomized search (baseline): New flow identifiers that reach a given interface are found by trial and error, and can amount to a significant fraction of the probing cost. The probability that a given flow identifier follows a branch segment $\beta = [i_1, i_2, \ldots, i_{|\beta|}]$, assuming each interface performs load balancing independently, is $P_{\text{branch}}(\beta) = \prod_{1 \leq k < |\beta|} P_{\text{next}}(i_k, i_{k+1})$. The probability that a flow identifier that goes through interface i at radius r_i also goes through interface j at radius $r_j > r_i$ is given by

$$P(i,j) = \sum_{\beta} P_{\text{branch}}(\beta), \text{ for all branches } \beta \text{ between } i \text{ and } j.$$

For example, in Figure 2, where we assume all interfaces distribute flows uniformly, $P(s,b_1) = \frac{1}{2}\frac{1}{2}$ and $P(s,b_2) = \frac{1}{2}\frac{1}{2} + \frac{1}{2}\frac{1}{2}$. The average number of trials to find a new identifier that reaches i from s is $1/P(s,i)$.

c) Reusing flow identifiers for identification: When searching for new flows that reach an interface i in order to enumerate its next hops, classic MDA will try new flow identifiers from the source s. Alternatively, we propose the reuse of flow identifiers sent to previous hops. We define a flow identifier ϕ as *reusable* if it has been sent to some radius smaller than i's radius and has not yet been sent to i's radius. For each flow identifier ϕ, we define the interface with the highest radius ϕ is known to reach as $\text{tip}(\phi)$. For each reusable flow identifier ϕ, we estimate the probability that it will reach interface $i \in h$ as $P(\text{tip}(\phi), i)$.

We try reusable flows in order of decreasing probability to reach i. Let $\mathcal{A}_i$ denote the set of ancestors of interface i in i's outermost diamond. In Figure 2, $\mathcal{A}_{c_1} = \{s, a_1, a_2, b_1, b_2\}$. The optimum flow trial order for finding new flows through c_1 is given by $P(b_1,c_1) = \frac{1}{2} > P(a_1,c_1) = \frac{5}{12} > P(b_2,c_1) = \frac{1}{3} >$

[2]We consider a confidence level $\alpha_d = 0.95$, which implies that MDA will send a maximum of 9, 17, or 24 probes through an interface that has one, two, or three next hops, respectively [6].

$P(s, c_1) = \frac{7}{24}$. When searching flows through c_1, MCA will *not* reuse any flow ϕ with $\text{tip}(\phi) = a_2$, as these flows have a lower chance, $\frac{1}{24}$, to reach c_1 than new random flow identifiers from the source (counted as a special case of reusable). Given that route measurement proceeds hop-by-hop and all radii up to i have been probed prior to identifying i's next hops, $\mathcal{A}_i$ and probabilities $P(a, i)$ of reaching i from all ancestors $a \in \mathcal{A}_i$ can be computed backwards from i in time $O(|\mathcal{A}_i|)$.

d) Searching sequence for identification: When identifying next hops of interfaces at a hop h, MDA may need to find a different number of new flows through each interface $i \in h$. The total number of trials required depends on the order in which we try flows. Let $M_h[i]$ denote the fraction of new flow identifiers missing for interface $i \in h$ relative to the total over h, and M_h the corresponding vector over all i. To minimize the total number of trials across h, we issue a probe with a flow identifier that best matches M_h. More precisely, for each trial, we select the identifier ϕ over all reusable flows that maximizes the utility

$$U(\phi) = \sum_{i \in h} M_h[i] P(\text{tip}(\phi), i). \tag{2}$$

We update M_h and recompute the optimal ϕ after each trial.

Example: Consider that MDA needs to find 1, 2, and 3 new flow identifiers through c_1, c_2, and c_3, respectively. Then $M_h[c_1] = \frac{1}{6}$, $M_h[c_2] = \frac{2}{6}$, and $M_h[c_3] = \frac{3}{6}$, and the flow reuse order is given by the utilities of available tips: denote Φ_x any flow ϕ such that $\text{tip}(\phi) = x$, then $U(\Phi_{a_1}) = 0.29 < U(\Phi_s) = U(\Phi_{b_2}) = 0.33 < U(\Phi_{a_2}) = 0.37 < U(\Phi_{b_3}) = 0.42$.

e) Varying a bit value for classification: After identifying next hops, MCA needs to send probes with flow identifiers ϕ and ϕ' that differ for a single header bit through an interface i to classify its behavior. It is potentially onerous to find enough of such ϕ', however here the fact that LBs on the route can be of different types, usually seen as a classification burden, can be exploited to obtain a supply of ϕ' efficiently. This optimization provided the most significant gains in our experiments, and works as follows.

Any flow identifier ϕ' that overwrites a header bit b in ϕ is guaranteed to also traverse i if b does not overlap the hash domains of LBs in the branch taken by ϕ to reach i. For such bits, MCA can generate a ϕ' flow identifier through an interface i from any known flow identifier ϕ without trials. This implies that bits that are seldom used for load balancing incur a known, fixed probing overhead given by Eq. (1).

