PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks

Thomas Koch
Columbia University

Shuyue Yu
Columbia University

Sharad Agarwal
Microsoft

Ryan Beckett
Microsoft

Ethan Katz-Bassett
Columbia University

ABSTRACT

Enterprises increasingly use public cloud services for critical business needs. However, Internet protocols force clouds to contend with a lack of control, reducing the speed at which clouds can respond to network problems, the range of solutions they can provide, and deployment resilience. To overcome this limitation, we present PAINTER, a system that takes control over which ingress routes are available and which are chosen to the cloud by leveraging edge proxies. PAINTER efficiently advertises BGP prefixes, exposing more concurrent routes than existing solutions to improve latency and resilience. Compared to existing solutions, PAINTER reduces path inflation by 75% while using a third of the prefixes of other solutions, avoids 20% more path failures, and chooses ingresses from the edge at finer time (RTT) and traffic (per-flow) granularities, enhancing our agility.

CCS CONCEPTS

• **Networks** → *Network reliability*; **Network experimentation**;

KEYWORDS

Anycast, enterprise cloud, path inflation, routing.

ACM Reference Format:
Thomas Koch, Shuyue Yu, Sharad Agarwal, Ryan Beckett, and Ethan Katz-Bassett. 2023. PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks. In *ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23), September 10, 2023, New York, NY, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3603269.3604868

1 INTRODUCTION

Everyday business needs that used to run on corporate LANs now rely on the public Internet, as more businesses depend on the cloud for services. However, these businesses continue to be plagued by routing problems and performance bottlenecks anywhere between clouds, enterprise users, and intermediate ASes. Most of us can recall the annoyance we feel when a tele-meeting is disrupted by poor network performance, but now imagine how that occurrence impacts an entire business that is paying to run many or all its services on the cloud. The impact of such problems will grow, as the Networking-as-a-Service market is projected to be a $60B industry by 2027 [50]. These are urgent problems clouds must solve for current services, and more critical problems clouds will face as they offer applications with stricter performance requirements—augmented reality requires 10 ms latency at 20 Mbps with a 10^{-6} packet-loss rate [1]. Delivering 5G's promise of ultra-reliable-low-latency communication and the Gbps data rates that 5G supports will similarly place more pressure on clouds [78].

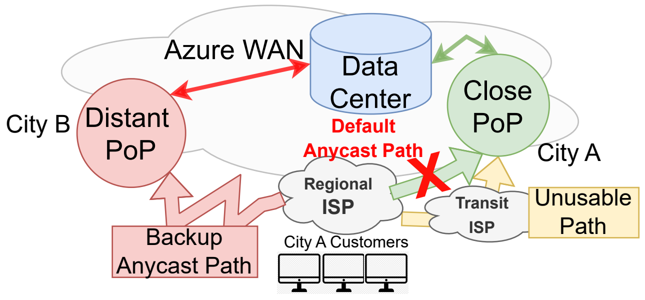


Figure 1: A difficult customer problem to avoid.

For example, investigation of a recent cloud customer performance issue uncovered that traffic from one of their branch offices in City A took an unusually circuitous anycast path to land at a distant Azure PoP in City B because one of City A's regional ISP's peering routers failed. Despite the possibility of a policy-compliant path to City A's PoP through another ISP (Transit ISP), there was no mechanism for detecting such paths and re-directing customer traffic (see Figure 1). Hence Figure 1 labels this path as 'Unusable'. Fiddling with route policies and weights to resolve this problem remained a risky and slow process. This dynamic failure can be difficult to mitigate and could lead to severe performance problems for vital customer services.

These problems are hard to avoid because current solutions force clouds to choose between availability and performance. Clouds use anycast for availability, but may sometimes use unicast for performance since anycast can inflate paths [21, 54]. Unicast routing is enabled by DNS which directs users to different PoPs, but at coarse granularities (per-recursive resolver), cannot select among different paths to a single PoP, and cannot react quickly during failure due to clients/recursives not respecting DNS TTLs [16, 35, 60, 73]. All these limitations of DNS hurt availability. BGP limits clouds further since it may similarly pick poorly performing paths, it only exposes one path, and failover between paths is slow [57].

To better equip clouds with tools for offering performant services, we designed a system—PAINTER (Precise, Agile INgress Traffic Engineering & Routing)—that provides a framework for routing user ingress traffic with precision. PAINTER mitigates network problems such as path inflation and congestion by intelligently and efficiently exposing and precisely selecting paths to the cloud.

PAINTER exposes paths to select from by advertising multiple prefixes via different subsets of cloud peerings, but it cannot expose all possible paths since prefix advertisements are expensive and may pollute BGP routing tables. Instead, PAINTER computes efficient prefix to peering allocation strategies that enhance performance (*e.g.,* by eliminating path inflation) and enhance resilience (*e.g.,* by providing backup paths). PAINTER limits its impact on BGP routing tables through *prefix reuse*—advertising the same prefix via multiple peerings if PAINTER predicts it will not inflate paths—and learns better strategies over time.

PAINTER selective advertisements expose the potential for better performance and reliability, but that potential alone is not enough. Customer traffic needs to be steered onto these better paths, at a fine enough granularity to allow each flow to utilize the best path for it, but current solutions do not suffice to take advantage of these new opportunities.

Our insight to solve this problem is to leverage clouds increasing access to powerful networking capabilities at the edge of the Internet. These powerful capabilities include MPTCP proxies [98], cloud-edge network stacks [5, 41, 44, 68], mobile apps with full stack control [8, 104], and integrated VPNs such as Apple Private Relay [48, 84]. Since clouds have invested in extending their reach into edge networks, there is an opportunity for clouds to enact fine-grained control over traffic at or close to the client, including steering traffic onto paths exposed by PAINTER. PAINTER is therefore the combination of path exposure with the use of edge presence to select among paths at fine granularities.

In this paper we focus on one particular type of edge presence to exert fine-grained control, cloud-edge network stacks, which are software stacks in enterprise networks that enable cloud-based networking solutions. For example, major clouds offer direct integration with on-premise network management devices [5, 41, 44, 68]. We found this type of edge presence to be the most deployable, most powerful solution for clouds like Azure (more details in Section 5.2.1). However, PAINTER could use other edge presences such as MPTCP-enabled clients [45] (§3.2).

In summary, PAINTER's design provides two key contributions. First, it provides a path exposure framework which (when paired with traffic steering) mitigates network problems such as path inflation and congestion; and second, it provides a practical deployment strategy—situating it in cloud-edge network stacks—which lets clouds precisely steer traffic over paths on a per-flow basis. These contributions independently provide improvements over current best practices and together provide more benefit than either of them alone.

We demonstrate the utility of PAINTER using both prototypes and measurement-driven evaluations that quantify PAINTER's benefits compared to other solutions. Measurements are from hundreds of thousands of user networks to two global cloud deployments, each of which has thousands of peerings. We measure from Azure and RIPE Atlas [93], and build a prototype using the PEERING testbed [85], which is now deployed at 25 Vultr cloud locations [103]. Vultr is a global public cloud that allows us to issue BGP advertisements to its peers/providers.

We show that PAINTER's advertisement strategies offer persistent latency improvements to users and use fewer prefixes (less routing table impact) than other solutions (§5.1), which stem from its intelligent decisions about which peers/providers to advertise prefixes to and when to reuse prefixes. We demonstrate that PAINTER's advertisement strategies can reduce path inflation by 60 ms on average for thousands of networks, and that these strategies maintain these benefits for at least a month. PAINTER intelligently improves its strategies over time, by observing how clients route to deployments. PAINTER's advertisement strategies expose more paths to the cloud than alternate solutions such as SD-WAN with multihoming (at least 23 for most networks), and so offer more resilience to 20% more path failures (§5.2.4).

We show PAINTER's steering mechanism is far more deployable than other solutions (§5.2) and that this mechanism steers traffic at finer (per-flow) granularities than other steering mechanisms (DNS/BGP updates) [23, 82]. Even using the finest control knobs available, these other steering solutions shift traffic at coarse granularities that negate half the benefits of PAINTER's selective advertisements (§5.2.2). Our prototype on the PEERING testbed [85] demonstrates a helpful use case of PAINTER that these other approaches fail to address—failover at RTT-timescales (§5.2.3).

PAINTER couples cloud-side intelligent advertisements to expose diverse, high-performance paths with client-side fine-grained selection from these paths to practically achieve more control over routing than has traditionally been possible in the interdomain setting. Whereas optimal routes can2 be computed, installed, and selected directly in single-domain settings, the interdomain setting traditionally divided decisions among distinct entities, each lacking global visibility and together lacking coordination, limiting solutions. Having this enhanced control over more legs of the routing decision is imperative to offering richer networked applications [72, 78]. We see PAINTER as the first in an approaching wave of systems that uses enhanced traffic control in the interdomain setting to provide the high-performance, Internet-scale systems of tomorrow.

Ethics. We use anonymized traces from residential buildings managed by Columbia University to motivate our system. Before data collection, our data collection protocol underwent formal Institutional Review Board (IRB) review and was declared exempt as non-human-subjects research. It was reviewed and approved by the university IT department's security and privacy team and networking team. We follow established practices to anonymize all PII and securely store data [53]. More information about this process is in Appendix A. This work raises no other ethical issues.

2 MOTIVATION AND CHALLENGES

2.1 Modern Enterprises, Old Protocols

Even though paths to the cloud are often low latency [29, 72] and near-optimal [21, 54, 56], occasional networking problems still affect critical enterprise operations (*e.g.,* Fig. 1). Such problems are not new, but two trends make them increasingly salient for clouds offering enterprise products (*i.e., enterprise clouds*).

First, critical business needs that would traditionally be on-premises such as network management and file storage are increasingly outsourced to enterprise clouds. The virtual WAN in Figure 2 demonstrates the degree to which the modern enterprise can be integrated with the cloud. This enterprise has a virtual corporate WAN, which uses the cloud's connectivity, physical infrastructure, and security to connect regional branch offices, HQ, and remote employees to each other. On top of this networked structure, the enterprise can use cloud services such as teleconferencing to have meetings or distributed databases to track sales. Management points such as

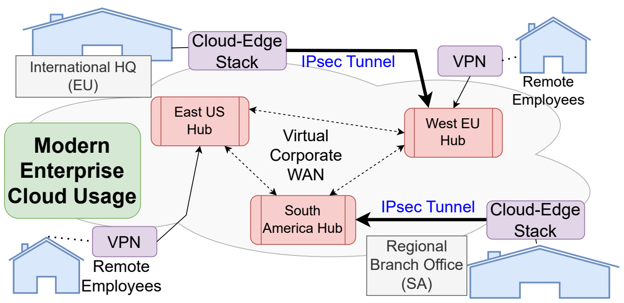


Figure 2: Modern enterprise integration with the cloud.

VPN middleboxes and SD-WANs running cloud network stacks act as traffic choke points where operators can enact policy on traffic.

Second, trends in application usage suggest ingress (*i.e.,* towards the cloud) traffic may impact user experience more than before. The rise in video conferencing traffic during the pandemic observed both in the literature [33, 63] and in Azure's traffic logs has already made this trend clear. Moreover, new 5G applications have the potential to drive massive amounts of traffic to the cloud with tighter performance requirements [72, 78]. For example, US mobile providers actively collaborate with clouds to provide 5G network functions [12, 42, 69, 77, 91]. These application workloads differ greatly from traditional web traffic which was cacheable and heavily biased towards egress. Collectively, these trends mean that paths to the cloud must meet increasingly strict performance requirements.

Faced with needing to meet tighter performance guarantees, clouds find themselves severely limited by how traffic is mapped onto Internet paths. First, cloud services are mapped to hostnames, effectively outsourcing traffic management to client DNS resolvers and caches. Resolvers and caches map to IP addresses via the DNS protocol, limiting cloud traffic management to the granularity of hostname, client recursive resolver, and DNS record TTL (or even worse—see Section 2.2). Finally, IP addresses are mapped to Internet paths via BGP which is not performance-aware and is slow to reconverge after failure [116]. These factors combine to form a mapping process that is coarse, slow to react, and often not performant.

2.2 Insufficiencies of Existing Techniques

IP Anycast and/or DNS. IP anycast is an approach to routing where distinct PoPs all advertise the same IP prefixes. This strategy is used by clouds and CDNs for its simplicity and resilience [21, 25, 40, 74, 79, 101]. Anycast offers limited control over paths, leading to path inflation or unpredictable mappings from clients to PoPs (as evidenced in our aforementioned customer issues) [21, 54, 59, 64], so some deployments use tailored DNS records to expose more paths and return a specific DNS record to recursive resolvers to send users to an optimal PoP [23, 76, 87].

However, using this DNS deployment either separate from or on top of an anycast deployment [21, 66, 111] to expose more paths leads to *reliability* problems. Recursive resolvers serve large populations of users who may benefit from finer redirection [23], and DNS records are hard to update quickly since records live in caches for a long time. Prior work demonstrates that many flows start after the corresponding DNS record has expired [16, 35, 60, 73], leaving these flows outside the control of the cloud to steer based on performance, availability, or load-balancing. However, the situation is even bleaker than this work suggests. Even flows that start while a DNS record is valid may be extant after DNS expiration, further restricting cloud control over traffic.

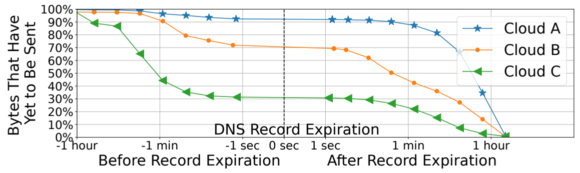


Figure 3: Of all traffic sent to Cloud A, 80% is sent at least 5 minutes after TTL expiration.

