Optimizing Interdomain Routing for the Services of Today and Tomorrow

Thomas Koch

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2024

Abstract

Optimizing Interdomain Routing for the Services of Today and Tomorrow

Thomas Koch

Large cloud and content Service Providers serve applications that are responsible for the vast majority of Internet traffic today. However, Service Providers have to contend with decades-old Internet protocols to do so and, in particular, have to route latency-sensitive user traffic over the unreliable public Internet to Service Provider networks. This reliance creates urgent problems as businesses, people, and governments increasingly rely on the Internet for critical activities, and as new applications introduce increasingly strict network performance requirements.

This dissertation explores the extent to which both the current methods Service Providers use and the Internet's old protocols are sufficient to meet the demands of traditional, current, and future applications. We find that current methods satisfy requirements of traditional applications reasonably well, but do not offer sufficient guarantees for newer use cases. This dissertation then proposes using old Internet protocols in new ways to reliably route user traffic over an unreliable public Internet by solving challenging optimization problems using new Internet measurement and modeling techniques. The systems described in this dissertation can help Service Providers work with existing infrastructure to deliver the reliable, performant service our increasingly connected society needs.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This dissertation is a group effort, including both folks in my research community and folks in my personal life. I would first-and-foremost like to thank Ethan, my advisor, who provided countless hours of advice, provided an example of a great researcher, served as an example of a great leader, helped my professional and personal development, and always pushed me to seek excellence in everything I did.

I would also like to thank my lab mates: Todd, Loqman, Shuyue, Sang, Carson, and Jiangchen, who served as sounding boards, friends, and mentors. They pushed me to continue on during the difficult moments and encouraged me by their own example to produce great research.

I would of course also like to thank my mentors and collaborators, with special thanks to John Heidemann, Matt Calder, and Sharad Agarwal, whose insightful comments and direction led me to become a better researcher and helped us produce great research along the way.

I'd furthermore like to thank the students and mentees I have had the privilege of interacting with over the years. Teaching and mentoring has given me immense joy, and continues to be my passion.

I would finally like to thank my friends and family, especially those who recognized the value of pursuing my long-term vision and kept encouraging me to keep going. Writing a dissertation is a marathon, not a sprint. Its immensely helpful to have folks on the sidelines holding their encouraging signs, and reminding you about the fulfilling reward that waits just across the finish line.

Chapter 1: Introduction

Cloud/content providers (hereafter Service Providers) are increasingly involved in delivering most of the services, traffic, and use cases on the Internet today. Service Providers offer a variety of services to application developers (who then offer applications to users) and/or directly to users. These services include, but are not limited to: caching, web hosting, enterprise software, network functions, compute, storage, and ML model training/evaluation. The vast majority of Internet applications use one or more of these functions and so rely on one or more of these Service Providers. As Internet applications become an increasingly integral part of our lives (as painfully demonstrated by the COVID-19 pandemic), understanding and improving the way Service Providers deliver networked services similarly becomes increasingly important.

Internet applications require the network to perform/behave well, with better performance/behavior generally leading to richer applications — as network performance has gotten better, the class of applications has expanded and become more immersive. Different applications have vastly different performance (latency/throughput/reliability) requirements, which we broadly term Service-Level Objectives (`SLO`*s*). For example, web browsing (low immersion) benefits from 100 ms end-to-end latency, gaming (medium immersion) requires 50 ms, and VR (high immersion) requires 10 ms [1]. As Service Providers are integral in offering networked applications, Service Providers must offer their services in a way that is conducive to meeting `SLO`*s*.

One way in which Service Providers have continued to meet tighter `SLO`*s* for increasingly global application users is by building global-scale deployments. An example Service Provider deployment is shown in Figure 1.1. These deployments consist of data centers which provide the infrastructure to offer services (*e.g.,* processor cores, GPUs), sites which (among other things) serve as entry points into the Service Provider's network, and in-user-network ("edge") presences which serve many functions including caching. Some deployments also have a private WAN inter-

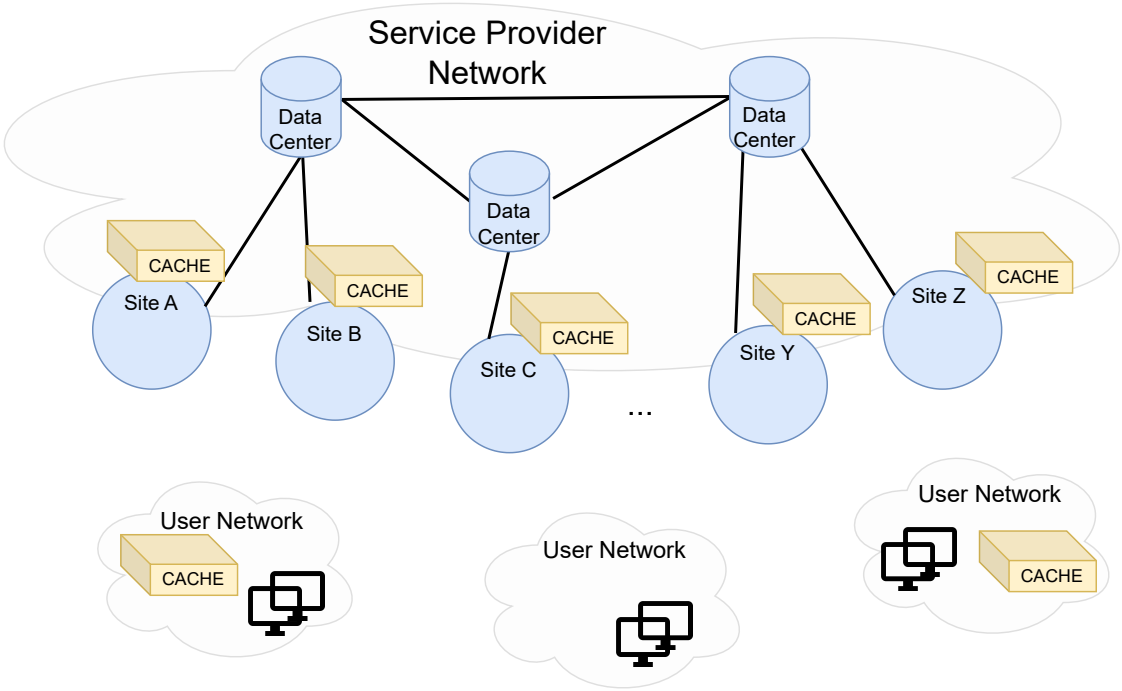


Figure 1.1: Service Providers have global deployments with tens of data centers and hundreds of sites.

connecting data centers and sites. These deployments often span hundreds of countries with tens to a hundred data centers interconnected by millions of miles of fiber-optic cable, tens to thousands of sites, and thousands of edge presences totalling investments of billions of dollars per Service Provider [2].