The same property can be used when hash domains $\mathcal{H}$ of all LBs in a branch β are identical. In these cases, we can generate new flow identifiers by overwriting all bits in $\mathcal{B} \setminus \mathcal{H}$ in flows known to reach i through β, and the new flows are guaranteed to also reach i. This property allows us to repurpose flow identifiers generated during the next hop identification phase, each with a distinct value over $\mathcal{H}$, into probes useful for classifying i's hash domain.

Consider that s and a_1 in Figure 2 perform per-destination load balancing. Consider MCA knows flows $\phi_1, \phi_2, \ldots, \phi_k$ reach b_1 through s and a_1. We can generate flow identifiers that reach b_1 and vary a bit b not in the destination address by picking, e.g., ϕ_1, and overwriting b. We can also generate k flow identifiers that reach b_1 and vary only the destination address by overwriting all fields other than the destination address in $\phi_1, \phi_2, \ldots, \phi_k$, for example with a fixed value.

D. Varying a Field Value for Classification

We have yet to address the question of the choice of $\mathcal{B}$. It is potentially highly onerous to select a large set and determine each hash domain to bit resolution as described above, as each bit requires up to n trials. However, it is possible to work at the level of header fields instead. LB discovery is unchanged: the hash domains are still general, we simply set $\mathcal{B}$ to the union of the header fields of interest. For LB classification, the same algorithms work unmodified by operating on $\mathcal{F}$ instead of $\mathcal{B}$ ($\mathcal{F}$ was defined in the first paragraph of §IV as the set of packet header fields that overlap $\mathcal{B}$). The end result is simply to report only at field granularity: i.e., the fields which have been found to intersect each hash domain.

E. Operational Considerations

MDA can identify, and MCA can classify, routers that load balance traffic over *visible* MPLS tunnels [25], [26], that is those that reuse the probe's TTL field value and whose router's ICMP time exceeded responses include the MPLS label stack. MDA can identify load balancing (but not the intermediate hops) and MCA can (partially) classify load balancing over invisible MPLS tunnels when, (i) probes arrive at different interfaces at the router where the invisible MPLS tunnels end, and (ii) the router originates ICMP time exceeded replies using different IP addresses.

Some approaches to load balancing include the TTL in the hash domain [24]. These mechanisms make it impossible to control the branches a flow identifier will follow, as traceroute cannot function without varying probe TTL.

When varying the last bits of the destination address inside the destination's AS, MCA probes may discover subnets other than the destination's [27]. However, MCA only finally reports interfaces found inside the destination AS that are observed on probes targeting the original destination.

V. MCA IMPLEMENTATION

We implement MCA in a command-line tool to identify and classify load balancers. Our implementation supports TCP, UDP, and ICMP measurements using both IPv4 and IPv6. It supports detection and classification covering bits in the DSCP, traffic class, flow label, destination address, source port, destination port, and ICMP checksum fields. Our implementation allows control of 10 parameters, including the confidence levels α_d and α_c, probing rate, timeouts, number of retries, output format, and halting conditions.

We also implement Route Explorer, a front-end for MCA measurements, which allows inspection probes and responses, and provides graphical visualizations integrating metadata such as IP-to-AS mapping and CAIDA's AS-rank [28], [29]. We make our tools and dataset (§VI) publicly available [30].

Table I: Dataset summary

Platform	VPs	Period	Number of traces IPv4	IPv6	ASes
UFMG	1	2018-08-21–2018-09-06	11040	13524	1217
Linode [38]	6	2018-08-21–2019-03-01	182676	166968	5858
Vultr [39]	6	2018-08-21–2019-03-01	211800	182460	6567
DOcean [40]	7	2018-08-21–2019-03-01	246984	222564	6601
Ark [41]	11	2018-08-21–2019-04-27	395388	325512	8051
All	31	2018-08-21–2019-04-27	1047888	911028	10075

VI. Dataset

We deploy MCA in 31 vantage points (VPs) in 6 platforms (cloud providers, monitoring testbeds, and one university) that provide IPv4 and IPv6 connectivity and that allow crafting and sniffing packets using Scapy [31]. The vantage points are spread across 16 countries in 5 continents. Table I summarizes our vantage points and dataset.

We run MCA toward a list of 19,866 IPv4 and 16,674 IPv6 addresses built by (i) resolving A and AAAA DNS records for domains on three Internet 'toplists' [32]–[34], and (ii) choosing representative addresses from ISI's IPv4 Internet census data from Sep. 2018 [35] and Gasser et al.'s IPv6 hitlist from Jan. 2019 [36]. Our IPv4 destinations are distributed in 4,388 ASes (61% in stub ASes), and our IPv6 destinations are in 8,103 ASes (50% in stub ASes). Destination ASes include all Tier-1 ASes and 95% of ASes with more than 500 indirect customers (as inferred by CAIDA [37]). We find the top 50 ASes with the most addresses are mostly large content and cloud infrastructure providers, and concentrate 42.2% of IPv4 and 20.7% of IPv6 destinations.