Our analysis captures the impact of flows that send traffic after the DNS record has expired. We find that *most* traffic to some clouds is sent to addresses from expired DNS records. Our analysis used anonymized residential traffic and matched observed DNS queries to observed flows as in prior work [4]. More about our DNS measurement and analysis pipeline are in Appendix A.

Figure 3 shows that, out of all the traffic sent to one of the three largest clouds, 80% still uses the old DNS record five minutes after TTL expiration, whereas routes change and failures occur at second timescales. Although less extreme, 20% of traffic is sent to the other two clouds at least a minute after the DNS record directing it expires. Traffic continues to use IP addresses from expired DNS records both because flows live past record expiration, and because clients cache the IP addresses and start new flows after the TTLs expire (we observed roughly a 2:1 ratio, respectively).

Hence, clouds have limited ability to quickly update paths that traffic takes to their networks in the face of sudden performance changes or failures.

Existing Commercial Products. Several companies have products targeted at enterprise networks that offer improved network performance [2, 27, 49, 96], which may use a combination of overlay routing (with PoPs/datacenters as overlay nodes) and multihoming (if enterprises have multiple ISPs). In particular, SD-WAN devices can select among a multihomed enterprises' different ISPs to optimize latency [90] which, by itself, does not solve our problem and has been around for decades [3]. We compare our proposal to an SD-WAN multihoming solution in Section 5.2.4. Products that use overlay routing cannot use different paths between users and the cloud since, in those products, variation comes from choosing existing paths through cloud WANs. Those products can therefore solve a different set of problems such as optimizing existing routes between branch offices. Other companies such as Megaport [67] claim to offer better routes between enterprises and the cloud, but enterprises still use DNS/BGP to reach these third parties and thus this solution suffers from the limitations of those protocols.

2.3 Opportunity: Cloud Control at the Edge

Current solutions to directing traffic suffer from a lack of control (§2.2), but new deployment trends could expose solutions that offer clouds precise traffic control.

Internet practitioners have recognized that fine-grained traffic control capabilities help offer richer network support and features. Traditionally, these capabilities have been implemented through traffic control points in or near the source. For example, operators can use SD-WAN to enact policy on corporate traffic and ensure the network security of corporate WANs.

The difference today is that clouds have extended their presence into the edge, offering them more access to these and other new control points—networking-as-a-service devices such as SD-WAN

[14, 102], cloud-edge network stacks [5, 41, 44, 68], recent OS versions that give application developers more control over the network stack (*e.g.,* MPTCP/MPQUIC, the Skype app) [7, 8, 11, 31, 47, 51, 98, 104, 114], and integrated VPNs such as Apple Private Relay [48, 84]. We refer to all these technologies as *edge proxies*, as they all allow clouds to exert more direct control over traffic.

For our setting (enterprise cloud) in particular, cloud-edge network stacks offer a uniquely powerful traffic control point. These control points may exist on customer-owned `SD-WAN` devices or devices provided by the cloud, are physically inside enterprise premises, are managed by enterprises, and enact networking policies on user traffic. Our insight is that clouds should extend their reach via cloud-edge network stacks since they are designed to enact policy on traffic, since these stacks already use software frameworks integrated with the cloud [5, 41, 44, 68], and since both parties (enterprise and cloud) would benefit from this synergy.

2.4 Key Challenges

Realizing a solution that overcomes the limitations of current protocols (§2.1) comes with three key challenges.

It is hard to deploy solutions. To overcome the limitations of BGP and DNS, we require fine-grained traffic control. The Path-Aware Networking research group (PAN-RG) [46] has characterized sets of networking solutions that offer fine-grained traffic control. The working group identified the key requirements to *deploying* successful intelligent routing solutions as requiring no major changes to ISP operations, working with all network hardware, being immediately partially deployable, and providing incentives for both operator and innovator.

That these are formidable challenges to overcome is evidenced by a lack of widely deployed solutions to the problem. Others have recognized the limitations of BGP and DNS and proposed various solutions [7, 55, 59, 80, 104, 108, 109, 114], but these solutions are not widely deployed.

We cannot make every route available. Making every route available to clients would let them choose the best routes, improving performance. However, BGP exports best paths per IP prefix and so some options are lost as advertisements travel from the cloud to edge networks. Clouds could bypass this limitation if they advertise more prefixes, one per path, except that each advertisement comes with a cost. Advertisement cost comes from the cost of IPv4 prefixes (often much more than \$20k per /24 [75]) and their impact on global BGP routing tables.

BGP routing tables are growing for both v4 and v6 address spaces [43], for which the only solutions are to reject advertisements (bad) or to buy expensive routers (also bad) [15]. The importance of this problem is evidenced by the large body of research on compressing routing tables [10, 24, 95, 99]. Moreover, other work proposes advertising multiple prefixes to enhance performance [17, 97, 100, 111] so in the future it may be imperative for all networks to balance their individual goals (*e.g.,* enhancing performance) with the good of the Internet (minimizing BGP table impact).

Using IPv6 does not work for two reasons: first, IPv6 peering is less common than IPv4 according to `Azure`'s BGP data, so we cannot expose all the paths, and, second, routers can store 8× fewer IPv6 addresses than IPv4 addresses [30].

We do not know which routes to make available. Advertising multiple prefixes to expose multiple routes is promising, but we just argued that we cannot make all routes available. Since we can only make a limited set of routes available, and since the same routes may not be equally beneficial for all clients, we have to find the optimal subset of routes to make available that balance our goals of minimizing cost and improving performance. Finding this subset is a challenging combinatorial optimization problem whose complexity grows exponentially with the number of peerings, and whose objective function can only be measured infrequently (see Section 3.1 for more details).

3 SYSTEM DESCRIPTION

`PAINTER` (Precise, Agile INgress Traffic Engineering & Routing), summarized in Figure 4, consists of the `Advertisement Orchestrator` (§3.1) and the `Traffic Manager` (§3.2). The `Advertisement Orchestrator` exposes performant paths to users by advertising multiple prefixes via different peerings (2.2.2.0/24 and 3.3.3.0/24 in Figure 4, see Section 3.1 for more details) and the `Traffic Manager` tunnels traffic along best paths at fine traffic granularities (§3.2). `Azure` still advertises the anycast prefix (1.1.1.0/24 in Figure 4) since anycast offers low latency for most users [21, 54].

We designed `PAINTER` to be a service run by `Azure`, with nodes in edge proxies that we call `TM-Edges` and nodes in `Azure` PoPs that we call `TM-PoPs`. We collectively refer to all `TM-Edges` and `TM-PoPs` as the `Traffic Manager`. The edge proxy can be any technology that enables `Azure` to select from multiple tunnels at fine granularity (§2.3), but we believe cloud-edge network stacks are the most sensible proxy technology for our setting (§3.2). `TM-PoP` can be integrated with `Azure` front-ends in PoPs. Front-ends are entry-points into `Azure`'s network, perform some caching, and terminate TCP connections.

Cloud tenants are oblivious to `Advertisement Orchestrator` advertisements since traffic is tunneled between `TM-Edges` and `TM-PoPs`—tenants always direct traffic towards the anycast prefix.

3.1 `Advertisement Orchestrator`

Overview of Mechanism. Since BGP decides paths on a per-prefix basis, and since different ISPs may select paths in different ways, advertising multiple prefixes to different peers/providers can expose more paths, some of which can be more performant. We use this mechanism to mitigate path inflation and congestion.

However, as discussed in Section 2.4, we have to find a good *subset* of paths to expose. To maximize the benefit from a *limited* set of prefixes, the `Advertisement Orchestrator` advertises prefixes via *multiple* peerings (which we call prefix reuse) when it predicts that reuse will not hurt performance of other traffic. We define benefit as latency improvement over anycast, although computed strategies also implicitly offer added resilience (§5.2.4). One could use `PAINTER` to optimize any function of latency; here, we choose minimum latency over several measurements to approximate path propagation delay.

Since finding the optimal subset of paths to expose is a challenging optimization problem, our algorithm greedily computes which prefixes to advertise via which peerings in a way that minimizes average latency over `Azure` traffic. However, there is a chance that, after computing strategies and conducting BGP advertisements, users ingress at peerings that offer sub-optimal performance (akin to path inflation [92]). Over time the `Advertisement Orchestrator` learns from these instances of poor routing, computing strategies that get more benefit with fewer prefixes with each iteration.

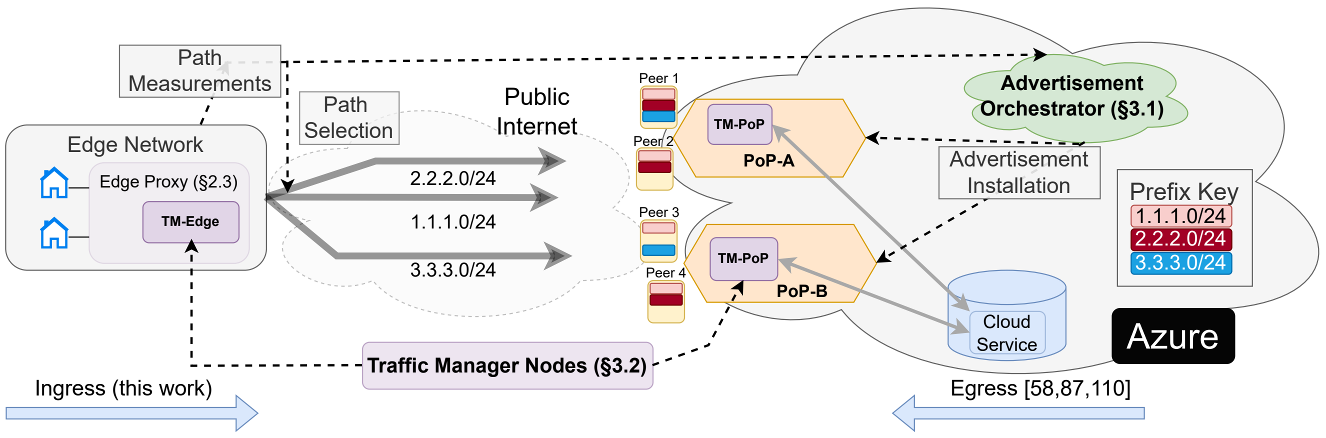


Figure 4: Overview of PAINTER. TM-Edges in edge proxies measure paths and tunnel traffic to TM-PoPs. TM-PoPs relay traffic destined to many prefixes to appropriate cloud services. The Advertisement Orchestrator uses path measurements to decide how to update PAINTER prefix advertisements. Prefix advertisements are represented by colored boxes inside peers at PoPs.

Unicast vs Anycast. Anycast commonly refers to announcing the same prefix via all PoPs and all peerings while unicast often refers to announcing the same prefix via *one* PoP via all peerings at that PoP [21, 116]. The Advertisement Orchestrator, however, may advertise a single prefix from multiple PoPs via a small subset of peerings at each PoP. Therefore the Advertisement Orchestrator's announcements do not fit nicely into either of these classifications, but we sometimes refer to them as unicast announcements to distinguish them from anycast and since unicast is commonly understood terminology. Prior work referred to advertisements to a subset of peerings as anycast [111].

Maximizing Benefit. We model an advertisement configuration A as a set of $\langle$peering, prefix$\rangle$ pairs where $\langle$peering, prefix$\rangle \in A$ means we advertise that prefix via that peering. More complex advertisement configurations (*e.g.,* BGP community tagging) are out of the scope of this paper. To simplify calculation, we logically group users in the same AS and large metropolitan area, referring to each group as a UG (user group), which is how Azure sometimes groups users for use cases such as monitoring performance.

Given a budget of prefixes, we seek an advertisement configuration A that maximizes benefit relative to a default anycast configuration, D, which is given by

$$B(A; D) = \sum_{\text{UG}} w(\text{UG}) \cdot I(A, \text{UG}; D) \tag{1}$$

where $w(\text{UG})$ is the weight (*e.g.,* traffic volume) of UG and $I(A, \text{UG}; D)$ is the improvement users in UG see under advertisement configuration A compared to configuration D. Improvement is latency from the advertisement compared to anycast. Since PAINTER can steer UGs with fine-grained control and includes anycast as an option, all UGs will have non-negative benefit over anycast.

Given the problem setup, we next describe how PAINTER solves this problem—*i.e.,* (1) how it models improvement, and (2) how it finds a maximizer of Equation (1).

Modeling Advertisement Improvement. PAINTER models (rather than directly measures) improvement associated with advertisement configurations since testing every configuration takes too long. Since we cannot measure all configurations, we use heuristics to predict the latency from UGs to Azure *ingresses*, and correct incorrect heuristics over time. TM-Edges (§3.2) conduct measurements. An ingress for a BGP peering is where traffic enters if Azure were to advertise a prefix solely via that peering.

We first compute possible ingresses for UGs. To determine whether an ingress is (very likely) a policy-compliant ingress for a UG—that is, which peerings each UG can reach according to routing policy—we inspect BGP routes, since BGP routes by nature encode policy-compliant routes, and derive policy-compliant ingresses in two ways. We first check Azure BGP feeds and say an ingress for a UG is policy-compliant if UG prefixes are announced over that peering (assuming the common case that a path that is policy-compliant in one direction also is in the opposite direction). To check for more policy-compliant ingresses, we then derive customer cones of each peer using ProbLink AS relationships [52] and Azure BGP feeds. An AS is in the customer cone of an Azure peer if the AS can reach the peer by following a series of customer to provider links [61]. By definition ASes will carry traffic from their customer cones to any destination, so if a UG's AS is in the customer cone of a peer, we call that ingress policy-compliant for that UG. Finally, we add all UGs to customer cones of Azure transit providers. To validate inferences about which ingresses are policy-compliant, we inspect millions of traceroutes from Azure clients and find that only 4% violate our assumptions.