Service Provider network traffic can either be intradomain or interdomain network traffic, referring to whether traffic flows solely on the Service Provider's network or between the Service Provider's network and other networks through the public Internet. Traffic between two data centers is intradomain traffic, for example. Some Service Providers have built global systems for managing intradomain traffic on their private WANs, including systems that provide traffic engineering, improved utilization, resilience under failure, and physical layer optimizations [3, 4, 5]. Recent research suggests some benefits may also be possible for those without a private WAN [6].

Managing interdomain traffic requires solving a different set of challenges than in the intradomain case since part of the path lies outside of the control of the Service Provider, and so the Service Provider has less control over whether it can realize network objectives and, in particular,

meet SLO*s*. For example, Service Providers cannot directly control the path traffic takes when exiting its network; Service Providers have to use the Border Gateway Protocol (BGP) which only lets Service Providers control the next hop traffic takes. To combat these limitations, Service Providers have both strategically connected to more networks [7] and built complex systems on top of their own networks (which leverage those many connections) to give Service Providers more control [8, 9, 10].

Service Providers connect to more networks for both economic and performance-based reasons. Traditionally, Service Providers relied on global transit providers at their sites to deliver their traffic to and from users, since Service Providers lacked the infrastructure to deliver the traffic to users themselves. However, as the footprint of private WANs grew, Service Providers could (physically) connect to more networks without first going through a global transit provider. By peering with user networks (or providers of user networks), Service Providers cut out the middleman transit provider, thus significantly reducing costs. Connecting with more networks also offers Service Providers better performance. By connecting directly to user networks or regional providers of those networks, Service Providers acquire more control over a larger fraction of the (interdomain) path traffic takes [11]. This increased control has practical advantages, such as reducing latency from users to the Service Provider [11, 12].

Largely because of these advantages, Service Providers have connected with thousands of networks, often at many locations. For example, both clouds including Google, Microsoft, Amazon, and IBM, and content providers such as Cloudflare connect with thousands of networks [13, 7, 14]. Vultr cloud connects to more than 6,000 networks at 32 sites totalling more than 10,000 entry points (some networks connect at multiple sites). Service Providers use this connectivity to address the aforementioned challenges with managing interdomain traffic. For example, Service Providers use this connectivity to improve the performance of interdomain traffic destined to user networks by directing traffic over different paths/links according to performance objectives [8, 9, 10, 15].

However, many of the challenges associated with managing interdomain traffic are largely unsolved. Specifically, Service Providers have a poor understanding of how to control the part of

the path from when traffic leaves a user until it enters the Service Provider's network.

Problem: Service Providers need to meet a myriad of `SLO`*s* for different applications but need to use limited Internet protocols to interface with an unreliable, uncontrollable public Internet to route traffic to their reliable networks so as to meet those `SLO`*s*.

Service Providers *need* to use current Internet protocols, as different protocols are difficult to deploy. Deployment is difficult since changes to protocols must be made in most networks, so that Service Providers can serve their global populations. Researchers have proposed solutions that, if implemented, would solve our key problem. For example, clean-slate architectures such as the SCION network [16], MIRO [17], or collaborative relationships between Service Providers and end-user networks [18] offer alternatives to the setting we consider and do (at least partially) solve our key problem, but show no signs of widespread deployment. Hence we consider a general setting where the path user traffic takes from an application to the Service Provider's network is determined using predominant protocols: the Domain Name Service (DNS) protocol and the Border Gateway Protocol (BGP).

Ideally, Service Providers would be able to measure network conditions such as path latencies, link utilizations, and quality metrics at all layers (for example, physical link health), compute traffic-to-path allocations according to these measured network conditions, and finally choose the entire path their user traffic takes according to these computations, as in the intradomain setting [4, 3, 5]. However, part of the path lies in other networks, which makes it difficult to measure, compute, and control paths. We next discuss the challenges Service Providers face when computing and controlling traffic-to-path allocations in the interdomain setting, as the challenges of measuring interdomain network properties are already understood [19, 20, 21, 22, 23].

In the standard Internet setting that we consider, users access a service by querying (*i.e.,* resolving) a hostname either known to the user (*e.g.,* `www.google.com`) or specified by the application. This hostname is translated to an IP address via the Domain Name Service (DNS) protocol. The query is often sent from a client to a server who handles resolution on behalf of the client (and often thousands of other clients). The exact translation process depends on implementation

details in the user's application, the user's operating system, routers along the path, and/or the user's recursive resolver(s), which limits Service Provider control over how this translation occurs. For example, translations may be cached by the user's operating system long after they are considered invalid. The two parts of the process the Service Provider has control over are specifying the hostname itself and the answer to the DNS query returned to the recursive resolver by the Service Provider's authoritative resolver. Traffic towards this translated IP address then takes a path through the public Internet through both the user's ISP and potentially other networks. The path taken depends on decisions made in those intermediate networks (*i.e.,* via BGP).

BGP is a path-vector protocol that allows networks to exchange routes towards destinations on the Internet in the form of network-level paths towards IP prefixes. An IP prefix is a consecutive range of IP addresses. Networks establish sessions with each other over which they exchange information: they can advertise paths towards prefixes to other networks and receive other advertisements. In advertising a prefix, a network agrees to receive traffic destined towards that prefix (and propagate it along that route), and, when a network withdraws that advertisement, it cancels this agreement. Networks may receive many routes towards a prefix but advertise at most one route per prefix to other networks for scalability.