We run six measurement campaigns to cover all combinations of IP protocol (v4, v6) and transport (TCP, UDP, and ICMP). In each run, we set $\mathcal{F}_{\text{IPv4}} = \{$destination IP, destination port/ICMP checksum, DSCP$\}$ and $\mathcal{F}_{\text{IPv6}} = \{$destination IP, destination port/ICMP checksum, flow label, traffic class$\}$ to cover currently understood standard LB types [20] as well as more general but still reasonable hash domains for novelty.

We augment our MCA measurements with IP-to-AS mapping information from Team Cymru [28], reverse DNS entries (PTR records), and network types from PeeringDB [42].

VII. Impact of Optimizations on Probing Cost

MCA requires additional probes for classifying the LBs, which further increases MDA's high probing costs. In this section we evaluate this cost with and without the optimizations introduced in §IV-C. We show that our optimizations save a significant fraction of probes, particularly for classification and complex LB configurations, without loss of accuracy.

We define the probing cost as the total number of packets for detecting and classifying LBs. Probing cost is a function of route length, the degree of load balancing, and the sequence of random flow identifiers generated during the identification and classification phases.

The comparison of probe cost with and without optimizations is challenging experimentally. Rate limiting implies each measurement takes several seconds, and Internet paths may change over time. These factors combine to make it impractical to measure the same path thousands of times when estimating its average probing cost. Instead, we compute the average probing cost using trace-driven simulations. For each route measurement in our dataset, we simulate the MCA with and without optimizations until the 95% confidence interval around the average probing cost of each outermost diamond is below ± 1 probe. We hash flow identifiers using SHA-256.

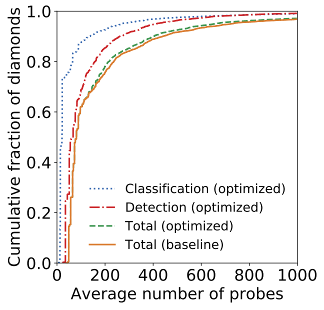


Figure 3: Probing cost for all outermost diamonds

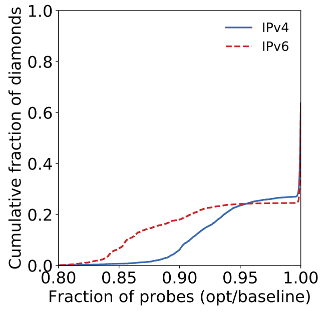


Figure 4: Normalized probing cost savings

Figure 3 shows the distribution of the average number of probes issued by MCA for all outermost diamonds in our dataset. The 'total (optimized)' line adds the cost of 'detection' and 'classification' with our optimizations applied. We first note that MCA's classification incurs a probing cost that is of the same order of magnitude as that of detection, and often significantly less, making MCA a practical addition to traceroute. When using our optimizations, classification amounts to 34% of the total probing cost of IPv4 measurements and to 39% of IPv6 measurements, on average.

Comparing the total probing cost with and without our optimizations (optimized *vs.* baseline) we observe savings that are small in this average view, with differences visible only around complex LBs that require a large number of probes. This is because the optimizations only apply to nested LBs, and incur more savings on complex diamonds, which also require more probes.

To enable the benefits to be better seen, Figure 4 shows the distribution of the total optimized MCA cost normalized by the number of probes sent by the unoptimized MCA (baseline) for IPv4 and IPv6 outermost diamonds. Our optimizations provide no savings for the 73% of IPv4 and 76% of IPv6 diamonds that have no nested LBs ($x = 1$). For the remaining diamonds with nested LBs, our optimizations provide savings around 5–15% (region between $0.85 \leq x \leq 0.95$).

Our optimizations provide only modest probe savings for detection with, on average, 2.8% less probes for IPv4 and 0.7% for IPv6. For classification, the savings are higher: reducing classification costs by 11% for IPv4 and 18% for IPv6 routes. Savings for classification for IPv6 are higher than for IPv4 because our MCA executions included four header fields for IPv6 and three for IPv4, and more fields means opportunities to apply optimization. In measurement campaigns covering more fields, savings would be higher than reported here. In total, we estimate by simulation that the optimizations reduce the total probing cost for routes in our datasets by 6% for

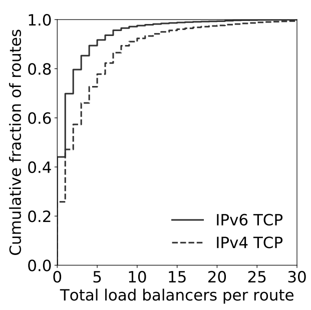


Figure 5: Number of load balancers per route

	IPv4			IPv6		
	UDP	TCP	ICMP	UDP	TCP	ICMP
Per-flow	69.6%	69.8%	1.5%	77.6%	78.5%	0.3%
Per-dest	24.4%	24.1%	94.2%	13.3%	13.5%	90.1%
Per-app	1.8%	2.1%	0.0%	0.9%	0.8%	0.0%
Per-packet	0.1%	0.1%	0.1%	0.1%	0.1%	0.3%
Per-flow + FL	—	—	—	2.9%	2.4%	0.0%
Per-dest + FL	—	—	—	0.2%	0.3%	3.2%
Other	2.3%	2.6%	2.7%	3.2%	2.8%	3.9%
Not classified	1.8%	1.4%	1.5%	1.7%	1.6%	2.2%
Total	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
Prevalence	74.9%	74.2%	72.9%	56.1%	55.8%	55.4%

Table II: Breakdown of load balancer classifications and prevalence by protocol

IPv4 and 8% for IPv6 for the default MCA configuration. Although our optimizations provide only single-digit savings, implementations would benefit from integrating them as they have *no* impact on accuracy and apply to *all* measurements.