We assume we have access to a system that measures latencies from UGs to each policy-compliant ingress individually (there are many examples in the literature [22, 32, 56, 111]). Hence, to predict UG latency we predict the *ingress*. We detail our measurement methodology in Section 5.1.

Since it is difficult to predict ingresses [64, 111], we make assumptions (detailed below) about UG ingresses and, in cases with uncertainty, assume all policy-compliant ingresses are equally likely. We then learn from incorrect assumptions over time. For each cloud prefix we calculate the average latency across all policy-compliant ingresses, and, since PAINTER can choose the *best* prefixes, we model improvement as the *highest* average improvement over the current configuration (Eq. (2)).

In cases where, to reach a prefix, a UG has two or more policy-compliant ingresses, we exclude ingresses that fall into two categories: first, we exclude ingresses for a UG that have a lower preference than other ingresses, where preferences are learned from past advertisements. For example, given that a previous advertisement advertised the same prefix to ISP A at a Tokyo PoP and ISP B at a Miami PoP and a UG in Miami routed to the prefix through Tokyo, we would exclude the ISP B at Atlanta from UG latency predictions in future calculations for that UG involving both ISPs. This information is valuable since the UG in Atlanta ostensibly has very poor performance to Tokyo. As we incorporate more observations over time, the `Advertisement Orchestrator` *learns* better advertisement strategies (§5.1). Preference models of routing are used in prior work [111].

Second, we exclude ingresses that would result in a UG reaching an Azure PoP more than D_{reuse} km (reuse distance) from the closest PoP advertising that prefix, as prior work found large path inflation is rare [21, 54]. D_{reuse} is a configurable parameter. For example, given that we advertise a prefix at a Central US PoP and a Tokyo PoP, we assume a UG near Eastern US (1,500 km to Central US, 11,200 km to Tokyo) would reach the Central US PoP so long as D_{reuse} < 11,200 - 1,500 = 9,700 km.

Increasing D_{reuse} leads to fewer incorrect assumptions, but limits how much we can reuse a prefix since greater D_{reuse} tends to require more physical distance between ingresses. Hence, D_{reuse} represents a tradeoff between greater prefix reusability (saving cost) and more learning iterations (saving time). We explicitly quantify this tradeoff in Appendix E.2.

To summarize, predicted improvement relative to the current configuration D can be written as

$$\tilde{I}(A, \mathsf{UG}; D) = \max_{P \in \mathscr{P}}\left(\min_{P' \in \mathscr{P}} \mathbb{E}_D(l(\mathsf{UG}, P'))\right) - \mathbb{E}_A(l(\mathsf{UG}, P)) \qquad (2)$$

where $\mathscr{P}$ is the set of prefixes being advertised, and $\mathbb{E}_A(l(\mathsf{UG}, P)$ is the expected latency from UG to P over policy-compliant ingresses under advertisement configuration A. For a given UG and configuration, PAINTER can select the prefix P that yields the lowest latency, and so the improvement compares the lowest latency prefix P' in the current configuration to the prefix P in the candidate configuration A that yields the biggest improvement.

As described above, our expectation assumes all policy-compliant ingresses are equally likely unless they are a lower preference than other active ingresses or they result in more than D_{reuse} inflation for UG*s* in which case they have zero likelihood; hence, the expectation operator varies with UG and as we learn more about UG preferences. We do not consider that prefix for a UG if a UG does not have a policy-compliant ingress for that prefix. The tilde above I emphasizes that Equation (2) approximates true improvement in Equation (1) due to this expectation.

Solving For Optimal Advertisement Configurations. Summarizing, we wish to maximize Equation (1), which we model using Equation (2). Maximizing Equation (1) by exhaustive enumeration is infeasible since the number of advertisement configurations grows exponentially with prefix budget, and since it takes time to test each configuration to avoid route flap damping, so PAINTER greedily allocates prefixes to peerings to maximize benefit. Algorithm 1 summarizes how we choose advertisements. The `Advertisement Orchestrator` would install computed configurations at Azure PoPs, and notify the `Traffic Manager` about available prefixes via a control channel.

Algorithm 1 Algorithm for selecting advertisements.

```
  Input Prefix Budget PB, minimum reuse distance $D_{reuse}$
$RM \leftarrow []$                                                  ▹ Routing model—how users route to deployment
while $learning$ do                                ▹ Terminate learning when little marginal benefit increase
    $CC \leftarrow []$                                                         ▹ Stores current configuration
    for $p\ in\ range(PB)$ do                                                  ▹ Each prefix in the budget
        while $True$ do                                          ▹ Advertise this prefix across many peerings
            $peering\_improvements \leftarrow calc\_improvements(CC, RM)$                       ▹ Equation 2
            $ranked\_peerings \leftarrow sort(peering\_improvements)$                                  ▹ Rank
            $found\_peering \leftarrow False$                                          ▹ Can we find a peering?
            for $next\_best\_peering\ in\ ranked\_peerings$ do                                ▹ Greedy search
                $NP \leftarrow (p, next\_best\_peering)$                      ▹ Proposed new prefix,peering
                if $B(NP; CC) > 0$ then                                            ▹ Require positive benefit.
                    $found\_peering \leftarrow True$                                      ▹ Choose this one
                    $break$
                end if
            end for
            if $found\_peering$ then                                  ▹ Found peering to advertise prefix to
                $CC.append(NP)$                                        ▹ Advertise prefix to new peering
            else                                                                  ▹ No beneficial peerings
                $break$                                                           ▹ Move to the next prefix
            end if
        end while
    end for
    $RM \leftarrow execute\_advertisement(CC)$                 ▹ Advertise CC and update routing model
end while
return CC
```

Algorithm 1 takes two hyperparameters: the minimum reuse distance, D_{reuse}, which implicitly affects improvement calculations (Eq. (2)), and a prefix budget *PB*.

At each iteration of the outermost loop in Algorithm 1, PAINTER computes and conducts an advertisement strategy. PAINTER measures which of its assumptions about how UG*s* are routed to ingresses were incorrect, and incorporates this information into future loop iterations. We abstract this information as a "routing model" object in Algorithm 1. We manually terminate this learning process after seeing the marginal benefit from more learning iterations fall below a threshold.

At each iteration of the second loop in Algorithm 1, PAINTER tries to advertise a prefix via as many peerings as possible. PAINTER considers adding peerings in ranked order of estimated improvements relative to the current configuration (Eq. (2)). We add a ⟨`peering, prefix`⟩ pair to the current configuration if the advertisement provides positive benefit (Eq. (1)). Advertising the same prefix via multiple peerings (prefix reuse) allows us to accumulate benefit without quickly exhausting our prefix budget. However, prefix reuse can lower expected improvement for certain UG*s* if the reuse introduces worse paths for some UG*s*. When the expected marginal improvement for a prefix is non-positive, we continue to the next prefix, and so on, until our prefix budget is exhausted.

After computing the configuration, PAINTER advertises the configuration to peers and measures new ingresses/latencies to update its routing model. Over learning iterations, the `Advertisement Orchestrator` learns which assumptions about UG ingresses were incorrect and incorporates these into benefit estimates via the routing model (Eq. (1)). Hence, each successive advertisement configuration tends to yield greater benefits with fewer prefixes.

Algorithm 1's complexity grows quadratically with the number of ingresses, linearly with the number of UG*s*, and linearly with the number of learning iterations. In practice, computation and convergence are fast (§4).

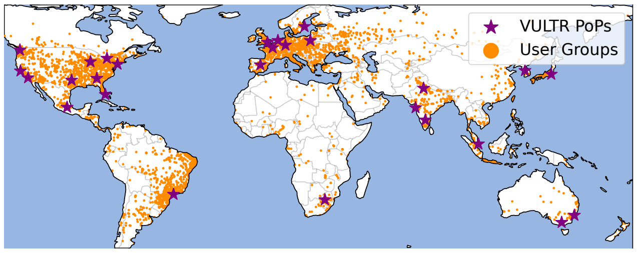


Figure 5: `PAINTER` prototype deployed globally at Vultr's PoPs, and `UG`*s* we probed.

3.2 `Traffic Manager`

For our setting, cloud-edge network stacks are the most sensible form of edge proxy in which to situate `PAINTER` since they provide compute, already perform networking decisions/operations, and reside close to/inside of edge networks. Cloud-edge network stacks can enact routing policy solely using *existing* hardware, facilitating deployment. Situating `TM-Edge` here allows clouds to iterate on system versions with existing software frameworks.

Resolving Available Prefixes. Each `TM-Edge` may send traffic to many `TM-PoPs`. `TM-Edge` resolves the set of available `TM-PoPs` via communication with an `Azure` service. `TM-Edge` queries `TM-PoP` for the available set of ingress IP addresses (*i.e.,* destinations) for each service (corresponding to possibly different paths). Available destinations are computed by the `Advertisement Orchestrator` in Algorithm 1. Upon establishing tunnels with each available destination, each `TM-Edge` identifies the `TM-PoP` it communicates with along that tunnel. Hence, each `TM-Edge` maintains a mapping of destination prefixes to PoPs (which is difficult to compute apriori, as prefixes may be advertised via multiple peerings at multiple PoPs). Although a `TM-Edge` may have paths to every `TM-PoP`, available PoPs may vary depending on the service since each service may only be served from certain PoPs or regions.

Selecting Destinations and Mapping Flows. Given a set of available destinations (prefixes), the `Traffic Manager` can use different destination selection policies according to enterprise network or service goals. (We do not innovate in this space.) We follow high-level lessons from prior work about how to select destinations to avoid oscillations [38].

As the `Traffic Manager` continuously measures and selects the best destination, it directs new flows toward this destination. Once the `Traffic Manager` maps a flow (5-tuple) to a `TM-PoP` (*i.e.,* PoP), the mapping is immutable for the lifetime of that flow. This design decision limits flexibility, but also prevents loss of connection state and subsequent performance problems for the user without needing to design a connection-handover system [13, 106].

3.3 `PAINTER` Limitations

`PAINTER` does not solve all performance and reliability problems faced by `Azure`. `PAINTER` cannot avoid performance problems or failure shared by all paths to `Azure` (*e.g.,* if the problem is due to an enterprise's single ISP), problems in the egress direction (although those are addressed by prior work [58, 87, 110]), and only works for traffic controllable by a `TM-Edge`. Moreover, `PAINTER` cannot mitigate problems at the application layer (other existing systems are designed to detect application layer failure [18]).

4 PAINTER IMPLEMENTATION

`Advertisement Orchestrator`. The `Advertisement Orchestrator` takes measurements from `TM-Edges` and hyperparameters as inputs (see algorithm 1) and installs advertisement configurations. The `Advertisement Orchestrator` computes configurations at a rate of approximately 30 seconds per prefix where calculations include thousands of ingresses and tens of thousands of `UG`s. Configurations need not change often (§5.1).

Despite the algorithm having a running time that is quadratic in the number of ingresses, in practice the implementation runs quickly (30 seconds), especially relative to how often it has to run (Section 5.1.3 shows that it need only be run monthly). Quick runtimes are due to the nature of `UG` connectivity, and implementation optimizations. For example, `UG`s tend to have paths via a relatively small fraction of ingresses, speeding up computation.

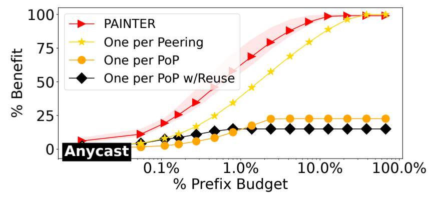
We prototype the `Advertisement Orchestrator` on the PEERING testbed [85], which is now deployed at Vultr cloud locations [103]. Vultr is a global cloud that allows tenants to announce their own IP prefixes from Vultr, letting us emulate the control we would have if we were the cloud offering `PAINTER`. Our prototype uses 25 Vultr PoPs on 6 continents with 5,000 neighbor ASes and 9,000 ingresses (Fig. 5), offering us a rich platform for testing advertisement strategies.

We also implement a partial `Advertisement Orchestrator` prototype on `Azure`. We could not change BGP announcements from `Azure` for operational reasons, nor could we conduct measurements to all `UG`s. Instead, we use a combination of real and simulated measurements to evaluate the `Advertisement Orchestrator` on `Azure`— we detail this methodology in Section 5.1. `Azure` is a global cloud with 200 data centers interconnected by 175,000 miles of lit fiber whose traffic is managed by a software-defined WAN [70]. Traffic enters and leaves Azure's WAN through roughly 200 PoPs which are often in major metropolitan areas [70]. `Azure`'s WAN connects PoPs to data centers. PoPs also have peering routers which connect `Azure` to more than 4,000 networks [71] Some networks connect at multiple PoPs, most only at one [9].

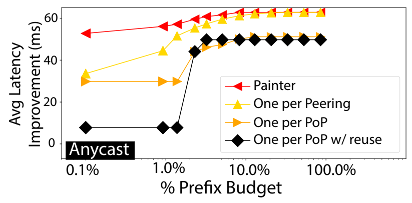
`Traffic Manager`. The `Traffic Manager` steers traffic between `TM-Edges` in edge proxies and `TM-PoPs` at `Azure` PoPs. We prototyped the `Traffic Manager` on cloud VMs using a lightly modified version of FlexiWAN [36] since it is open source and works on common cloud VMs. A key difference between our solution and typical `SD-WAN` use is how tunnels are configured. We configure multiple tunnels between the same two physical endpoints using addresses from different IP prefixes, which is not a common configuration for `SD-WAN` as `SD-WAN` would not benefit from such a configuration without an `Advertisement Orchestrator`. We more thoroughly describe how tunneling works in Appendix D.