Today Service Providers rely primarily on two simple strategies for the translation from hostname to IP address. The first, `anycast`, is where an Service Provider translates every request for a hostname to an IP address in the same IP prefix and announces that IP prefix to all networks at all sites. BGP then decides the specific path users take to the Service Provider. `Anycast` is simple to implement and offers a natural resilience to failure: upon withdrawing an advertisement over one BGP connection with a network (*e.g.,* due to failure), BGP automatically computes another route to the prefix through another connection, often within tens of seconds [24]. Due to this simplicity and reliability, `anycast` is used by the majority of Service Providers today in some form.

`Anycast`, although simple, relinquishes nearly all control to other networks over how traffic routes to the Service Provider. Hence, some Service Providers use the second common strategy, `unicast`, where a distinct prefix is announced at each site. The same prefix is announced to all

connected networks at that site. By responding to DNS queries with an IP address in a specific prefix, Service Providers can direct traffic (for users using a certain recursive resolver) to a particular site. This enhanced control allows Service Providers to, for example, direct clients to a site with low latency. `Unicast` does not provide fast failover in general, however, since results can grow stale in DNS caches [25].

The extent to which these limitations are actual issues faced by Service Providers is not fully understood, as any one of these issues might rarely happen in practice. For example, some prior work reports that `anycast` introduces latency/reliability issues and should be replaced [26, 27, 28], or even that BGP should be fixed [27]. However, in addition to the fact that most Service Providers use `anycast` in some way today, other work looking at Microsoft's CDN and Google Public DNS suggests that `anycast` performs quite well [29, 30]. `Unicast` similarly works well by some metrics according to large CDNS [29, 31, 32, 8], but it remains unclear how big of an issue the use of stale DNS records can be. Prior DNS staleness investigations look at the caching properties of recursive resolvers (but not of clients) [33] and/or single applications [34].

Recent work proposes hybrids of `unicast` and `anycast` to overcome some of these limitations (again, the extent of which seem ill-understood) [29, 35, 36]. These solutions advertise different prefixes to subsets of all networks to offer users many low-latency options across different links/sites. We refer to these solutions as `selectivecast` solutions since they are *selective* about who they advertise prefix reachability to, and have traits of both `anycast` (advertising at many sites to many networks) and `unicast` (many prefixes, selectivity). However, it is unclear if these solutions fully solve the aforementioned problems, and, in some cases, they may introduce new problems. For example, `AnyOpt` does not scale to all Service Providers [35].

Proposed modifications to DNS or BGP could solve these problems but face deployment obstacles. For example, MIRO would allow networks to choose/propagate multiple paths via BGP [17], thus potentially giving more control to Service Providers over which routes are chosen and which routes are used. However it and other ideas requiring modifications to widely-used protocols or hardware have not seen deployment in nearly 20 years, as changing entrenched Internet proto-

cols/practices in billions of devices and tens of thousands of networks remains a hard problem.

Thesis statement: Existing approaches to interdomain routing suffice for traditional Internet services. To meet the more demanding `SLO`*s* of emerging applications, Service Providers can build systems that use limited Internet protocols in new ways to better meet `SLO`*s* by offering performant, resilient interdomain routes to users, and directing traffic on those routes, at the cost of additional deployment complexity and deployment infrastructure.

In this thesis, we first describe how we used global Service Provider traces to establish a better understanding of the current state-of-practice for interdomain routing. This analysis enhances our understanding of the limitations of both `anycast` and `unicast`, and specifically contextualizes the limitations for particular applications and use cases. We then propose a new `selectivecast` solution that provides better latency/reliability than `anycast`, `unicast`, and other `selectivecast` solutions, and can be deployed in more networks than other approaches. These benefits apply to a wide range of networks and Service Providers, but are particularly useful today in enterprise-cloud settings. We finally generalize this solution, proposing a framework that optimizes a wider set of ingress interdomain traffic `SLO`*s*.

1.1 Understanding the Performance of Ingress Traffic Engineering Approaches

Two widely used methods of steering traffic from user networks to Service Providers are `anycast` and `unicast`. Their wide use by all of the worlds' largest Service Providers, and measurement studies [29, 30], suggests that these methods work well. However the literature paints a confusing and incomplete portrait of how well these methods help Service Providers satisfy `SLO`*s*.

A key limitation of `anycast` is that BGP, the protocol that chooses among paths to different sites, does not incorporate performance in its decision process, and so some chosen paths may be poor (*e.g.,* high latency). One can quantify this limitation using a metric called path inflation [20], which is the path latency users achieve minus the path latency users could achieve if they visited their lowest-latency site. The literature disagreed on how frequently this inflation occurred in practice – some studies say this inflation is quite bad [26, 27, 28, 37], and one study goes further

to say more sites makes `anycast` worse and that BGP should be modified to make it better [27].

`Unicast`, on the other hand, allows Service Providers to limit path inflation by roughly controlling which site users arrive at, but has two other key limitations: (a) control over which site users route to is coarse-grained and (b) this control can take time to enact due to caching in clients and resolvers. The first is well-understood: it would benefit Service Providers to direct users with `unicast` at finer granularity, but this direction is not always possible via DNS [31, 29, 32]. This direction is not always possible because mappings from hostnames to IP addresses are conducted at the granularity of a recursive resolver, and recursive resolvers can serve large groups of users. However, it was less well-understood how frequently mappings could be propagated to users (slow updates means less control which means more problems). One study quantified the limitation by looking at what fraction of sessions still used an old DNS record 5 minutes after changing it for a specific application [34] but did not quantify this effect with respect to *traffic* or for different applications/Service Providers.

One common thread in the above-mentioned analyses of `anycast` and `unicast` is that each one assessed a single application and thus a single type of deployment. However, these techniques are used for many deployments and can be instantiated in many ways. Hence, in clarifying our understanding of `anycast` and `unicast` performance, we strove to analyze their performance across deployments and networks.

To answer questions about `anycast`, we first analyzed two global services that use `anycast`: Microsoft's CDN and the root DNS. Microsoft's CDN serves web content for itself and its customers, while the root DNS mostly serves the root DNS zone. The root DNS zone serves a couple thousand top-level-domains (*e.g.,* COM and ORG).