VIII. Characterization of Load Balancing

We characterize IPv4 and IPv6 load balancing on the Internet. We study the prevalence of LBs (§VIII-A), identify common hash domains and classes of LB behavior (§VIII-B), and quantify routers overwriting packet header fields (§VIII-C). Finally, we revisit previous work and discuss the evolution of load balancing and path diversity (§VIII-D).

A. Occurrence of Load Balancing

We provide an overview of load balancing observed in our measurements. Linode, Digital Ocean, some networks hosting Ark nodes, and UFMG's provider employ LBs in their networks. All paths from these vantage points traverse at least one LB. Unless otherwise noted, we report results on load balancers identified *outside* the vantage point's network to avoid biasing results.

a) Load balancing is prevalent: Figure 5 shows the distribution of the number of LB interfaces on IPv4 and IPv6 routes. Even when ignoring load balancing in the vantage point's network, we find that 74% of IPv4 routes and 56% of IPv6 routes traverse at least one LB, and some traverse many more. When considering LBs in the origin network, we find that 86% of IPv4 routes traverse at least one LB, a fraction similar to that observed in 2009 [3], and that 77% of IPv6 routes traverse at least one LB, not significantly larger than the 74% observed in 2016 [20]. These results indicate the extent of traffic engineering used in the Internet and the need for considering LBs in Internet measurement efforts. Even though our measurements are collected from different vantage points to different destinations than measurements in previous work, and thus are not directly comparable, the observations are consistent and indicate prevalence of load balancers.

Our results miss load balancers that use fields not included in $\mathcal{F}$ and may misidentify path changes as load balancing. We expect these errors to have minor impact on the results: Table II indicates few load balancers employ nonstandard behavior (lines with "FL" and "Other"), and MCA measurements are fast compared to usual route durations [22].

b) IPv4 load balancing is more widespread: We find 55% of ASes traversed in our IPv4 measurements employ IPv4 load balancing; compared to 41% in IPv6 measurements. Moreover, although 58% of ASes that appear in both IPv4 and IPv6 measurements employ both IPv4 and IPv6 load balancing, ASes more often employ IPv4 load balancing exclusively. We find that 82% of the ASes that employ IPv6 load balancing and are traversed in IPv4 measurements employ IPv4 load balancing, but that only 66% of the ASes that employ IPv4 load balancers and are traversed in IPv6 measurements employ IPv6 load balancing. These results indicate richer traffic engineering in IPv4 than in IPv6, possibly due to IPv4 still carrying most traffic or simply due to being around (and engineered) for longer.

We find that load balancing is applied similarly across all transport protocols. The 'prevalence' row in Table II shows the fraction of routes in the dataset with load balancing, and that it does not depend on transport protocols.

B. Classes of Load Balancing Behavior

As a measure of the accuracy of our classifications, we check whether multiple measurements of the same LB identify identical hash domains. When using TCP probes, we find that 9.7% of IPv4 and 8.8% of IPv6 LBs are measured multiple times (this ratio is similar across TCP, UDP, and ICMP), and we infer that 98.1% of these LBs have the same hash domain, which is within our 95% confidence threshold.

Table II shows the percentage of LBs of each class per protocol for each of the most common types of load balancing (§III) in Internet routes, ignoring LBs in the vantage point's network. As explained before, load balancing is more prevalent on IPv4 routes and there is no significant difference in prevalence between transport protocols. We observe that per-flow LBs are the most common, followed by per-destination. We observe a few LBs considering only transport port numbers in their hash domain (we found this can be configured in RouterOS as "per-application load balancing" [43]) and some IPv6 load balancers employing the flow label field [44].

We note that existing MDA implementations would correctly identify and classify only per-flow, per-destination, and per-packet load balancers. In particular, existing implementations would misclassify per-application load balancers as per-flow, and entirely miss load balancing using the DSCP, traffic class, and flow label fields.

a) Load balancing behavior is improving: Per-packet load balancing is at an all-time low relative to previous characterizations. As per-packet LBs induce packet reordering and TCP performance degradation, this is a positive trend.

We find few LBs whose hash domains include the ICMP checksum field (shown as per-flow in Table II). Augustin et al. [3] reported a considerable decrease in the number of routers performing per-flow load balancing for ICMP between 2006 and 2009; we find this trend has continued, with an all-time low of IPv4 LBs considering the ICMP checksum field (3.2% of load balancers, down from 20% in 2009). This behavior may follow from more mature implementations defaulting to per-destination load balancing for ICMP packets.