5 PAINTER EVALUATION

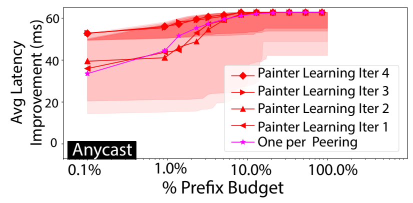
We thoroughly evaluate `PAINTER` on many dimensions. Perhaps most importantly, we deploy a functional prototype on a public cloud and achieve an average latency improvement of 60 ms across thousands of `UG`*s* (§5.1, Fig. 6b). We also show that the `Traffic Manager` fails over from a unicast path to a backup at RTT timescales using our prototype (§5.2.3, Fig. 10).



(a)



(b)



(c)

Figure 6: PAINTER achieves more benefit with less budget compared to other advertisement strategies shown using simulated measurements from Azure (6a) and from an actual global cloud (6b). PAINTER learns from incorrect assumptions over iterations (6c). Shaded regions show PAINTER's uncertainty prior to testing a strategy.

5.1 Advertisement Orchestrator

5.1.1 Measurements

We evaluated the Advertisement Orchestrator in two settings: first, in a simulation using Azure measurements to assess how the Advertisement Orchestrator scales, and second in a prototype on PEERING (§4) to assess how the Advertisement Orchestrator performs in the wild. We measure all targets using ping 7 times and compute minimum latencies to approximate propagation delay.

Azure Measurements. We could not test our advertisements on Azure due to operational reasons, so we estimated the latency UGs would experience to Azure ingresses and computed a range of latencies UGs could experience to each prefix. To estimate the latency UGs would experience through an ingress to a prefix, we measure latency from UGs to either (a) the corresponding peering subnet or, when that is not possible, (b) an IP address in the peer/provider's IP space geolocated to within *GP* km (geo-precision, configurable) of the associated PoP. We verify IP address locations using speed of light constraints from RIPE Atlas probes [93] with known locations.

We use RIPE Atlas probes to measure latencies to Azure peerings. Considering paths from all UGs to Azure through all policy-compliant peerings, we were able to obtain measurement targets corresponding to 80.6% of Azure user traffic when $GP = 450$ km. Specifically, we counted all policy-compliant ⟨UG, ingress⟩s, weighted by UG traffic volume (counting all geographically proximal, policy-compliant ingresses for a UG as equally likely). (We more thoroughly discuss and validate our heuristic for estimating latency in Appendix B and verify that this measurement heuristic for predicting latency through ingresses agrees with the actual latency to within 2 ms for most cases where we were able to measure both, suggesting that our measurement heuristic estimates latency with sufficient accuracy.) We found 450 km to be a good tradeoff between coverage and accuracy.

We measure latency from probes to all their policy-compliant ingresses (§3.1). We group measurements from RIPE Atlas probes in the same UG, yielding measurements from 4k UGs. Since RIPE Atlas covers a relatively small number of UGs (only 47% of Azure traffic volume), we then simulate measurements using a methodology that extrapolates RIPE Atlas ingress latencies to nearby UGs that do not house RIPE Atlas probes. Our simulated measurements assume users in the same location have the same mean latency to Azure and the same distribution of relative latencies along alternate paths to Azure. (Not the same latencies, just the same distribution of latencies.) Extending our measurements helps us observe convergence/scaling properties that are only visible using measurements from all UGs. We describe our simulation methodology in Appendix C. Simulated measurements are from hundreds of thousands of UGs to thousands of ingresses.

PEERING Measurements. For our PEERING prototype (§4), we measured client latencies by pinging clients from the deployment as in prior work [32, 111]. We do not need to estimate latency here since we can conduct actual advertisements with our prototype. We measure from PEERING to 40k UGs. From our measurements, latency gains are heavily concentrated among its ingresses—we only saw latency improvement for approximately 8k UGs through 250 out of 9,000 ingresses which were mostly transit providers. We show the global scale of both our deployment and UGs with which we evaluated PAINTER in Figure 5.

5.1.2 Efficiently Maximizing Benefit

Methodology. We first compare our ability to efficiently improve latency to other advertisement strategies used by clouds. We consider (a) announcing regional prefixes to transit providers since Azure makes *regional* advertisements for services in some regions to transit providers to enable multiple route offerings for customers, and (b) assigning each PoP its own prefix that it announces to all peerings (One per PoP) since prior work explored using per-PoP announcements to lower latency [21, 111]. In practice, regional offered little to no latency benefit over anycast so we do not include it in figures. To the best of our knowledge, these strategies comprise the state-of-practice.

In addition to these practical/studied configurations, we also compare PAINTER to two hypothetical ones: one that advertises a single prefix at each PoP, but allows prefix reuse when PoPs are more than D_{reuse} km apart (One per PoP w/ Reuse), and one that advertises a unique prefix across each peering (One per Peering). We set $D_{reuse} = 3{,}000$ km. The One per Peering strategy uses many prefixes to realize benefit, but is guaranteed provides all the benefit since all UGs have a route to their best ingress.

For each strategy, we compute a range of possible latency benefits since UGs may have several possible ingresses for a prefix. Using this range of improvements, we compute an *estimated* improvement, which uses the fact that inflated paths to far-away PoPs are less likely. We compute a weighted average of benefit over possible ingresses, where the weights correspond to approximate probabilities that paths are inflated by corresponding amounts. (We calculate probabilities from Azure's inflation data.) Prefix budgets are reported as a percent of the number of ingresses, since advertising a unique prefix via each ingress would trivially give UGs all the latency benefit since it would expose all the paths.

Results. Figure 6a shows the *estimated* benefit each strategy attains as a percent of the total possible benefit (Eq. (1)) as the prefix budget varies on Azure's deployment, demonstrating that PAINTER finds advertisement strategies for Azure that give far more benefit than other advertisement strategies at each prefix budget. We show the entire range of possible benefits for PAINTER since the range is small, and for One per Peering since that strategy has no uncertainty. Including ranges of possible benefit for solutions with high uncertainty renders the graph difficult to read and so those ranges are shown in Appendix E.1. We show benefit as a percent of the total possible for anonymity. For example, 50% benefit corresponds to UGs achieving half the total possible latency decrease over anycast, on average.

The One per PoP w/Reuse and One per PoP strategies do not consider advertising different prefixes to different peerings at a single PoP, so both strategies fail to uncover the routes necessary to offer as good estimated benefits as PAINTER. In practice, this means UGs have several policy-compliant ingress options at each PoP which leads to low expected benefit, even though all peerings are covered with very few prefixes. PAINTER saves 3× the number of prefixes as One per Peering at 75% benefit due to prefix reuse.

Figure 6b plots average latency improvement over clients that have non-zero improvement for our prototype on PEERING (§4), demonstrating that the Advertisement Orchestrator also performs well on real Internet paths. To attain 90% of the benefit (54 ms average), PAINTER uses roughly 10% as many prefixes as the next-best strategy (One per Peering). After convergence, 25 prefixes was enough to achieve more than 99% of the benefit. Figure 6c demonstrates that it took a few iterations for PAINTER to realize these benefits—as PAINTER learned from incorrect assumptions about client ingresses, it was able to find drastically better advertisement strategies. Shaded regions show uncertainty before testing strategies, where the narrowing darker region demonstrates that we gain confidence that our strategies perform well over time, going from 44 ms uncertainty to 8 ms. For example, PAINTER quickly learned that many New York users preferred an ingress in Amsterdam to one in New York, and learned not to advertise the same prefix to those two ingresses.

PAINTER's initial assumptions led to poorly performing strategies for two reasons: (a) a large percentage of the benefit was concentrated in a relatively few number of ingresses so a few incorrect assumptions had outsized effects and (b) most benefit was through transit providers but those transit providers tended to inflate routes even over very large distances (10k+ km).

Our prototype outperforms all other strategies, even using only one prefix since, with other strategies, too many UGs get a *bad* route to some sub-optimal ingress, with little or no benefit over anycast. PAINTER identifies which subsets of routes offer improvement and only advertises those, refining its routing model and hence its set of advertisements over time to replace poorly performing routes.

PAINTER saves prefixes compared to other strategies which is important to Azure since prefixes are expensive and since advertising too many prefixes can bloat BGP routing tables [30]. In practice, we would want to limit PAINTER's BGP footprint to be similar to other large cloud/content providers, which still leaves a lot of room to optimize. For example, 8 out of 22 of the hypergiants [39] advertise at least 500 /24 prefixes according to Routeviews [19] and Figure 6a suggests that even 200 prefixes could get Azure roughly 90% of the possible benefit (we cannot share the precise number). Realizing this benefit requires at least 3× as many prefixes when advertising One per Peering and is impossible with the existing strategies of advertising One per PoP (with or without reuse).

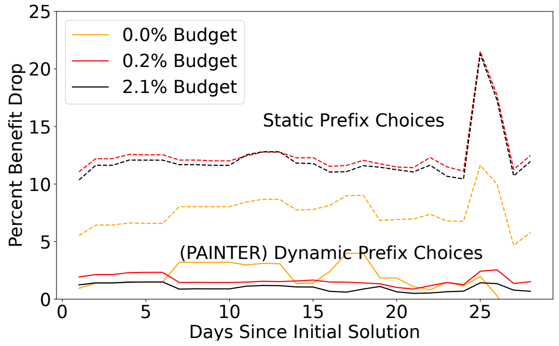


Figure 7: PAINTER advertisement benefits remain for at least a month whether users change their prefix choice over time (Dynamic Prefix Choices) or they do not (Static Prefix Choices).

5.1.3 Reconfiguration, Who Needs That?

Frequent announcement reconfiguration could inundate routers and hinder the ability to predict traffic dynamics so we next assess how frequently PAINTER's announcements need to be updated.

Methodology. To analyze advertisement optimality over time, we first solve for a configuration using a week of RIPE Atlas measurements to ingresses before an arbitrary start date. We only use RIPE Atlas measurements, not simulated measurements, since we want to evaluate reactions to real network dynamics. We then evaluate how optimal the *fixed* configuration is over time (*i.e.,* recalculate the fraction of benefit we achieve) with respect to *updated* latencies from continuous RIPE Atlas measurements to ingresses conducted over the following month.

Results. The solid lines in Figure 7 show the drop in benefit over time for a few representative prefix budgets. There is minor, random benefit degradation over time (at most 3%) which could suggest that (a) most latency benefits are from steady state routing inefficiency, (b) most problems that arise tend to degrade all good route options similarly [86], and/or (c) PAINTER configurations are resilient to new problems that arise through routing changes. To assess how often the last case occurs, the dashed lines show the drop in benefit over time assuming each UG continues to use its original prefix choice at t = 0 whereas the solid lines use the original configuration of announcements but use the routes made available by those announcements to allow PAINTER to switch UGs to different prefixes dynamically. Benefit loss when UGs do not switch prefixes over time (shown by the dashed lines) is approximately 10% worse (*i.e.,* UGs attain about 85% of the benefit rather than about 95%), suggesting that a major reason PAINTER needs infrequent changes is that it offers good backup paths to UGs, so that at each time step a low latency path is available. Hence, PAINTER's advertisement strategy is naturally resilient to routing changes and so likely requires little reconfiguration in practice.

5.2 Traffic Manager

5.2.1 Highly Deployable, Very Precise

Situating the Traffic Manager on cloud-edge network stacks jointly optimizes PAINTER in two dimensions: deployability and precision, which we capture in Figure 8. A solution is more deployable if it can direct more traffic with less deployment effort. A

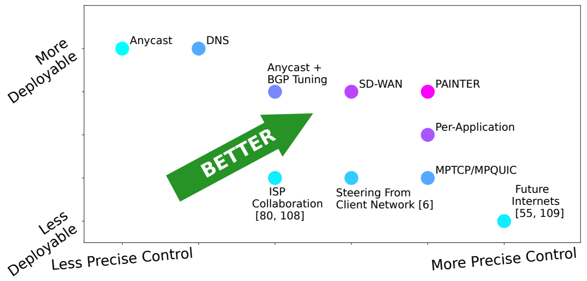


Figure 8: Situating PAINTER partially on cloud-edge network stacks gives clouds precise, deployable traffic control.

solution is more precise if it can control finer traffic quantities at finer time granularities and direct them over more paths to clouds. Figure 8 qualitatively buckets these metrics to provide a simple comparison among solutions; we quantitatively compare subsets of these solutions on specific dimensions in subsequent sections.

Popular approaches such as anycast and DNS are highly deployable (thus their popularity) but fail to give the same precise control over traffic as PAINTER. Variants of PAINTER that use different edge proxies—*i.e.,* where TM-Edge is built into user applications or OSes and/or using MPTCP/MPQUIC to steer traffic could achieve the same precision but face deployability obstacles [11, 104]. Deploying the Traffic Manager (specifically TM-Edge) into applications could similarly face deployability obstacles since the framework may have to be deployed and maintained independently for each application. Enterprise networks that use SD-WAN devices with multihoming to optimize routing have fewer path options than PAINTER (§5.2.4), making them less precise. Section 6 discusses related systems that involve ISP Collaboration [80, 108], future Internets [55, 109], and steering solely from the client network [6], but they generally offer precise traffic control at the cost of unrealistic/unscalable assumptions (sacrificing deployability).

5.2.2 Fine-Grained Traffic Control

Figure 9 quantifies the benefit of PAINTER's fine-grained traffic control compared to other approaches, showing the network granularity at which other traffic engineering strategies steer traffic and a possible implication of controlling traffic at coarse granularities. Both modifying prefix announcements (BGP) and updating DNS records (DNS) steer traffic at far coarser granularities than PAINTER, which has latency implications for users.