Using global-scale measurements from both Microsoft and the root DNS, we computed the amount of path inflation in both deployments. We found that the root DNS had far more inflation than Microsoft. We found 10% of users travel an extra 2,000 km to the root DNS (at least 20 ms) while the same is true for less than 2% of Microsoft's users. At global scale, such differences are stark as they represent millions of users in tens of thousands of networks.

We then argue that far fewer Microsoft users incur significant inflation because Microsoft has a much greater incentive to fix inflation. We found that, since root DNS records are highly cacheable, users rarely actually experience latency due to the root DNS since most users incur latency from less than one query to the root DNS per day, whereas, when loading web content from Microsoft, users usually incur 10 or more round trips and so will experience a multiple of any excess latency. Microsoft fixes inflation by peering widely with other networks and working with operators from user ISPs to fix instances of poor routing. Hence, `anycast` works well (but not perfectly) for most users and is likely good enough considering web content's round-trip path latency goal of less than approximately 100 ms [1].

To study `unicast's` limitations, we analyzed traces from a residential network housing graduate students, postdocs, faculty, and their families at Columbia University. These traces allowed us to observe DNS caching behavior over many applications and contextualize this behavior with respect to traffic and flow volumes. By associating traffic with corresponding DNS queries, we computed how long users used stale DNS records to send traffic to Service Providers. We found that, for traffic destined to most large Service Providers, nearly half of traffic used a stale DNS record, and 10% of traffic used records that had been stale for at least 10 minutes. We also found that failover between multiple DNS addresses takes tens of seconds to minutes on popular browsers/applications, making it difficult for Service Providers to implement failover mechanisms using DNS. Hence we found that Service Providers are severely limited in their ability to change how users route to a `unicast` deployment at timescales on the order of minutes, which may be especially important during failure.

We then investigated whether specifying many addresses as answers to a single query could enhance resilience since clients could, in theory, switch to a backup address if one failed. We found, however, that popular operating systems and browsers often took minutes to switch to backup addresses which could severely affect user experience. Clients also *randomly* chose which address to use, hampering Service Provider control.

We conclude from this investigation that `anycast` is possibly better than widely thought and

is good enough for distributing web content. However, anycast inevitably leads to some inflation which can hurt performance for some users, and fixing this inflation may require constant effort from operators (for example, updating peering strategies or BGP announcements). Unicast, on the other hand, gives Service Providers more control over where users go, but may introduce reliability problems due to caching. These two methods of directing traffic are sufficient for many use cases, but in the next section we explore whether they can continue to provide the guarantees modern Service Providers need.

1.2 Improving Interdomain Routing By Exposing and Selecting Good Paths

This dissertation demonstrates that anycast and unicast work well enough for traditional applications with lower SLO*s* such as non-critical web/video content. Increasingly, however, the Internet is being used for more demanding applications and use cases. Emerging network use cases such as factories of robots evaluating machine learning models in the cloud, VR/AR, self-driving cars, and remote surgery have tighter throughput/latency requirements and demand higher reliability than any existing application (*e.g.,* VR/AR requires 10 ms [1]). 5G applications could drive massive amounts of data (> 1 Gbps per device). For businesses that use the cloud for online meetings and productivity software, poor network performance now means decreased efficiency. For such critical use cases, Service Providers need interdomain routing strategies that provide better path latency and reliability guarantees than anycast and unicast.

We provided these guarantees by designing a system, PAINTER, which relied on two observations. The first is that anycast and unicast only make available a small set of all possible paths from users to the Service Provider, and those paths might not be the best ones. Hence it is likely better to instead find ways of exposing more paths, specifically those paths that offer the best performance. The second observation is that DNS limits both the rate and traffic granularity at which Service Providers can change which path user traffic takes, so Service Providers would benefit from finding ways to sidestep DNS.

PAINTER uses these two observations to significantly lower steady-state latency to a Service

Provider by first making available low-latency paths to the Service Provider for users and second allowing the Service Provider to change which path users take at fine granularities (both time and traffic granularities). PAINTER does not simply make every path available, as user networks choose one path per IP prefix, and prefixes are expensive. PAINTER instead computes a good subset of paths to expose by using an efficient heuristic to approximately solve an integer optimization problem over a very large search space through which blind search is very slow due to the limitations of Internet dynamics.

To sidestep DNS and allow fine-grained redirection, we use Service Providers ' increased presence near the user as a control point from which to measure and select among paths. The increased presence consists of choke points through which traffic flows at which the Service Provider can enact increased control compared to our standard Internet setting, and may vary depending on the Service Provider. These points include networking-as-a-service devices such as SD-WAN, cloud-edge network stacks, mobile applications that give developers more control over the network stack, multipath transport protocols (MPTCP/MPQUIC), and integrated VPNs such as Apple Private Relay. We call the set of all these choke points "edge proxies". We designed PAINTER to specifically apply to the enterprise cloud setting where the control point is a software stack on commodity hardware in an enterprise's network, but it could be easily adapted to these other types of edge proxies.

We deployed PAINTER on the PEERING testbed [38] which uses the connectivity of a large Service Provider, Vultr cloud, to emulate an actual Service Provider. Vultr cloud connects to more than 6,000 networks at 25 sites. We found that PAINTER reduces steady-state path inflation by 75% (30 ms) compared to anycast for thousands of networks, using a third of the resources (IP prefixes) compared to other solutions. PAINTER also enables per-flow traffic redirection at RTT-timescales, representing orders of magnitude improvement compared to DNS-based redirection.

1.3 Towards Meeting General Ingress Routing Objectives

This dissertation demonstrates that we can move past the performance and reliability offerings of traditional unicast and anycast with better path options and traffic direction mechanisms.

However, this conversation has focused (until now) on improving *steady-state latency*.

Service Providers, however, have to increasingly meet diverse requirements for a myriad of applications. For example, enterprise services have tight reliability requirements [25, 39], and new applications such as virtual reality require $\leq$ 10 ms round trip latency [1] and $\leq$ 3 ms jitter [40]. Complicating matters, Service Providers must meet these requirements subject to changing conditions such as peering link/site failures [41, 42], DDoS attacks [43, 44, 45], flash crowds [46, 47, 48], new applications/business priorities (LLMs), and route changes [49, 50, 51, 52, 53, 42]. Critically, such changes can cause *overload* if the Service Provider cannot handle new traffic volumes induced by the change, and DNS/BGP are slow mechanisms with which to change how traffic flows over paths. This overload can lead to degraded service for users, hurting reliability [54, 55, 56, 39], and may require manual intervention.