We find that an average of 2.8% of IPv6 LBs consider the flow label field in their hash domain, when measuring with TCP. We also observe the use of flow label in conjunction with per-flow or per-destination load balancing classes: we classify 2.7% of IPv6 LBs as having this behavior. A significant fraction (85%) of LBs including the flow label in their hash domains are in content and infrastructure providers' networks (e.g. Facebook's AS32934 and Google's AS15169), which adopt IPv6 and rely on advanced traffic engineering [15], [16].

b) Traffic engineering triggers load balancing: IPv6's traffic class and IPv4's DSCP fields serve a similar purpose of classifying packets. Although their use is not widespread we identified a non-negligible number of LBs with the IPv6 traffic class (2.7%) and IPv4 DSCP fields (2.6%) in their hash domains when measuring with TCP (the default in JunOS [44]).

c) Classification errors: When measuring with TCP, MCA failed to classify 1.4% IPv4 and 1.6% IPv6 load balancers; i.e., MCA detected load balancing, but did not observe load balancing during the classification phase. Possible reasons for this include measurement errors (possibly aggravated by non-uniform load balancing) or hash domains that only trigger load balancing when multiple fields are varied simultaneously.

C. Overriding of Packet Header Fields

One challenge when trying to identify and classify load balancers whose hash domains include the DSCP, traffic class, and flow label fields is that these fields may be overwritten by intermediate routers and middleboxes [45]. Such overwriting will interfere with identification and classification of load balancers by preventing control of the field's value.

We recover the values of the DSCP, traffic class, and flow label fields received by each interface in a multi-route from the probe headers encapsulated in router ICMP time-exceeded responses. When we find an interface in a route that receives a DSCP value, traffic class, or flow label field with a value different from the *expected* value, we infer that the preceding interface overwrites that field. For any overwritten field, we identify whether it is overwritten with a fixed or variable value by looking at multiple encapsulated probe headers. Note that the *expected* values for DSCP, traffic class, and flow label fields change after an interface that overwrites them with a fixed value and become undetermined (which we consider as a special value) after an interface that overwrites with a variable value. We identify interface overwriting behaviors proceeding hop-by-hop starting from the vantage point.

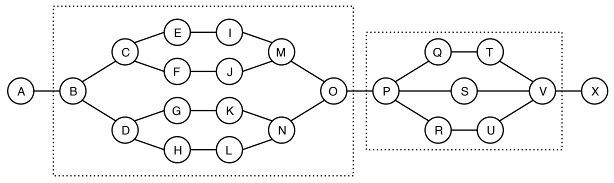


Figure 6: An example multi-route with two diamonds.

We find that 4.1% of IPv4 interfaces overwrite the DSCP field (0.7% with a variable value) and that 3.6% of IPv6 interfaces overwrite the traffic class field (0.7% with a variable value). The 3.4% of interfaces that overwrite the DSCP and 2.9% that overwrite the traffic class field with a fixed value prevent identification and classification of subsequent load balancing including these fields in their hash domains: This leads to an underestimation of load balancers using the DSCP and traffic class fields reported in Table II. MCA tracks interfaces that overwrite fields with variable values to avoid incorrectly inferring per-packet load balancing if subsequent load balancers include DSCP and traffic class in their hash domains. We did not find a significant number of interfaces overriding the IPv6 flow label field.

D. Diamonds and Branched Route Properties

In this section we characterize LB outermost diamonds (called simply *diamonds* in this section) in Internet routes in an attempt to better understand how LBs are used. Overall, we find that IPv4 diamonds are more complex, traversing a higher number of LBs and resulting in greater path diversity.

We revisit Augustin et al.'s original diamond metrics and consider new ones [3]. Figure 6 shows a route with two diamonds. Augustin et al. [3] defined the *length* as the number of edges in the longest branch across the diamond; the *asymmetry* as the maximum length difference between any sequence of interfaces across the diamond (a symmetric diamond has zero asymmetry); the *min-width* of a diamond as the number of edge-disjoint sequences of interfaces across the diamond; and the *max-width* as the maximum cardinality of any hop. The min- and max-width give lower and upper bounds on route diversity. We define the *depth* of a diamond as the maximum number of LBs traversed by any of its branches. In Figure 6, diamond B–O is symmetric, has length 5, min-width 2, max-width 4, and depth 2; diamond P–V has asymmetry 1, length 3, min-width 3, max-width 3, and depth 1. We have compiled a list of illustrative branched route measurements, which can be interactively inspected in Route Explorer [30].

a) Diamonds are similar across transport protocols: We do not observe significant differences between diamonds measured with TCP, UDP, or ICMP (not shown). In this section we report on diamonds measured using TCP probes. For most metrics, we observe no significant differences between diamonds measured using IPv4 or IPv6, or between diamonds from LBs with different hash domains; in the following paragraphs we point out the significant differences we identified.