Methodology. Because users relay DNS requests via recursive resolvers, serving an updated DNS record can generally impact all users of that resolver, which corresponds to a certain volume of Azure traffic. BGP-based steering can modify prefix announcements to peers/providers at PoPs. To obtain an optimistic bound on the granularity at which BGP can control traffic (without PAINTER), we assume that traffic is affected at the ⟨peering, user AS⟩ granularity (all traffic entering Azure via peering from users from a particular AS). We choose this granularity to model a case where, for example, Azure only updates an announcement via a specific peering targeting a specific user AS using BGP communities. As in DNS, affecting traffic at this granularity corresponds to a certain volume of Azure traffic which we measure as the amount of traffic traversing that connection from that user AS. In practice, BGP advertisement updates could shift greater or lesser amounts of traffic than the ⟨peering, user AS⟩ level, but in any case the shifts will be unpredictable and coarse.

To quantify the benefit of precise control, we compute benefit over budget (as in Fig. 6a) (a) for PAINTER and (b) for PAINTER but assuming PAINTER uses DNS to assign clients to prefixes. Using DNS, PAINTER maps each recursive resolver to the prefix with the best overall benefit for traffic directed by that resolver. The prefix may be optimal for some of the resolver's clients but not others.

Recursive resolvers *could* serve users at finer network granularities if they support EDNS0 Client Subnet [ECS]. However, recent work found only 72 networks *worldwide* use ECS [20]. Most significantly, Google Public DNS supports it. Hence, we compute the benefit over budget assuming traffic mapped by Google Public DNS can be mapped per /24 using ECS.

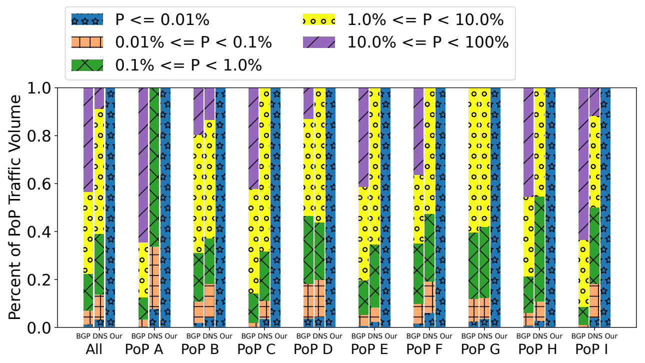
Results. In Figure 9a we show the granularity at which each solution affects different volumes of traffic overall (column 'All') and for the top 10 PoPs (PoP-X) by volume. Each PoP is associated with three bars for BGP, DNS, and PAINTER from left to right. The hatching and coloring of the bars (denoted with P for Precision in the legend) corresponds to the granularity at which each solution controls traffic. For the example of PoP A, 64% of ingress traffic comes from (⟨peering, user AS⟩) pairs responsible for between 10% and 100% of all traffic arriving at PoP A, meaning that, if Azure tried to shift traffic from one of these ASes to a different peering or path, the shift would entail at least 10% of traffic moving en masse. For the remaining 36% of traffic, 23% comes from pairs responsible for between 1% and 10%, 9% comes from pairs responsible for between 0.1% and 1%, and the other 4% comes from pairs responsible for less than 0.1%. In contrast, 70% of traffic is directed by recursive DNS resolvers that each steer between .1% and 1% of traffic arriving at PoP A, and so Azure would shift less than 1% of traffic by changing its DNS response to any one of these resolvers allowing more fine-grained redirection. The granularities at which each of BGP and DNS controls traffic vary significantly across PoPs—for example, DNS controls 100% traffic arriving at PoP A at granularities finer than 1%, whereas DNS only controls 43% of traffic at this granularity at PoP B. PAINTER could control all traffic at the finest granularity, since PAINTER controls individual flows.

Figure 9b quantifies one drawback of coarse control—inability to fully benefit from the Advertisement Orchestrator's advertisement strategies. Using DNS sacrifices roughly half the benefit as is possible with fine-grained redirection, since some DNS resolvers serve diverse UGs for which no single path is optimal. We found that regions with poor routing (*i.e.,* those responsible for most of the benefit the Advertisement Orchestrator provides) correlated with regions that hosted LDNS serving geographically disparate users. This correlation leads to a drastic benefit difference between PAINTER with and without its Traffic Manager.

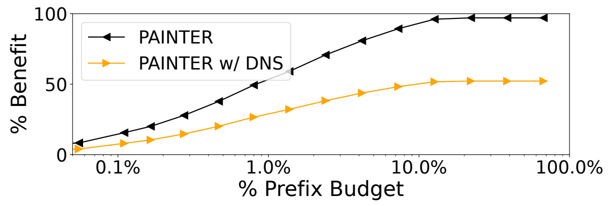
5.2.3 Quick, Agile Reactions

Methodology. Prior work proposed announcing unicast prefixes and directing users to them via DNS to improve latency [21, 34]. Using DNS to improve latency raises *availability* concerns since DNS reaction times are slow (§2.2). We show PAINTER realizes the latency improvements of unicast while retaining availability.

We advertise an anycast prefix (1.1.1.0/24) at two PoPs and one prefix to each ISP at those two PoPs. Figure 10a depicts the physical scenario our system models (not showing all the advertisements to remove clutter). During normal operation, PAINTER chooses a prefix to a provider at PoP-A (2.2.2.0/24) since the path to it is lower latency than the default anycast path. At 60 seconds, we withdraw

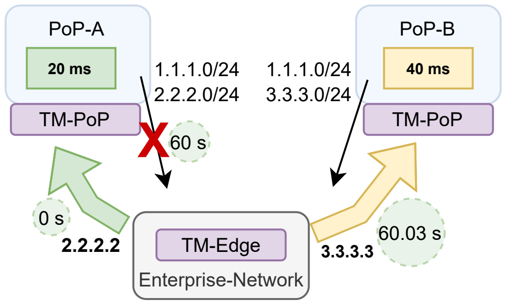


(a)

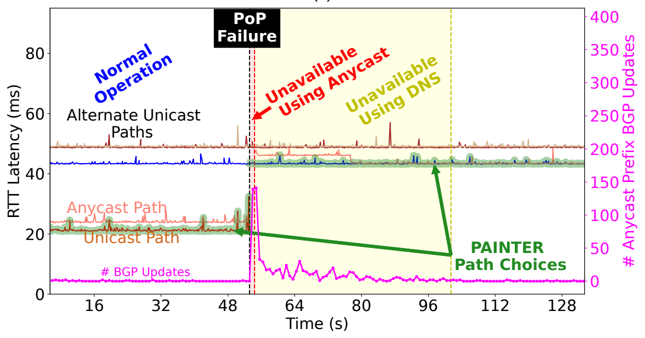


(b)

Figure 9: Alternate approaches to steering ingress traffic have coarse control (Fig. 9a) which limits the advertisement effectiveness (Fig. 9b).



(a)

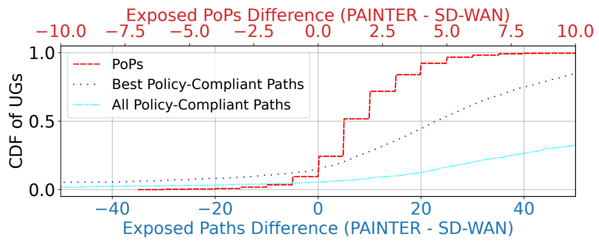


(b)

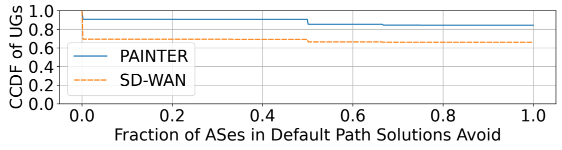
Figure 10: PAINTER switches between two paths during PoP failure much faster than other solutions. First (0 s) PAINTER chooses the prefix 2.2.2.0/24. Second (60 s) PoP-A fails, so prefix 2.2.2.0/24 is withdrawn and 1.1.1.0/24 reconverges. Third (60.03 s) PAINTER switches over to prefix 3.3.3.0/24 at PoP-B in approximately 1 RTT.

all prefixes at PoP-A, which is meant to model a failure in the PoP advertising PAINTER's chosen prefix.

Results. The time series graph in Figure 10b illustrates typical system operation during such a failure. The left axis measures latency to each prefix, and the right axis measures the number of BGP updates for the anycast prefix as seen by RIPE RIS BGP collectors



(a)



(b)

Figure 11: PAINTER exposes more paths/PoPs than multihoming (Fig. 11a) enhancing resilience to intermediate AS failure (Fig. 11b).

[94] which estimates BGP churn. BGP churn can cause performance problems for the underlying paths so even after a route is available, performance problems can continue until convergence [105]. PAINTER's selected paths are shown with highlighted lines. Prior to failure, PAINTER measures five possible prefixes (the anycast and four single-transit prefixes) and selects the prefix with lowest latency (2.2.2.0/24 at PoP-A). PAINTER detects the loss of reachability (labeled vertical line) and switches to an alternate single-transit prefix (3.3.3.0/24) at the PoP-B within roughly an RTT, since that destination has the next-lowest latency. Over many experiments we found the Traffic Manager typically detected failure within 1.3 RTTs (the theoretical minimum is $\frac{1}{2}$ RTT).

PAINTER's reaction times represent an order of magnitude of improvement over existing techniques—typical BGP convergence times are on the order of minutes [116] and DNS TTLs are usually between 1 and 10 minutes [4, 73]. It took one second for the anycast address to become reachable after withdrawal (red region), and roughly 15 seconds to converge to the final path as shown by the spike in RIPE RIS updates and the change in latency for the anycast path at around 80 seconds. The figure assumes DNS takes 60s to respond to failure (yellow region), but actual failover time would depend on many factors (*e.g.*, client OS, last-mile caches, TTL at the time of failure [4]). Although anycast availability is good compared to DNS (1 second loss), PAINTER's is near optimal (30 ms).

5.2.4 Exposing More Paths

We now compare PAINTER to SD-WAN which has been around for decades and has path-switching capabilities for performance and resilience. SD-WAN devices typically select between paths via a multihomed enterprises' ISPs, or a direct path to Azure if the network has a direct peering. We use the term SD-WAN below to refer to this capability to select among ISPs, and evaluate PAINTER against this scenario. Figure 11a shows that PAINTER exposes more paths and PoPs than selecting among ISPs, enhancing resilience.

Methodology. We compute paths that SD-WAN and PAINTER would have to Azure for all UGs. We first compute the number of paths for SD-WAN by counting the number of ISPs for each UG for which we see traffic to Azure, adding one additional path if the UG's AS connects directly to Azure. An SD-WAN device could use these different paths if it tunneled traffic through each of these

ISPs, or the direct connection. We also note the ingress PoP for each of these paths through ISPs, assuming that, were traffic from an SD-WAN device to be routed through a provider to Azure, the traffic would be routed similarly to Azure clients in that ISP. This assumption is reasonable since routing is destination-based.

To calculate the number of paths for PAINTER, we tabulate possible PoPs that Azure clients may ingress in by looking at the PoPs at which 90% of user traffic in that UG's geographic region ingress according to Azure logs. We do not consider all PoPs to remove high-latency routes. This restriction likely does not overestimate low-latency PoP choices, given that prior work found 90% of traffic in a large CDN reaches a PoP within 1,000 km of the closest possible [54]. We then count the number of policy-compliant paths from that AS through peerings at each of these PoPs to Azure according to the common definition of policy-compliant [37] using BGP data.

After calculating policy-compliant routes, we form two estimates of the number of paths PAINTER could expose. As a lower bound, we consider one path per peering, while as an upper bound we consider *all* policy-compliant paths. The lower bound corresponds to counting Azure ingresses for UG*s* (which is what our Advertisement Orchestrator exposes), whereas the upper bound models a hypothetical Advertisement Orchestrator that announces prefixes with different advertisement attributes to expose even more paths (*e.g.,* prepending) as in prior work [100].

Finally, to quantify PAINTER's added resilience, for each UG we compute the fraction of ASes in the default path to Azure that we could avoid with PAINTER and with SD-WAN. We tabulate ASes on paths using traceroutes from clients [22].

Results. Most networks have only 2 or three ISPs and so only have 2 or 3 paths to choose from with SD-WAN. Figure 11 shows PAINTER offers 23 more paths than SD-WAN for most UG*s*, and it offers at least 40 more paths for 25% of UG*s* (Best Policy-Compliant Paths) as shown by Figure 11a. All Policy-Compliant Paths indicates that PAINTER *could* expose far more by manipulating advertisements. PAINTER offers policy-compliant routes to 4 more nearby PoPs than SD-WAN for 10% of UG*s* (PoPs).

Having more paths could help route around congestion or failures in intermediate ASes. Figure 11b shows that, for 90.7% of UG*s*, PAINTER can redirect traffic through a policy-compliant path that avoids *all* ASes on the default path, but the same is only true for 69.5% of UG*s* for SD-WAN. Hence, it is more likely that PAINTER could avoid routing problems introduced by intermediate ASes.

6 RELATED WORK

Egress Traffic Engineering. There are large scale systems that steer egress traffic, selecting one of multiple paths to client prefixes to either improve performance [58, 87, 110] or optimize peering costs [88, 113]. PAINTER coexists with and acts independently of these systems, improving end-to-end path latency.

Ingress Traffic Engineering. PECAN issues multiple advertisements to a single ISP to expose routes and uses DNS to steer traffic [100]. PAINTER is more agile than DNS and works at scale in a global cloud—it is unclear how PECAN's design (tested with a single ISP) translates to this setting. Another study used a combination of MPTCP and SD-WAN to steer traffic across tunnels to the cloud [112]; PAINTER differs in that it investigates which computes which tunnels to set up. Other work steers traffic by advertising prefixes to different ISPs [97], but does not scale to networks like Azure with thousands of peerings.