Current solutions to this problem are either (a) too costly (*e.g.,* by excessively overprovisioning resources), (b) too reactive (*e.g.,* by redirecting traffic during failure, which is dangerous), or (c) too *specific*. Even PAINTER, while flexible, is difficult to extend to considering more general objectives other than steady-state latency. Service Providers are thus forced to make a choice to deploy expensive, reactive, or custom solutions for each unique problem they face and service they provide.

We present Service Providers with a flexible framework, SCULPTOR, which accepts as input cost, performance, and reliability objectives and outputs BGP advertisements and traffic allocations that help achieve those desired objectives. SCULPTOR is the first system that optimizes ingress interdomain routing objectives such as maximum link utilization, transit cost, and latency for *interdomain* traffic (existing systems optimize for intradomain traffic, *e.g.,* [4, 3, 57, 5]). SCULPTOR computes BGP advertisements proactively, only placing live traffic on them after convergence and so minimizes risk to Service Providers during deployment.

To solve each optimization problem, SCULPTOR efficiently searches over the large BGP advertisement search space ($> 2^{10,000}$ possibilities) by modeling how different strategies perform, without having to predict the vast majority of actual paths taken under different configurations,

since predicting interdomain paths is hard and measuring them is slow (§5.2.3). SCULPTOR then optimizes these (modeled) performance metrics using gradient descent, which is appropriate in our setting due to the high dimension of the problem and the parallelism that gradient descent admits (§5.2.4). This modeling enables SCULPTOR to assess > 20M configurations (10,000× more than other solutions including PAINTER [35, 25]) while only measuring tens in the Internet (§5.4.4). Like other work that uses gradient descent with success (*e.g.,* deep learning), we sacrifice the ability to provide a formal characterization of which objective functions are possible for an approach that lets us optimize for multiple criteria (§5.4)

We prototyped SCULPTOR on the PEERING testbed [38] (§5.3) and evaluate our framework on two specific objectives (computed separately): (a) optimizing latency under unseen traffic conditions, and (b) routing different traffic classes. We compare SCULPTOR's performance on both problems to that of an unreasonably expensive "optimal" solution (computing the actual optimal is infeasible).

For the first objective, we found that, among other benefits, SCULPTOR can handle flash crowds (*e.g.,* DDoS attacks) at more than 3× expected traffic volume, reducing the amount of overprovisioning that Service Providers need, thus reducing costs. For the second objective, SCULPTOR routes bulk low-priority traffic in ways that avoid congesting high-priority traffic, achieving near-optimal latency and reducing congestion by up to 2× (§5.4.3). SCULPTOR thus brings Service Providers closer to providing the same high-level programmable intent to the *interdomain* setting that Service Providers already have in the intradomain setting.

Chapter 2: Background and Related Work

2.1 Service Providers

2.1.1 Networked Service Basics

Most people understand the Internet as something that provides services in the form of applications. Most of the applications people use today are networked – that is, they involve communication between the user's application and some other application (which is almost always on another physical device, so that is how we will refer to this situation). This communication may be necessary due to information or resources only being available on the other system, or it may be the feature of the application (*e.g.,* messaging). We refer to these networked applications as clients.

As users would necessarily like to use a networked application at any time, the user should be able to communicate over the network with the other client at any time. Hence, oftentimes the second endpoint is an always-on client waiting to engage in the networked application. We refer to these always-on clients as servers, as they exist in service of networked applications. This client-server framework where a user's client communicates with one or more servers to enable a networked application is the focus of this thesis.

A network is a collection of interconnected routers and clients, and is the physical/logical organizational component from which the Internet is composed. A network is managed by a single organization, and different networks connect with each other to form the Internet. The function of the network is to enable communication among clients in the network, and between clients in its network and other networks. We distinguish mostly between three types of networks: user networks, transit providers, and Service Providers. User networks provide Internet service to people for a fee, transit providers provide connectivity to networks for a fee, and Service Providers connect to user networks and transit providers (sometimes with or without payment) to provide the

Internet with access to services. Another type of network that Service Providers connect to is a peer (as opposed to a provider). Peers connect to Service Providers at no cost to either party (*i.e.,* a settlement-free arrangement), but will only help deliver traffic from and carry traffic to customers of that peer. A transit provider, on the other hand, will help carry traffic to and deliver traffic from any network on the Internet.

2.1.2 Network Performance

One factor that can affect how much the user enjoys their quality of experience (QoE) using a networked application is the network's performance. Network performance can be measured in many ways, but we focus on three metrics: latency, throughput, and reliability. Latency is the total amount of time it takes information to propagate from one client to another, and generally lower is better. Throughput refers to the amount of information per second a client can send to another client, and generally higher is better. Reliability refers to the fraction of time a certain (latency, throughput) requirement is met by the network, where higher is better. While zero latency, infinite throughput, and 100% reliability would be ideal, users can generally obtain a good QoE with less-than-ideal network performance. We refer to the set of network performances that would enable a client to experience a good QoE as an application's `SLO`. This association of application with `SLO` is a simplification as, for example, network performance good enough to watch one Netflix video might not be good enough to watch another Netflix video.

In our context, latency manifests itself in two key ways: propagation delay and queueing delay. Propagation delay refers to the time packets spend on the communication medium(s) between the source and destination. Propagation delay is usually dominated by the time spent on fiber-optic links and is on the order of milliseconds to hundreds of milliseconds. Electromagnetic waves (*i.e.,* packets) can travel at 200 km/ms in fiber-optic cable, and so propagation delay is correlated with geographic path length between the source and destination. Queuing delay, on the other hand, refers to the amount of time packets spend in router/switch buffers. This delay can become significant relative to propagation delay if links are congested: that is, the rate at which packets

arrive at a buffer significantly exceeds the rate at which they leave it. With modern hardware, congestion usually occurs because routers cannot transmit packets into the fiber-optic medium faster than the rate they arrive in the buffer (as opposed to, for example, taking too long to switch a packet from an input to an output port). Low transmission rates can be due to medium contention in the case of wireless transmission (WiFi/cell links), or too-low-bandwidth links in the case of wired transmission. From these descriptions, propagation delay is therefore a fixed property of a path between a source and destination, whereas queueing delay is a dynamic property that depends on externalities.