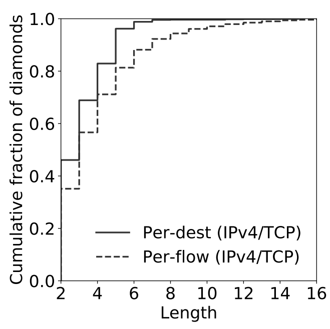


Figure 7: Diamond Length

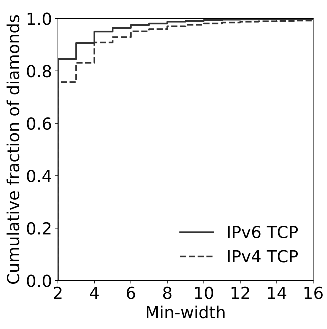


Figure 8: Diamond min-width

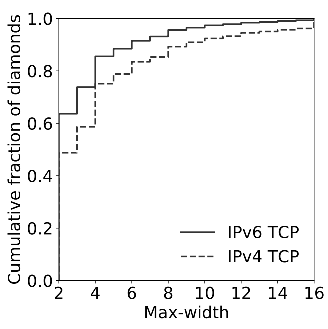


Figure 9: Diamond max-width

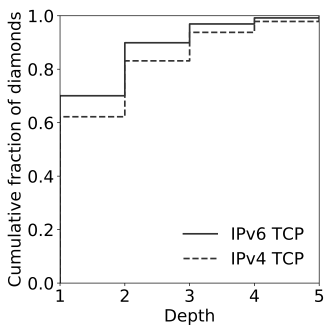


Figure 10: Diamond depth

b) Diamond length and asymmetry: Figure 7 shows the distribution of diamond lengths for per-flow and per-destination load balancers. We find per-destination diamonds to be shorter than per-flow diamonds, as previously observed by Augustin et al. [3]. We also find that diamonds are usually short (80% of diamonds span at most 5 hops) and that IPv6 diamonds are shorter than IPv4 diamonds (not shown).

We observe asymmetry to be rare and small for both IPv4 and IPv6 (83% of the diamonds are symmetric), following previous results on diamond asymmetry [3] (not shown).

c) Diamond width and route diversity: Figures 8 and 9 show the distributions of min-width and max-width for diamonds in our dataset. We observe the min-width to be small, with 86% of diamonds having only two edge-disjoint branches. As previously observed by Augustin et al. [3], we find that diamonds are narrow, i.e., their max-width is small, and that IPv4 diamonds are slightly wider than IPv6 diamonds. We also observe that wide and long IPv6 diamonds are considerably less common than in IPv4.

The majority of LBs (80% for IPv4 and 82% for IPv6) have two next hops (not shown). We find less than 1% of LBs have more than 16 next hops. This may be a limitation of router implementations; e.g., Juniper's JunOS limits routers performing ECMP to 16 next hops [44].

d) Number of LBs and diamond depth: We find that diamonds usually have few LBs, but a few diamonds have more than 20. Diamonds often have between 1 and 3 LBs, which often appear in a symmetric fan-out pattern as that of routers B, C, and D in Figure 6.

Figure 10 shows the distribution of diamond depth in our dataset. IPv4 diamonds have slightly more LBs and higher depth than IPv6 diamonds. Most diamonds have depth 1 or 2. We also find that the difference between min- and max-width is correlated with the diamond depth: 86% of the diamonds with a max-width of 8 or larger have a depth of 2 or more.

IX. Related Work

Traceroute applications and limitations: Network administrators use Internet route measurements to troubleshoot failures [46], routing anomalies [47]–[49], bad performance [50], and misconfigurations [51]. Researchers study properties of Internet routes to propose new models and develop solutions to address limitations [37], [52]–[55]. Previous work has shown that traceroute measurements may be incomplete, e.g., due to routers that rate-limit responses or that fail to respond to probes with ICMP Time Exceeded messages; that do not decrement the TTL field of packets; that encapsulate packets using MPLS; or that respond to probes using off-path interfaces [3], [56]–[58]. Load balancing, which our work improves on, is another source of artifacts in traceroute measurements.
Load balancing measurement and characterization: MDA was first conceived for the IPv4 Internet. IPv6 implementations include Scamper [18] and a new version of Paris traceroute. These implementations directly adapt MDA from IPv4 to IPv6, with the same underlying assumptions. Although these assumptions often hold (§VIII-B), we quantify how often and, more importantly, provide a principled approach to more accurately identify and classify both IPv4 and IPv6 LBs. Previous work used MDA to characterize load balancing on the IPv4 [3], [19] and IPv6 Internet [20]. In this work, we use MCA to revisit these results and show how load balancing evolved over the years. More recently, MDA-lite [21] proposed lossy optimizations to significantly reduce the probing cost for identifying LBs. In this work, we provide complementary optimizations that further reduce probing cost without loss of accuracy.