Research investigated the efficacy of using DNS [23, 56, 76], anycast [21, 111], or a combination [22, 34] to steer traffic. Concurrent work looks at regional anycast announcements where DNS maps clients to regions [115]. PAINTER can work on top of such deployments to add agility and precision. One company uses tight coordination to map hypergiant traffic to ISP ingresses [80]. Our current implementation only requires the enterprise to deploy a cloud-edge stack that Azure can control, but PAINTER's architecture could use edge proxies that do not require coordination with the enterprise (§2.3). PAINTER never requires coordination with networks in between the proxy and the cloud. TIPSY infers where traffic will end up if announcements are withdrawn [64], an orthogonal problem. Anyopt exposes paths using advertisements, but does not scale to deployments with thousands of ingresses [111].

Contemporaneous work called Tango exposed multiple paths by advertising multiple prefixes and tunneled traffic in real-time over the best one [17]. Despite the similar mechanisms, differences in the settings lead to Tango addressing different challenges than the ones needed in our setting. Tango exposed performant paths on the public Internet between distributed edge networks that lacked a private WAN between them, where the path choices were between a few transit providers each for a few tens of data centers. PAINTER instead optimizes paths between the cloud and edge networks such as enterprises and 5G edges. Azure has thousands of paths to choose from and hundreds of thousands of user groups to simultaneously optimize for [71]. This difference in focus and scale introduces different challenges that Tango does not address (§2.4).

The IETF's Path Aware Networking research group proposed several ingress traffic engineering solutions [46]—PAINTER incorporates those lessons (*e.g.,* immediate deployability) into its design. Masque is an IETF effort which could enable performance/security-enhancing QUIC proxies [48]; PAINTER uses "proxies" but to enhance route diversity. Emerging/futuristic routing technologies [6, 109] or architectures [55] could facilitate ingress traffic engineering. Miro also exposes more paths and tunnels traffic over these paths [108], but relies on a proposed extension to BGP that requires adoption from every ISP in the path, making deployment more challenging. PAINTER is designed to exist in today's Internet with no cooperation from intermediate ISPs.

Client-Side Reliability. Systems exist for measuring client performance at scale [18, 22]. PAINTER both measures *and* solves networking-related performance problems but, unlike prior work, cannot measure application-layer issues.

7 CONCLUSIONS

PAINTER lowers latency and enhances resilience for cloud services. It is widely deployable due to the growing prevalence of edge proxies. By enbracing new network management trends PAINTER simultaneously enhances control and solves problems that clouds face today. We see PAINTER as the first of many such systems that will define tomorrow's Internet.

Acknowledgements. This paper has been partially funded by NSF CNS-1835253. We would like to thank the anonymous reviewers for their insightful comments. We would also like to thank Arpit Gupta, Sanjay Chandrasekaran, Mahshid Ghasemi, Gil Zussman, and the Columbia University networking IT staff for their help in gathering residential campus data used in Section 2.2. Finally, we thank Abhishek Udupa for ideas and feedback on an early version of the project and Italo Cunha for his support with PEERING.

REFERENCES

[1] 3GPP. 2020. System Architecture for the 5G System (5GS). https://3gpp.org/ftp//Specs/archive/23_series/23.501/23501-g60.zip

[2] Akamai. 2022. Akamai Secure Access Service Edge. https://akamai.com/resources/akamai-secure-access-service-edge-sase

[3] Aditya Akella, Srinivasan Seshan, and Anees Shaikh. Multihoming Performance Benefits: An Experimental Evaluation of Practical Enterprise Strategies. In *NSDI 2004*.

[4] Mark Allman. Putting DNS in Context. In *IMC 2020*.

[5] Amazon. 2022. Simplify SD-WAN Connectivity With AWS Transit Gateway Connect. https://aws.amazon.com/blogs/networking-and-content-delivery/simplify-sd-wan-connectivity-with-aws-transit-gateway-connect/

[6] Maria Apostolaki, Ankit Singla, and Laurent Vanbever. Performance-Driven Internet Path Selection. In *SOSR 2021*.

[7] Apple. 2020. Improving Network Reliability Using Multipath TCP. https://developer.apple.com/documentation/foundation/urlsessionconfiguration/improving_network_reliability_using_multipath_tcp

[8] Apple. 2021. Configuring Network Extensions. https://developer.apple.com/documentation/xcode/configuring-network-extensions

[9] Todd Arnold, Jia He, Weifan Jiang, Matt Calder, Italo Cunha, Vasileios Giotsas, and Ethan Katz-Bassett. Cloud Provider Connectivity in the Flat Internet. In *IMC 2020*.

[10] Hirochika Asai and Yasuhiro Ohara. Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup. In *SIGCOMM 2015*.

[11] Florian Aschenbrenner, Tanya Shreedhar, Oliver Gasser, Nitinder Mohan, and Jörg Ott. From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet. In *IFIP Networking Conference 2021*.

[12] ATT. 2021. ATT Moves 5G Mobile Network to Microsoft Cloud. https://about.att.com/story/2021/att_microsoft_azure.html

[13] Arati Baliga, Xu Chen, Baris Coskun, Gustavo de los Reyes, Seungjoon Lee, Suhas Mathur, and Jacobus E Van der Merwe. VPMN: Virtual Private Mobile Network Towards Mobility-as-a-Service. In *MCS 2011*.

[14] Barracuda. 2023. Accelerate Your Business With Secure SD-WAN. https://barracuda.com/products/network-security/sd-wan

[15] BGP.us. 2016. Full View or Not Full View: The Benefits and Dangers of the Full BGP Table. https://bgp.us/routing-table/full-bgp-table-benefits-and-dangers

[16] Protick Bhowmick, Mohammad Ishtiaq Ashiq Khan, Casey Deccio, and Taejoong Chung. TTL Violation of DNS Resolvers in the Wild. In *PAM 2023*.

[17] Henry Birge-Lee, Maria Apostolaki, and Jennifer Rexford. It Takes Two to Tango: Cooperative Edge-to-Edge Routing. In *HotNets 2022*.

[18] Sam Burnett, Lily Chen, Douglas A. Creager, Misha Efimov, Ilya Grigorik, Ben Jones, Harsha V. Madhyastha, Pavlos Papageorge, Brian Rogan, Charles Stahl, et al. Network Error Logging: Client-Side Measurement of End-to-End Web Service Reliability. In *NSDI 2020*.

[19] CAIDA. 2023. BGP Stream. https://bgpstream.caida.org/data

[20] Matt Calder, Xun Fan, and Liang Zhu. A Cloud Provider's View of EDNS Client-Subnet Adoption. In *TMA 2019*.

[21] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the Performance of an Anycast CDN. In *IMC 2015*.

[22] Matt Calder, Ryan Gao, Manuel Schröder, Ryan Stewart, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System. In *NSDI 2018*.

[23] Fangfei Chen, Ramesh K. Sitaraman, and Marcelo Torres. End-User Mapping: Next Generation Request Routing for Content Delivery. In *SIGCOMM 2015*.

[24] Hao Chen, Yuan Yang, Mingwei Xu, Yuxuan Zhang, and Chenyi Liu. Neurotrie: Deep Reinforcement Learning-Based Fast Software IPv6 Lookup. In *ICDCS 2022*.

[25] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. Characterizing IPv4 Anycast Adoption and Deployment. In *CoNEXT 2015*.

[26] Cisco. 2023. Cisco SD-WAN. https://cisco.com/c/en/us/solutions/enterprise-networks/sd-wan/index.html

[27] Cloudflare. 2022. Argo Smart Routing. https://cloudflare.com/products/argo-smart-routing/

[28] Gerald Combs. 2020. TShark. https://wireshark.org/docs/man-pages/tshark.html

[29] The Khang Dang, Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Jörg Ott, and Jussi Kangasharju. Cloudy With a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet. In *IMC 2021*.

[30] Danny Pinto. 2021. What Will Happen When the Routing Table Hits 1024K? https://blog.apnic.net/2021/03/03/what-will-happen-when-the-routing-table-hits-1024k/

[31] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and Evaluation. In *CoNEXT 2017*.

[32] Wouter B. De Vries, Ricardo de O. Schmidt, Wes Hardaker, John Heidemann, Pieter-Tjerk de Boer, and Aiko Pras. Broad and Load-Aware Anycast Mapping with Verfploeter. In *IMC 2017*.

[33] Anja Feldmann, Oliver Gasser, Franziska Lichtblau, Eric Pujol, Igmar Poese, Christoph Dietzel, Daniel Wagner, Matthias Wichtlhuber, Juan Tapiador, Narseo Vallina-Rodriguez, et al. Implications of the COVID-19 Pandemic on the Internet Traffic. In *IMC 2020*.

[34] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs. In *NSDI 2015*.

[35] Ashley Flavel, Pradeepkumar Mani, and David A Maltz. Re-Evaluating the Responsiveness of DNS-Based Network Control. In *LANMAN 2014*.

[36] flexiWAN. 2022. flexiWAN Documentation. https://docs.flexiwan.com/

[37] Lixin Gao and Jennifer Rexford. 2001. Stable Internet Routing Without Global Coordination. *ToN* (2001).

[38] Ruomei Gao, Constantinos Dovrolis, and Ellen W Zegura. Avoiding Oscillations Due to Intelligent Route Control Systems. In *INFOCOM 2006*.

[39] Petros Gigis, Matt Calder, Lefteris Manassakis, George Nomikos, Vasileios Kotronis, Xenofontas Dimitropoulos, Ethan Katz-Bassett, and Georgios Smaragdakis. Seven Years in the Life of Hypergiants' Off-Nets. In *SIGCOMM 2021*.

[40] Danilo Giordano, Danilo Cicalese, Alessandro Finamore, Marco Mellia, Maurizio Munafò, Diana Zeaiter Joumblatt, and Dario Rossi. A First Characterization of Anycast Traffic from Passive Traces. In *TMA 2016*.

[41] Google. 2021. Network Connectivity Center. https://cloud.google.com/network-connectivity-center

[42] HashiCorp. 2022. Architecting Geo-Distributed Mobile Edge Application With Consul. https://www.youtube.com/watch?v=At6cHrL6sOQ

[43] Geoff Huston. 2023. BGP Routing Table Analysis Reports. https://bgp.potaroo.net

[44] IBM. 2022. IBM SevOne Network Performance Management. https://ibm.com/products/sevone-network-performance-management

[45] IETF. 2011. Architectural Guidelines for Multipath TCP Development. https://datatracker.ietf.org/doc/rfc6182/

[46] IETF. 2017. Path Aware Network RG. https://datatracker.ietf.org/rg/panrg/about/

[47] IETF. 2020. Multipath Extension for QUIC. https://datatracker.ietf.org/doc/html/draft-liu-multipath-quic-02

[48] IETF. 2022. IP Proxying Support for HTTP. https://datatracker.ietf.org/doc/draft-ietf-masque-connect-ip/

[49] INAP. 2022. INAP Network Connectivity. https://inap.com/network/

[50] Mordor Intelligence. 2022. Network as a Service Market - Growth, Trends, COVID-19 Impact, and Forecasts. https://mordorintelligence.com/industry-reports/network-as-a-service-market-growth-trends-and-forecasts

[51] Junchen Jiang, Rajdeep Das, Ganesh Ananthanarayanan, Philip A Chou, Venkata Padmanabhan, Vyas Sekar, Esbjorn Dominique, Marcin Goliszewski, Dalibor Kukoleca, Renat Vafin, et al. Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection. In *SIGCOMM 2016*.

[52] Yuchen Jin, Colin Scott, Amogh Dhamdhere, Vasileios Giotsas, Arvind Krishnamurthy, and Scott Shenker. Stable and Practical AS Relationship Inference with ProbLink. In *NSDI 2019*.

[53] Hyojoon Kim and Arpit Gupta. ONTAS: Flexible and Scalable Online Network Traffic Anonymization System. In *Workshop on Network Meets AI & ML 2019*.

[54] Thomas Koch, Ethan Katz-Bassett, John Heidemann, Matt Calder, Calvin Ardi, and Ke Li. Anycast in Context: A Tale of Two Systems. In *SIGCOMM 2021*.

[55] Cyrill Krähenbühl, Seyedali Tabaeiaghdaei, Christelle Gloor, Jonghoon Kwon, Adrian Perrig, David Hausheer, and Dominik Roos. Deployment and Scalability of an Inter-Domain Multi-Path Routing Infrastructure. In *CoNEXT 2021*.

[56] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving Beyond End-to-End Path Information to Optimize CDN Performance. In *IMC 2009*.

[57] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2001. Delayed Internet Routing Convergence. *ToN* (2001).

[58] Raul Landa, Lorenzo Saino, Lennert Buytenhek, and João Taveira Araújo. Staying Alive: Connection Path Reselection at the Edge. In *NSDI 2021*.

[59] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Internet Anycast: Performance, Problems, & Potential. In *SIGCOMM 2018*.

[60] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. Efficiently Delivering Online Services over Integrated Infrastructure. In *NSDI 2016*.

[61] Matthew Luckie, Bradley Huffaker, Amogh Dhamdhere, Vasileios Giotsas, and KC Claffy. AS Relationships, Customer Cones, and Validation. In *IMC 2013*.

[62] Matthew Luckie, Bradley Huffaker, Alexander Marder, Zachary Bischof, Marianne Fletcher, and KC Claffy. Learning to Extract Geographic Information from Internet Router Hostnames. In *CoNEXT 2021*.

[63] Andra Lutu, Diego Perino, Marcelo Bagnulo, Enrique Frias-Martinez, and Javad Khangosstar. A Characterization of the COVID-19 Pandemic Impact on a Mobile Network Operator Traffic. In *IMC 2020*.