2.1.3 Service Provider Deployment Structure

Due mostly to economics, servers running most popular applications are owned/managed by a relatively small set of large companies, which we refer to as Service Providers. These companies include Microsoft, Google, Tencent, Akamai, and more. To meet the aforementioned network performance goals, these Service Providers have built highly optimized global networks to meet `SLO`*s* for their global users. These global networks have similar characteristics/components, which we now describe.

Since clients are not always-on, clients almost always initiate communication with servers to enable their services. A single server handles communication with many clients which consumes various resources on the server: network bandwidth, CPU, and/or peripheral devices. Since Service Providers often have global user bases, many servers are needed to provide (in aggregate) enough resources to handle communication with all clients. To accommodate this reality, Service Providers group their servers in large *data centers* which are physical spaces designed to provide sufficient space, cooling, and power to thousands of servers.

Service Providers often have data centers in tens of locations around the world to provide redundancy and to provide Service Providers' global user bases with physically close servers (as physical closeness can yield low propagation delay). To enable some services, Service Providers connect data centers to each other via either a logical or physical Wide Area Network (WAN) —

i.e., the set of all data centers around the world forms the Service Provider's network. A significant amount of traffic flows between data centers: for example, data centers might store copies of information to enable fault-tolerant storage. Hence, intra-network performance is important for meeting application SLO*s*.

Service Providers have complete knowledge and a high degree of control over how their networks function: they can measure network state, change routes, develop custom hardware, and change processes at every layer (network, physical, transport, *etc.*). Since intra-network performance can be important for enabling applications, and since Service Providers have so much control in this domain, Service Providers have systems that ensure performance [3, 4, 5, 58], reduce costs [59, 60, 15, 61], and ensure resilience to failure [62, 63, 64, 65, 66, 67, 68, 69].

Service Providers need to connect to the rest of the Internet to receive requests from clients and do so at locations we call sites. Sites therefore serve, among other things, as entry points into an Service Provider's network. Sites are often near major cities since users concentrate around cities and since networks often come together near cities to physically connect to each other. Having more sites as close as possible to the most users generally helps meet SLO*s* (low propagation delay), so many Service Providers have hundreds of sites around the world. For reasons we discuss in Section 2.2.2, Service Providers connect to thousands of networks across their sites, and connect to many networks in more than one location.

2.2 Routing

Networked applications involve communication between two or more clients through one or more networks. A key challenge in networking is pathing/routing traffic from one client to another so as to meet overarching application SLO*s*. A network path/route through the Internet can be thought of as a path through a graph where nodes are routers and edges are links connecting those routers. Some edges connect routers in the same network, while other links connect routers from two different networks. A route between two destinations can involve an *interdomain route* which consists of the latter type of edge and each part of the route between interdomain edges consists

of an intradomain route. We now describe how intradomain routes and interdomain routes are computed.

2.2.1 Intradomain Routing

Intradomain routing tries to compute a set of forwarding rules to install in routers in a single network that will optimize performance objectives subject to a desired traffic matrix and physical constraints. The traffic matrix specifies the set of source nodes, sink nodes, and their associated traffic volumes. Source nodes refer to traffic entry points in the network which could be traffic from other networks or traffic from clients within the network. Sink nodes similarly could be connections to other networks or clients. Performance objectives may include minimal latency, minimal maximum link utilization, minimal traffic flow completion times, minimal cost, and more. Physical constraints may include not inundating links, having a maximum number of forwarding rules per node, or finding a solution within a certain time window. Network designers can also optimize for unseen conditions such as resilience under k-link failures or changes in the traffic matrix.

One challenge in the above formulation is that, for traffic destined to other networks, there may be many valid exit locations (the set of which is determined via interdomain routing) and so there may be many possible traffic matrices that satisfy traffic demands. Two simple strategies for choosing traffic egress points are hot-potato and cold-potato routing. Hot-potato routing chooses the "closest" exit point to the entry point, while cold-potato routing chooses the closest exit point to the final destination. The specific definition of closeness is defined by the operator. It is generally unclear which exit point is "best" as networks have limited visibility into other parts of the path outside of their own network, and this aspect of intradomain routing can almost be viewed as an interdomain routing problem.

2.2.2 Interdomain Routing and BGP

Interdomain routing finds a route through separate networks between clients. As the Internet is large and distributed, interdomain routes are computed in a scalable, distributed fashion. This process is standardized by the Border Gateway Protocol (BGP) [70].

BGP computes routes per IP prefix based on information received from other networks. Routers communicating via BGP can either be border routers or non-border routers. Border routers sit on the "border" of networks as they connect with other networks, and speak BGP to other border routers in other networks. Border routers then distribute information received from other networks to non-border routers in their own network. All routers in a network often speak BGP and perform route computation according to the BGP decision process.

BGP-speaking routers may receive many routes towards an IP prefix but only send one route per prefix to connected neighbors (and internal routers) according to a specified policy (*i.e.,* the BGP decision process). Given multiple paths towards a prefix, a BGP router will prefer routes with higher local preference (local preferences generally represent economic incentives and are specified manually by the operator). If two paths have the same local preference, the router will choose the path with a shorter AS-path (loosely tied to performance), and the router breaks further ties using other miscellaneous attributes.

BGP is scalable since only one path is chosen and exported per prefix per router, and flexible as local preferences are mutable/arbitrary, but for the same reasons BGP limits visibility into how routing in the Internet works. As a BGP router advertising a prefix, it is difficult to predict/control how that route will propagate through the Internet, and similarly, given many routes towards a prefix, it is difficult to obtain much information about each available path or about other, unchosen routes. It is difficult to control how a route will propagate because how a route propagates depends on decisions made in other networks to whom the BGP router advertises the route (and whoever those networks advertise routes to, and so on), and networks lack visibility into what other networks look like, and how other networks assign local preferences. Similarly, it is difficult to determine properties of received routes (*e.g.,* performance) as the only information passed along

with a route is the AS path. These limitations of BGP make achieving networking goals related to interdomain routes a key challenge for many networks today, and changing the protocol itself is hard as the protocol is so entrenched.