X. Conclusion

In this paper we presented a more general model for load balancing that considers that load balancers can use arbitrary bits in packet header fields for load balancing. We designed and implemented MCA, a probing algorithm that identifies and classifies load balancing, alongside optimizations to reduce its overall probing cost. MCA is practical, increasing overall probing cost on top of existing techniques by only 34%, while providing richer information.

We collected a large dataset to characterize load balancing practices for all combinations of IP protocol version (v4 and v6) and transport protocol (UDP, TCP, and ICMP). Our results show load balancing is more prevalent and load balancing strategies more mature than previously reported. We identified that existing measurement tools cannot identify 4.7% of load balancers, and will misclassify an additional 2.3%. Given the rise of IPv6 traffic [14], [59], programmable data planes [9], [10], and software-defined networking [7], [12], these percentages may increase and current tools become increasingly inadequate over time.

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 *IMC*, 2006.

[2] V. Jacobson, “traceroute,” 1989.

[3] B. Augustin, T. Friedman, and R. Teixeira, “Measuring Multipath Routing in the Internet,” *IEEE/ACM Trans. Netw.*, vol. 19, no. 3, pp. 830–840, 2011.

[4] C. Pelsser, L. Cittadini, S. Vissicchio, and R. Bush, “From Paris to Tokyo: On the Suitability of Ping to Measure Latency,” in *IMC*, 2013.

[5] F. Viger, B. Augustin, X. Cuvellier, C. Magnien, M. Latapy, T. Friedman, and R. Teixeira, “Detection, Understanding, and Prevention of Traceroute Measurement Artifacts,” *Comput. Netw.*, vol. 52, no. 5, pp. 998–1018, 2008.

[6] D. Veitch, B. Augustin, T. Friedman, and R. Teixeira, “Failure Control in Multipath Route Tracing,” in *INFOCOM*, 2009.

[7] A. Gupta, R. MacDavid, R. Birkner, M. Canini, N. Feamster, J. Rexford, and L. Vanbever, “An Industrial-scale Software Defined Internet Exchange Point,” in *NSDI*, 2016.

[8] N. Kang, M. Ghobadi, J. Reumann, A. Shraer, and J. Rexford, “Efficient Traffic Splitting on Commodity Switches,” in *CoNEXT*, 2015.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, “P4: Programming Protocol-independent Packet Processors,” *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, 2014.

[10] A. Sivaraman, A. Cheung, M. Budiu, C. Kim, M. Alizadeh, H. Balakrishnan, G. Varghese, N. McKeown, and S. Licking, “Packet Transactions: High-Level Programming for Line-Rate Switches,” in *SIGCOMM*, 2016.

[11] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, “BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing,” in *SIGCOMM*, 2015.

[12] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, “B4: Experience with a Globally-deployed Software Defined Wan,” in *SIGCOMM*, 2013.

[13] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, “Achieving High Utilization with Software-driven WAN,” in *SIGCOMM*, 2013.

[14] Google, “IPv6 Adoption Statistics by Google,” 2019.

[15] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, “Engineering Egress with Edge Fabric: Steering Oceans of Content to the World,” in *SIGCOMM*, 2017.

[16] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, V. Lin, C. Rice, B. Rogan, A. Singh, B. Tanaka, M. Verma, P. Sood, M. Tariq, M. Tierney, D. Trumic, V. Valancius, C. Ying, M. Kallahalla, B. Koley, and A. Vahdat, “Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering,” in *SIGCOMM*, 2017.

[17] Z. Zhang, M. Zhang, A. G. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, “Optimizing Cost and Performance in Online Service Provider Networks,” in *NSDI*, 2010.

[18] M. Luckie, “Scamper: a Scalable and Extensible Packet Prober for Active Measurement of the Internet,” in *IMC*, 2010.

[19] B. Augustin, T. Friedman, and R. Teixeira, “Measuring load-balanced paths in the internet,” in *IMC*, 2007.

[20] R. Almeida, O. Fonseca, E. Fazzion, D. Guedes, W. Meira, and Í. Cunha, “A Characterization of Load Balancing on the IPv6 Internet,” in *Proc. PAM*, 2017.

[21] K. Vermeulen, S. D. Strowes, O. Fourmaux, and T. Friedman, “Multilevel MDA-Lite Paris Traceroute,” in *IMC*, 2018.

[22] I. Cunha, R. Teixeira, D. Veitch, and C. Diot, “DTRACK: A System to Predict and Track Internet Path Changes,” *IEEE/ACM Trans. Netw.*, vol. 22, no. 4, pp. 1025–1038, 2014.

[23] S. Amante, B. Carpenter, S. Jiang, and J. Rajahalme, “RFC 6437 - IPv6 Flow Label Specification,” 2011.

[24] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, “WCMP: Weighted Cost Multipathing for Improved Fairness in Data Centers,” in *Proc. ACM EuroSys*, 2014.

[25] B. Donnet, M. Luckie, P. Mérindol, and J. Pansiot, “Revealing MPLS Tunnels Obscured from Traceroute,” *SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 2, pp. 87–93, 2012.

[26] Y. Vanaubel, P. Mérindol, J. Pansiot, and B. Donnet, “A Brief History of MPLS Usage in IPv6,” in *Proc. PAM*, 2016.