[64] Michael Markovitch, Sharad Agarwal, Rodrigo Fonseca, Ryan Beckett, Chuanji Zhang, Irena Atov, and Somesh Chaturmohta. TIPSY: Predicting Where Traffic Will Ingress a WAN. In *SIGCOMM 2022*.

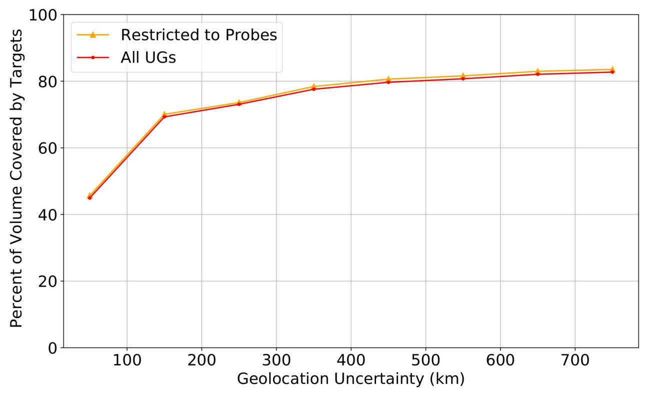
[65] Maxmind. 2022. GeoIP2 Databases. https://maxmind.com/en/geoip2-databases

[66] Stephen McQuistin, Sree Priyanka Uppu, and Marcel Flores. Taming Anycast in the Wild Internet. In *IMC 2019*.

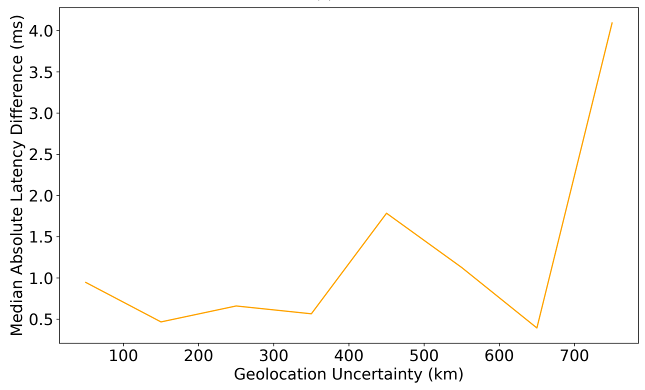
[67] Megaport. 2023. Agile Networking for Real-Time IT Transformation. https://megaport.com

[68] Microsoft. 2022. SD-WAN Connectivity Architecture with Azure Virtual WAN. https://learn.microsoft.com/en-us/azure/virtual-wan/sd-wan-connectivity-architecture

[69] Microsoft. 2023. Microsoft Azure Private 5G Core. https://azure.microsoft.com/en-us/products/private-5g-core
[70] Microsoft. 2023. Microsoft Datacenters. https://datacenters.microsoft.com/
[71] Microsoft. 2023. Microsoft Global Network. https://learn.microsoft.com/en-us/azure/networking/microsoft-global-network
[72] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. Pruning Edge Research With Latency Shears. In *HotNets 2020*.
[73] Giovane C. M. Moura, John Heidemann, Ricardo de O Schmidt, and Wes Hardaker. Cache Me If You Can: Effects of DNS Time-to-Live. In *IMC 2019*.
[74] Giovane C. M. Moura, Ricardo de Oliveira Schmidt, John Heidemann, Wouter B. de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *IMC 2016*.
[75] NANOG. 2022. Panel: Buying and Selling IPv4 Addresses. https://youtube.com/watch?v=8FlTJEct9_s
[76] Srinivas Narayana, Wenjie Jiang, Jennifer Rexford, and Mung Chiang. Joint Server Selection and Routing for Geo-Replicated Services. In *International Conference on Utility and Cloud Computing 2013*.
[77] Nokia. 2022. Nokia 5G Core Software as a Service in Practice. https://www.youtube.com/watch?v=_4DHzjrtB34
[78] Larry Peterson and Oğuz Sunay. 2020. 5G Mobile Networks: A Systems Approach. *Synthesis Lectures on Network Systems* (2020).
[79] Matthew Prince. 2013. Load Balancing without Load Balancers. https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/
[80] Enric Pujol, Ingmar Poese, Johannes Zerwas, Georgios Smaragdakis, and Anja Feldmann. Steering Hyper-Giants' Traffic at Scale. In *CoNEXT 2019*.
[81] RIPE. 2022. RIPE IPmap. https://ipmap.ripe.net/
[82] ASM Rizvi, Leandro Bertholdo, João Ceron, and John Heidemann. Anycast Agility: Network Playbooks to Fight DDoS. In *USENIX Security Symposium 2022*.
[83] Corey Satten. 2008. Lossless Gigabit Remote Packet Capture with Linux. https://staff.washington.edu/corey/gulp/
[84] Patrick Sattler, Juliane Aulbach, Johannes Zirngibl, and Georg Carle. Towards a Tectonic Traffic Shift? Investigating Apple's New Relay Network. In *IMC 2022*.
[85] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *CoNEXT 2019*.
[86] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. Internet Performance from Facebook's Edge. In *IMC 2019*.
[87] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *SIGCOMM 2017*.
[88] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. Cost-Effective Cloud Edge Traffic Engineering with Cascara. In *NSDI 2021*.
[89] Raffaele Sommese, Leandro Bertholdo, Gautam Akiwate, Mattijs Jonker, Roland van Rijswijk-Deij, Alberto Dainotti, KC Claffy, and Anna Sperotto. MAnycast2: Using Anycast to Measure Anycast. In *IMC 2020*.
[90] Harrison J. Son. 2017. Comparison of the SD-WAN Vendor Solutions. https://netmanias.com/en/post/oneshot/12481/sd-wan-sdn-nfv/comparison-of-the-sd-wan-vendor-solutions
[91] Austin Spreadbury and Nicole Singh. 2023. Azure Operator Voicemail: Take the First Step to Move Voice Workloads to the Cloud. https://techcommunity.microsoft.com/t5/azure-for-operators-blog/azure-operator-voicemail-take-the-first-step-to-move-voice/ba-p/3751315
[92] Neil Spring, Ratul Mahajan, and Thomas Anderson. The Causes of Path Inflation. In *Applications, technologies, architectures, and protocols for computer communications 2003*.
[93] RIPE NCC Staff. 2015. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal* (2015).
[94] RIPE NCC Staff. 2023. RIS Live. (2023). https://ris-live.ripe.net
[95] Thibaut Stimpfling, Normand Belanger, JM Pierre Langlois, and Yvon Savaria. 2019. SHIP: A Scalable High-Performance IPv6 Lookup Algorithm that Exploits Prefix Characteristics. *ToN* (2019).
[96] Subspace. 2022. Optimize Your Network on Subspace. https://subspace.com/solutions/reduce-internet-latency
[97] Peng Sun, Laurent Vanbever, and Jennifer Rexford. Scalable Programmable Inbound Traffic Engineering. In *SOSR 2015*.
[98] Tessares. 2019. Introducing a Client Library for 0-RTT Converter. https://tessares.net/open-source/introduction-a-client-library-for-0-rtt-converter/
[99] Zartash Afzal Uzmi, Markus Nebel, Ahsan Tariq, Sana Jawad, Ruichuan Chen, Aman Shaikh, Jia Wang, and Paul Francis. SMALTA: Practical and Near-Optimal FIB Aggregation. In *CoNEXT 2011*.
[100] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C Snoeren. Quantifying the Benefits of Joint Content and Network Routing. In *SIGMETRICS 2013*.
[101] Verizon. 2020. Edgecast. https://verizondigitalmedia.com/media-platform/delivery/network/
[102] VMware. 2023. VMware SD-WAN. https://sase.vmware.com/sd-wan
[103] VULTR. 2023. VULTR Cloud. https://vultr.com/
[104] Chengke Wang, Hao Wang, Feng Qian, Kai Zheng, Chenglu Wang, Fangzhu Mao, Xingmin Guo, and Chenren Xu. Experience: A Three-Year Retrospective of Large-Scale Multipath Transport Deployment for Mobile Applications. In *MobiCom 2023*.
[105] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. . A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance. ([n. d.]).
[106] Kaiqiang Wang, Minwei Shen, Junguk Cho, Arijit Banerjee, Jacobus Van der Merwe, and Kirk Webb. MobiScud: A Fast Moving Personal Cloud in the Mobile Network. In *AllThingsCellular 2015*.
[107] Jun Xu, Jinliang Fan, Mostafa H. Ammar, and Sue B Moon. Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme. In *ICNP 2002*.
[108] Wen Xu and Jennifer Rexford. MIRO: Multi-Path Interdomain Routing. In *SIGCOMM 2006*.
[109] Xiaowei Yang, David Clark, and Arthur W Berger. 2007. NIRA: A New Inter-Domain Routing Architecture. *ToN* (2007).
[110] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *SIGCOMM 2017*.
[111] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, and Xiaowei Yang. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *SIGCOMM 2021*.
[112] Yang Zhang, Jean Tourrilhes, Zhi-Li Zhang, and Puneet Sharma. 2021. Improving SD-WAN Resilience: From Vertical Handoff to WAN-Aware MPTCP. *ToN* (2021).
[113] Zheng Zhang, Ming Zhang, Albert G. Greenberg, Y. Charlie Hu, Ratul Mahajan, and Blaine Christian. Optimizing Cost and Performance in Online Service Provider Networks. In *NSDI 2010*.
[114] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. Xlink: QoE-Driven Multi-Path QUIC Transport in Large-Scale Video Services. In *SIGCOMM 2021*.
[115] Minyuan Zhou, Xiao Zhang, Shuai Hao, Xiaowei Yang, Jiaqi Zheng, Guihai Chen, and Wanchun Dou. Regional IP Anycast: Deployments, Performance, and Potentials. In *SIGCOMM 2023*.
[116] Jiangchen Zhu, Kevin Vermeulen, Italo Cunha, Ethan Katz-Bassett, and Matt Calder. The Best of Both Worlds: High Availability CDN Routing Without Compromising Control. In *IMC 2022*.



(a)



(b)

Figure 12: Coverage of policy-compliant ingresses at various geolocation uncertainties (12a) and median absolute difference in estimated and actual latency (12b).

Appendices are supporting material that has not been peer-reviewed.

A RESIDENTIAL NETWORK DATA

To demonstrate the limited ability of clouds to quickly redirect network traffic due to the prevalence of DNS TTL violations, we passively collected traffic from 12 residential buildings managed by Columbia University. One building accommodates returning and nontraditional students, while the other buildings house graduate students, faculty, staff, and their families. All the buildings have 20-50 private or shared apartments, and approximately 400 rental units are in the dataset. The university serves as the residents' ISP, and they share the set of recursive DNS resolvers by default.

To protect user privacy, our data collection process adheres to existing anonymization best practices [53]—it anonymizes privacy-sensitive fields (*e.g.,* MAC addresses and IP addresses) and discards the payload at collection time. We use Gulp to capture the traffic [83], tshark to extract non-sensitive information (*e.g.,* protocol, TLS SNI, DNS A record) [28], and cryptoANT to anonymize privacy-sensitive fields [107]. We discard the payload above layer 4 except for TLS fields and DNS packets. Cryptographic anonymization keys are rotated every six hours. The privacy and security team of the IT department at Columbia University thoroughly reviewed and approved the data collection process, and our Institutional Review Board (IRB) declared that our project is not human-subjects research and does not require further review.

To obtain our results regarding DNS dynamics (§2.2), we passively captured all traffic sent to and from residential units during 10-11 am and all DNS traffic from 5-11 am during December 1-23, 2022. Packets with the same 5-tuple in the IP header (transport protocol, source IP, destination IP, source port, destination port) in a 6 hour window are considered as a single flow, and flows are associated with the latest DNS record that was transmitted to the same residential IP and included the destination IP in the DNS response. Since only 0.06% of flows are paired with DNS records between 5-6am and extending the collection period does not notably increase the number of matched flows, we limited our use of DNS records to 6 hours to match an one-hour capture of network traffic. Our methodology for matching traffic to DNS records is similar to that in prior work [4].

B INGRESS LATENCY ESTIMATION

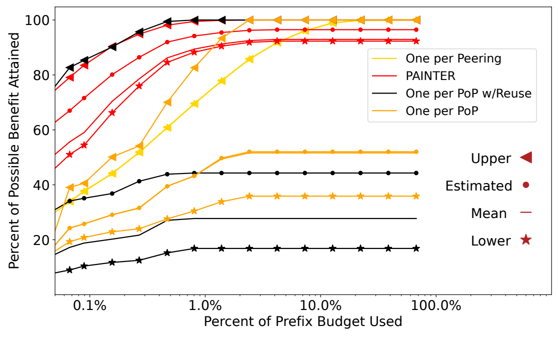
`Azure` connects to other organizations at over 200 PoPs around the world [70]. We wished to estimate latency to `Azure` through specific ingresses, as we could not conduct advertisements from `Azure` for operational reasons. The key idea of our methodology was to estimate the latency through an ingress as the latency to an IP address in the peer/provider's IP space physically close to the corresponding ingress. Here we thoroughly explain our target-determination methodology, estimate how accurate our ingress latency approximation heuristic is, and describe how we chose allowable target geolocation uncertainty.

Methodology. `Azure`'s peerings are often between two physical interfaces on peering routers. Oftentimes, these interfaces are assigned IP addresses belonging to either an `Azure` or peer/provider's subnet (each case occurs roughly half the time). We use the latency to the interface's IP address as a proxy for latency through that ingress if the address was in the peer/provider's IP space. We could not target IP addresses in `Azure`'s IP space since `Azure` advertises covering prefixes for these addresses at all PoPs (hence the route is the same as the anycast route). In this way, we were only able to obtain target addresses for fewer than 500 ingresses as many addresses were unresponsive.