2.2.3 `Anycast`, BGP `Anycast`, and `Unicast`

IP `anycast` refers to when many different servers all host an identical service and are tied to the same IP address. BGP `anycast`, on the other hand, refers to when all border routers advertise an identical prefix/route. For the purposes of this dissertation we use the term `anycast` to refer to both: a network originating an `anycast` IP prefix will announce that prefix from all border routers, and that IP prefix contains IP `anycast` addresses. `Anycast` is very simple to implement and has the advantage of providing a type of failover, which we now describe. If a site/BGP session advertising an `anycast` prefix suddenly goes down then, at first, that route will become invalid to all downstream networks using that route, so some networks may experience a loss of reachability to the prefix. However, BGP soon recomputes a valid route for those networks towards that prefix, as the prefix is still being advertised over several other BGP sessions. Hence, `anycast` provides failover at a rate on the order of BGP reconvergence, which is usually tens of seconds to minutes [24].

`Unicast` refers to when an identical service can be reached at IP addresses in distinct IP prefixes. Each IP prefix is announced from one site to all networks at that site (and so could be viewed as BGP `anycast` from that particular site). Addresses in `unicast` prefixes could be IP `anycast` if multiple servers in a network share the same address. Hence, the terms "uni" and "any" refer (as they are commonly used) more to the sites at which prefixes are advertised, and less to whether an IP address has been assigned to one/many server(s) or has been advertised to one/many network(s).

The reason for this distinction is twofold: first, most Service Providers today use `unicast` and/or `anycast` and so they deserve specific names. The second is that oftentimes `unicast` and `anycast` are discussed (*i.e.,* named) in the context of the research challenges they pose or

solve (§2.4). Many of these research challenges are related to the (less controllable) interdomain part of the path, as the intradomain part of the path is under full control of the network and therefore presents fewer challenges. Hence distinctions over, for example, whether IP addresses in `unicast` prefixes are assigned to multiple servers is usually irrelevant in a research context.

`Unicast` gives networks more control over how clients route to a destination than `anycast`, but does not give networks full control. As with any other BGP advertisement, a network has limited visibility into how `unicast` routes will propagate through the Internet. The increased control refers to the fact that networks can use `unicast` prefixes to direct clients to particular *sites*, which is much more difficult to do with `anycast` prefixes.

2.2.4 DNS

The Domain Name Service (DNS) is both a protocol and a global, distributed system for mapping human-readable names to IP addresses. (DNS also serves other uses but we do not discuss those uses.) The protocol is a simple question-answer protocol where a client asks for the IP address of a human-readable name, and the server responds to this request. We discuss a common model for how this process works as it relates to Service Providers.

Clients query the DNS for hostnames either known to them or specified by an application. These requests are addressed to a client's local DNS resolver which is usually specified by the Dynamic Host Configuration Protocol (DHCP), but can also be manually specified by the client. A local DNS resolver either serves only users in an ISP (private) or in many ISPs (public). This local DNS resolver then queries other DNS resolvers to fully resolve the IP address.

The local DNS resolver first consults its cache to see if the record to query for has already been queried before. Assuming the cache is completely empty, the local DNS resolver queries the root DNS for the Top Level Domain (TLD) server corresponding to the hostname the client asked about. There are a few thousand TLDs: examples include `COM`, `ORG`, and `GOV`. The root DNS is a service run on 13 distinct `anycast` prefixes, identified by the letters, for redundancy `a-m`. Upon querying for a TLD record, the recursive resolver can choose any letter to query and so can query

the root DNS according to performance, reliability, or some other policy. There are thousands of root DNS servers around the world for redundancy spread across these 13 deployments, with each letter having between 6 and 353 sites.

After successful retrieval of the TLD record, the local DNS resolver then queries the TLD server for the authoritative server corresponding to the domain (*e.g.,* `google`, `columbia`, `yahoo`). Finally, the local DNS resolver queries the authoritative server for the record of the domain the client is trying to access, and forwards the response to the client. In this process, Service Providers have control over the domain being queried and the response of this authoritative server.

Each of these records is associated with a Time to Live (TTL), which is a field set in the answer message specifying how low that answer can be cached before re-retrieval. Setting a high TTL (longer caching) helps mitigate load on the DNS infrastructure and, since local DNS resolvers are often close to their users, lowers latency for clients who may not have to incur the latency of a DNS query. However, high TTLs also limit the rate at which name to address mappings can be distributed to users as DNS servers cannot "push" updates to resolver caches. DNS caches may also exist in client browsers, operating systems, and in-home routers.

TTLs for root DNS records are between one and two days as these records rarely change, whereas TTLs are often between 1 and 60 seconds for domains associated with popular services.

From this description of DNS, it follows that Service Providers can map user queries to IP addresses at the network granularity of a local DNS resolver and at the temporal granularity of a TTL. Under this model Service Providers cannot, for example, map many clients who share the same local DNS resolver to different IP addresses when those clients query for the same hostname. Extensions to DNS could support redirection at the granularity of the client's IP prefix [32], but such extensions are not widely supported [71].

2.3 Measurements and Platforms

As the Internet is a large, distributed, information-hiding system, it is difficult to answer questions about the Internet. For example, it is difficult to determine what certain properties of the In-

ternet are, the relative importance of factors that play into an issue, and how well systems/solutions to problems in the Internet work. We cannot use simulations to get answers to these questions as it is unclear if our models of the Internet are sufficiently representative.

Sometimes we can *fully* answer questions about networked systems— perhaps we can exhaust those scenarios using our limited resources or we can analytically answer questions. However, in cases where we cannot fully answer questions, an alternative is to measure the system across a wide range of scenarios and argue that those scenarios are sufficiently representative of all/most behavior. Researchers and practitioners have public and privileged options for conducting these measurements which vary depending on the context.

2.3.1 Public Measurement Platforms/Testbeds

Opportunities for general researchers/practitioners to measure networked systems at scale often come in the form of measurement platforms/testbeds. We use two of these in this thesis: RIPE Atlas and PEERING.