[27] R. Beverly, A. Berger, and G. Xie, “Primitives for Active Internet Topology Mapping: Toward High-Frequency Characterization,” in *IMC*, 2010.

[28] Team Cymru, “IP to ASN Mapping,” 2019.

[29] CAIDA, “AS Rank,” 2019. [Online]. Available: http://as-rank.caida.org/

[30] R. Almeida, I. Cunha, R. Teixeira, D. Veitch, and C. Diot, “Multipath Classification Algorithm (MCA) Code and Dataset,” 2019. [Online]. Available: https://www.dcc.ufmg.br/~rlca/mca/

[31] P. Biondi, “Scapy Documentation.” [Online]. Available: https://scapy.net

[32] Alexa, “Top Global Sites Ranking,” 2019.

[33] Cisco, “Umbrella Popularity List,” 2019.

[34] Majestic, “The Majestic Million,” 2019.

[35] X. Fan and J. Heidemann, “Selecting Representative IP Addresses for Internet Topology Studies,” in *SIGCOMM*, 2010.

[36] O. Gasser, Q. Scheitle, P. Foremski, Q. Lone, M. Korczynski, S. D. Strowes, L. Hendriks, and G. Carle, “Clusters in the Expanse: Understanding and Unbiasing IPv6 Hitlists,” in *IMC*, 2018.

[37] M. Luckie, B. Huffaker, K. Claffy, A. Dhamdhere, and V. Giotsas, “AS Relationships, Customer Cones, and Validation,” in *IMC*, 2013.

[38] Linode. [Online]. Available: https://www.linode.com/

[39] Vultr. [Online]. Available: https://www.vultr.com/

[40] DigitalOcean. [Online]. Available: https://www.digitalocean.com/

[41] CAIDA, “Archipelago (Ark) Measurement Infrastructure,” 2019.

[42] PeeringDB. [Online]. Available: https://www.peeringdb.com

[43] Mikrotik Wiki, “ECMP Load Balancing With Masquerade,” 2016.

[44] Juniper, “Configuring Per-Packet Load Balancing.” [Online]. Available: https://www.juniper.net/documentation/en_US/junos/topics/usage-guidelines/policy-configuring-per-packet-load-balancing.html

[45] J. Jaeggli, “IPv6 Flow Label: Misuse in Hashing,” RIPE Labs Blog.

[46] E. Katz-Bassett, C. Scott, D. R. Choffnes, I. Cunha, V. Valancius, N. Feamster, H. V. Madhyastha, T. Anderson, and A. Krishnamurthy, “LIFEGUARD: Practical Repair of Persistent Route Failures,” in *SIGCOMM*, 2012.

[47] A. Lakhina, M. Crovella, and C. Diot, “Diagnosing Network-wide Traffic Anomalies,” in *SIGCOMM*, 2004.

[48] K. V. M. Naidu, D. Panigrahi, and R. Rastogi, “Detecting Anomalies Using End-to-End Path Measurements,” in *INFOCOM*, 2008.

[49] F. Silveira, C. Diot, N. Taft, and R. Govindan, “ASTUTE: Detecting a Different Class of Traffic Anomalies,” in *SIGCOMM*, 2010.

[50] P. Barford, N. Duffield, A. Ron, and J. Sommers, “Network Performance Anomaly Detection and Localization,” in *INFOCOM*, 2009.

[51] F. Le, S. Lee, T. Wong, H. S. Kim, and D. Newcomb, “Minerals: Using Data Mining to Detect Router Misconfigurations,” in *Proc. of MineNet*, 2006.

[52] V. Giotsas, M. Luckie, B. Huffaker, and K. Claffy, “IPv6 AS relationships, cliques, and congruence,” in *Proc. PAM*, 2015.

[53] V. Giotsas, M. Luckie, B. Huffaker, and kc claffy, “Inferring Complex AS Relationships,” in *IMC*, 2014.

[54] Í. Cunha, P. Marchetta, M. Calder, Y. Chiu, B. Schlinker, B. Machado, A. Pescape, V. Giotsas, H. Madhyastha, and E. Katz-Bassett, “Sibyl: A Practical Internet Route Oracle,” in *NSDI*, 2016.

[55] H. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani, “iPlane: an Information Plane for Distributed Services,” in *OSDI*, 2006.

[56] P. Marchetta, A. Montieri, V. Persico, A. Pescapé, Í. Cunha, and E. Katz-Bassett, “How and How Much Traceroute Confuses Our Understanding of Network Paths,” in *Proc. LANMAN*, 2016.

[57] M. Luckie, Y. Hyun, and B. Huffaker, “Traceroute Probe Method and Forward IP Path Inference,” in *IMC*, 2008.

[58] R. Beverly, R. Durairajan, D. Plonka, and J. Rohrer, “In the IP of the Beholder: Strategies for Active IPv6 Topology Discovery ,” in *IMC*, 2018.

[59] Facebook, “IPv6 Traffic Statistics by Facebook,” 2019.