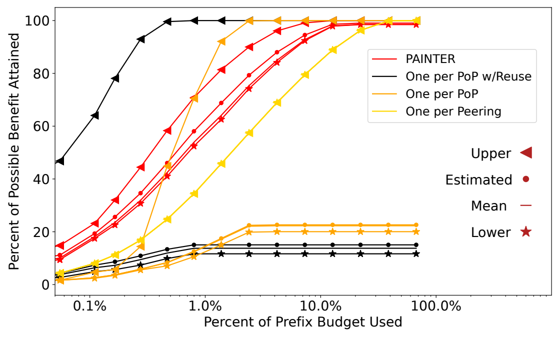
To find targets for the remaining (vast majority of) ingresses, we obtained lists of candidate IP addresses by (a) crawling RIPE Open IPMap [81] and Maxmind [65], (b) parsing `Azure` traceroutes from clients, and (c) using RDNS hints (specifically using Hoiho [62]). We exclude addresses known to be anycast according to a publicly available list [89]. We confirmed target locations using measurements from RIPE Atlas probes with known locations. The geolocation uncertainty of each target was then the minimum latency from any RIPE Atlas probe to the address converted to distance in fiber, plus the distance from the RIPE Atlas probe to the associated ingress's PoP. We conducted a secondary check that targets were not anycast by checking for speed of light violations when measuring to targets from several RIPE Atlas probes.

We were only able to find a subset of ingress targets at a given geolocation uncertainty due to the limitations (*i.e.,* coverage) of our geolocation methodology. To quantify how representative ingress targets were for a given geolocation uncertainty, we tabulated all ⟨`UG, ingress`⟩ tuples for which a path from the `UG` through the ingress could be policy-compliant (§3.1) and for which an ingress target was geolocated to within the uncertainty.

For the purposes of determining our coverage, we excluded ⟨`UG, ingress`⟩ tuples which were unlikely to provide latency benefits to that `UG`. We say an ingress at a PoP is unlikely to provide latency benefits if the anycast latency from that `UG` is lower than



(a)



(b)

Figure 14: Range of latency benefits for each strategy over prefix budget, computed using real (14a) and simulated (14b) measurements.

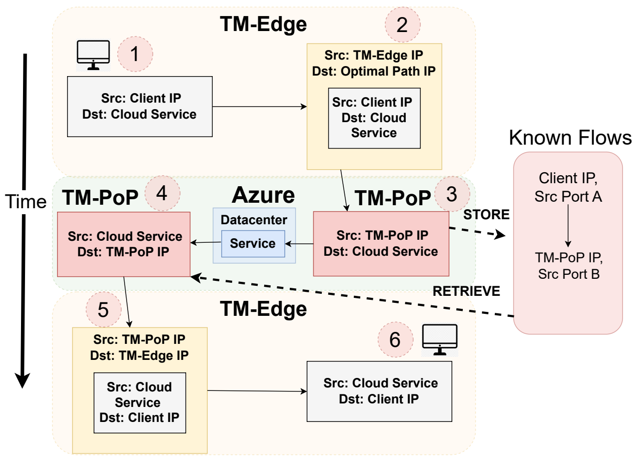


Figure 13: Packet journey through PAINTER. TM-Edge tunnels client traffic along different ingress paths (1-2), TM-PoP NATs client traffic (3-4), and sends the traffic back to TM-Edge who forwards it to the client (5-6).

the distance from the UG to the PoP converted to the speed of light in fiber (*i.e.,* the best possible latency to the PoP). For example, we do not count policy-compliant ingresses at Azure's Chicago PoP for a UG in Ashburn (900 km) towards our coverage if that UG had anycast latency < 9 ms (speed of light in a direct fiber path). To calculate our coverage metric we divide each UG's traffic volume evenly among its possible ingresses and accumulate volume over ingresses for which we have a target.

In addition to ensuring we had broad coverage of ingresses, assessed the accuracy of our latency estimates by comparing estimated latencies to actual latencies through the corresponding peerings to Azure for a subset of cases. We conducted traceroutes from RIPE Atlas probes to Azure's anycast address and tabulated a list of ⟨probe, peering⟩ tuples corresponding to cases where we observed a known peering connection's IP address in the traceroute. We compared the minimum latencies reported by traceroutes to our estimated latencies. We bucket absolute latency differences by geolocation uncertainty and in Figure 12b plot the median difference for each bucket.

Results. In Figure 12a we show our coverage of ingresses (weighted according to the above metric) at different maximum geolocation uncertainties. In Figure 12b we show how ingress target latencies compared to actual latencies through those ingresses at different geolocation geolocation uncertainties.

Figure 12a demonstrates that we geolocate more targets, and thus cover significantly more ingresses, as we admit less precise target geolocation. In particular, the "knee" of the curve is at around a geolocation accuracy of 400 km. Our target coverage when restricting to RIPE Atlas probes and when considering all UGs are similar, likely since RIPE Atlas probes tend to be in UGs that generate lots of Azure traffic volume.

Figure 12b shows the median absolute difference between our targets and the actual latency to Azure. Figure 12b demonstrates that latencies agree as we require more geolocation accuracy in our ingress targets. We chose to use targets within an uncertainty of 450 km for our evaluations since this uncertainty gives a nice tradeoff between coverage (80.6% of Azure volume) and methodological accuracy (median comparisons within 2 ms).

Close inspection revealed that disagreements in latencies at low geographic uncertainty are likely due to inflation inside the peer/provider's AS to Azure specifically or on the reverse path from Azure to the probe—*i.e.,* the path to our ingress target was direct and low latency, whereas the path to/from Azure was circuitous. Hence, these cases still indicate room for latency improvement although realizing those improvements might not be feasible through advertisements to peers/providers if they route Azure traffic inefficiently.

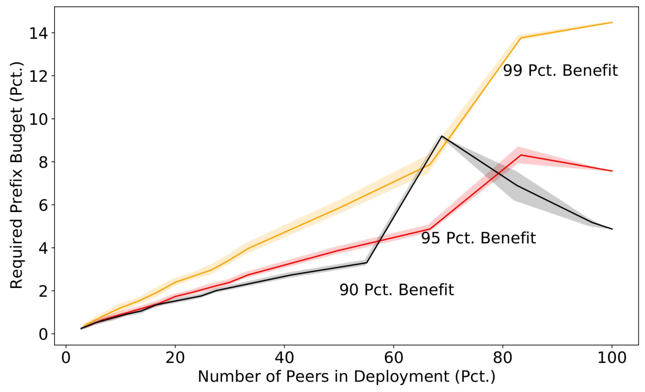
C SIMULATING MEASUREMENTS FROM UGs

To evaluate the Advertisement Orchestrator on a much larger set of UGs than are represented by RIPE Atlas probes, we simulate measurements from UGs for which there is no available RIPE Atlas probe. We consider the set of UGs that represent 99% of Azure traffic volume, as this significantly reduces computational complexity (*i.e.,* the number of UGs we need to consider).

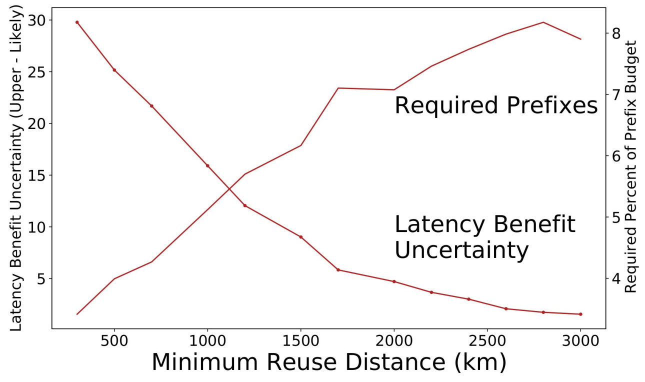
We first tabulate the set of all policy-compliant ingresses from UGs to Azure through Azure's peerings. Then, for each UG, we find all RIPE Atlas probes within 500 km of the UG whose median anycast latency to Azure is within 10 ms of the UG's anycast latency. We determine anycast latencies from Azure's global measurement system [22].

Finally, we take the union of all improvements these RIPE Atlas probes saw along all their policy-compliant ingresses as a set of 'representative improvements'. For each policy-compliant ingress for the UG in question, we then randomly draw improvements over anycast from this set of representative improvements.

As an example, assume a UG in East US is physically close to a RIPE Atlas probe and experiences similar anycast latency to Azure. Assume this probe sees ingress latencies relative to anycast of -5 ms,



(a)



(b)

Figure 15: PAINTER scales linearly in the number of required prefixes with deployment size (15a), and naturally provides a tradeoff between benefit uncertainty and prefix cost via a tunable parameter (D_{reuse}) (15b).

0 ms, 0 ms, and 3 ms. Then, hypothetical latencies from the UG along its (possibly different) policy-compliant ingresses will be randomly drawn from this distribution. Intuitively, probes in areas with "good" routing (*i.e.,* little improvement over anycast) will induce simulated measurements for nearby UGs with "good" routing while areas with bad routing will induce simulated measurements with bad routing in those areas.

D TRAFFIC MANAGER TUNNELING MECHANISM

The Traffic Manager uses a tunneling mechanism similar to some used by SD-WAN solutions [26] since it requires no endpoint modification.

We show a typical packet's journey through PAINTER using our tunneling mechanism in Figure 13. Packets generated by clients reach TM-Edge (1), which encapsulates traffic in UDP datagrams and sets the destination IP address of the outer packet according to the optimal path to Azure (2) (§3.2). TM-PoPs decapsulate traffic arriving at the other end of the optimal path. TM-PoP NATs the traffic, storing the client's source port and IP address in a lookup table ('Known Flows') to retrieve later (3). TM-PoP acts as a NAT to ensure return traffic goes back through the tunnel (not directly to the client). TM-PoP receives response traffic from services (4) and, using the lookup table, replaces the destination address with the corresponding client IP address. TM-PoP then re-encapsulates response traffic and sends it to the corresponding TM-Edge (5) which then decapsulates the traffic and forwards it to the client (6). Each TM-PoP has multiple IP addresses/NICs and so handles 65k connections for each IP address, spread across all TM-Edges.

Scaling to Azure. TM-Edge performs minimal operations on packets, looking up optimal destinations for flows and encapsulating packets so as to route traffic toward these destinations. The added overhead of the UDP header (approximately 16 bytes per 1400) is a small price to pay for the performance improvements PAINTER provides. PAINTER scales linearly with the number of edge proxies since TM-Edges only communicate directly to other TM-PoPs. PAINTER similarly scales with the number of PoPs since there need only be one TM-PoPs per PoP.

E ADVERTISEMENT ORCHESTRATOR FURTHER ANALYSIS

E.1 Benefit Ranges over Budget

In Section 5.1 we showed *estimated* latency benefits for each advertisement strategy. In Figure 14, we explicitly show the entire range of possible benefits from a 'Lower' to an 'Upper' bound. The 'Mean' line corresponds to an unweighted average across all possible ingresses, whereas the 'Estimated' corresponds to a weighted average, where the weights assume inflated paths are less likely (§5.1). The set of possible ingresses for a UG corresponds to the set of policy-compliant ingresses over which the Advertisement Orchestrator advertises the prefix the UGs selects. UGs select the highest 'Mean' prefix over all their prefix choices (Eq. (2)).

One per PoP strategies have very large ranges of possible benefits since they advertise prefixes via all peerings at PoPs, and so expose many (possibly poor) valid ingresses for UGs. Hence, these strategies tend to quickly achieve high Upper performance bounds (since they quickly make all ingresses possible to reach, in theory), but do not provide high Mean or Estimated benefits since UGs may be routed to worse ingresses. In practice, larger ranges mean that some UGs will be routed optimally while some will not, as was observed in prior work [21].

PAINTER's intelligent prefix reuse from far-away PoPs and via peerings with non-overlapping customer cones allows it to quickly achieve most latency benefit with little uncertainty. The One per Peering strategy has no uncertainty since it advertises a unique prefix via each peering, but it uses at least 3× the number of prefixes as PAINTER to yield the same latency benefit.

E.2 Scale and Uncertainty

Figure 15 summarizes the Advertisement Orchestrator's convergence properties with respect to two parameters: deployment size and the minimum reuse distance (D_{reuse}). Figure 15a shows that the required number of prefixes to obtain various percent benefits scales linearly with deployment size, so we can expect system overhead to grow proportionally with Azure growth. Hence PAINTER's required resources to calculate and deploy latency-improving advertisements scale with the many peerings that several clouds have [9], unlike prior work [100, 111].

Figure 15b shows that increasing D_{reuse} leads to less uncertainty about user benefit, but requires more prefixes to obtain benefit for users. As more uncertainty may lead to more incorrect latency prediction heuristics, decreasing D_{reuse} may require more learning iterations for convergence in practice.

To generate this figure, we calculate PAINTER solutions over a range of D_{reuse} and calculate (a) how many prefixes PAINTER needs to obtain 99% of the benefit (upper range) to quantify solution cost and (b) the difference between the upper and estimated ranges in Figure 6a at 99% benefit (upper range) to quantify benefit uncertainty. As we make more "reasonable" assumptions about path inflation (increasing D_{reuse}), our uncertainty about user benefit decreases. We use $D_{reuse} = 3{,}000$ in Figure 6a as it provides a decent tradeoff between uncertainty and cost. This is in contrast to other solutions (*e.g.,* One per PoP) which provide no way to control benefit uncertainty (and have quite high uncertainty).