RIPE Atlas is a distributed set of probing devices in networks around the world capable of issuing network measurements such as latency and path measurements [72]. Users can conduct measurements by spending periodically allotted credits, subject to measurement rate limits. There are approximately 10,000 probes spread across 3,500 networks and hundreds of countries giving us a decent view of network performance/properties and a much better view than just relying on a single vantage point. However, coverage is heavily biased towards European countries.

The PEERING testbed provides us with a way of conducting BGP experiments on the Internet[38]. PEERING has BGP sessions with networks at tens of sites around the world and allows users to advertise IP prefixes to those networks which then propagate through the Internet. Recently, PEERING was additionally deployed at 32 Vultr cloud sites [73]. Vultr allows cloud tenants to advertise and receive routes from/to BGP speaking routers at Vultr sites with a high degree of flexibility. This flexibility allows us to, for example, advertise prefixes only to specific networks at specific sites. Vultr connects with more than 3,000 networks totalling more than 10,000 entry

points as some networks connect at multiple sites, giving PEERING the same geographic footprint and connectivity as a major Service Provider.

2.3.2 Privileged Measurement Capabilities

Some parties have opportunities to measure network properties only available to those parties. These parties often do not share these viewpoints due to either logistical constraints, to maintain a competitive advantage, or to adhere to financial constraints. One common manifestation of this case is when a Service Provider uses their representative, large-scale systems to answer questions. For example, Odin allows Microsoft to issue arbitrary network measurements from billions of users in more than one hundred thousand networks [74], but does not share these measurements with the public. RIPE Atlas' coverage pales in comparison, but is available to everyone.

2.3.3 Basic Measurement Tools

Even with full control over a network, it may not always be clear how to measure network properties with precision and accuracy. Hence, opportunities for successfully measuring properties of networked systems grow as we develop new measurement methods. We regularly use two basic measurement tools to build more complex measurement systems: these tools are `ping` and `traceroute`.

`Ping` measures round-trip delay between two networked devices by sending a specific packet from the source to the destination, waiting for a specific reply, and measuring the time difference between the sending and receiving.

`Traceroute` tries to measure the forward path from the source to the destination, where a path is a sequence of IP addresses corresponding to routers along the path. `traceroute` works by sequentially incrementing the TTL field in the IP header, which is a field that specifies the maximum number of routers a packet can pass through before being discarded. By incrementing the TTL field from 1 to 255, the source sending these packets receives responses from routers along the path. Those routers then notify the source that their packet is being discarded. By noting

the router's address in these received "error" packets, the source thus enumerates a series of IP addresses comprising the path from source to destination.

Both `ping` and `traceroute` have limitations. A limitation common to both tools is that they cannot directly measure properties of the reverse path (from destination/router to source). Hence, for example, the latency along the path from source to destination is in general not half of the time returned by `ping`, as the reverse path may have a different latency than the forward path.

2.4 Related Work

Root DNS `Anycast`. Many prior studies look at latency and inflation performance in the root DNS [26, 75, 76, 77, 27]. This thesis hopes to build on these studies, conducting analysis for nearly every root letter and calculating inflation at global scale. Larger scale measurements would offer broad coverage, enable comparisons among root letter deployments, and will allow us to assess inflation in the root DNS *system* as a whole. We also hope to compute latency inflation differently than in prior work, which we believe offers a useful, orthogonal picture of inflation. We also hope to compute inflation using the same methodology for both Microsoft's and root DNS, which would allow us to compare inflation directly between Microsoft's CDN and root DNS (not possible with prior studies). Finally, this thesis hopes to place latency and inflation in the context of user experience and in different systems with different `SLO`*s*, while prior work on the root DNS does not.

Other prior work looks at `anycast's` ability to defend against DDoS attacks that actually took place [26, 43]; this thesis would like to compare how interdomain routing strategies other than `anycast` hold up against DDoS attacks, but does not have real DDoS attack against which to evaluate the strategies. Hence, we offer a complementary view of how interdomain routing strategies could affect resilience to such events. Other prior work discussed how ad-hoc `anycast` deployments can lead to poor performance and load balancing and is an early study of inflation in the root DNS [75]. We hope to corroborate these conclusions and use them in a larger conversation about `anycast` in the context of applications. We would also like to follow up an old study that

found that anycast site affinity is high (*i.e.,* if clients in the same network tend to reach the same site) [78].

CDN Anycast. Some CDNs use IP anycast [79, 80, 29, 81, 82]. Some prior work looks at inflation in CDNs [29], finding it to be low. In this thesis we hope to present a much larger study of latency and inflation (in terms of size of the deployments considered, and in terms of the numbers of users/networks considered), thereby updating the numerical results and confirming/denying whether inflation is high. We also hope to place performance metrics in the context of user experience, compare performance to other systems that use anycast, and provide some evidence of if/how CDNs can keep inflation low. Other prior work looked at how prefix announcement configurations can impact the performance of an anycast CDN [28]. More recent work has investigated how to diagnose and improve anycast performance through measurements in production systems [83, 74, 84]. Concurrent work examined addressing challenges faced by CDNs, proposing a scheme to decouple addressing from services that is compatible with anycast [85]. This thesis looks at (a) analyzing anycast as it is and (b) comparing anycast to other, non-anycast routing strategies.

Recursive Resolvers, The Benefits of Caching, and Web Performance. One significant part of the DNS ecosystem, and thus a significant part of what affects performance characterizations of the root DNS, are recursive resolvers, their caching behaviors, and how those behaviors interact with web performance. Prior work has looked at statistics and latency implications of local resolvers [86, 87]. We hope to characterize similar statistics, but from a different perspective and with global data to assess not only how they interact with web performance, but also how they interact with anycast as a means of routing DNS requests.

Some previous work looked at certain pathological behaviors of popular recursives and the implications these behaviors have on root DNS load times [88, 89, 90, 91]. For example, popular software can trigger unnecessary root DNS queries. We hope to assess whether these pathological behaviors still affect the root DNS using global scale data and in particular how this pathological behavior might influence our assessment of anycast in the root DNS system.

Another point of interest for this thesis is to contextualize anycast's performance in an application. Doing so will allow us to assess to what degree anycast is a viable routing strategy for Service Providers who have different application requirements. Many studies characterize web performance and consider DNS's role in a page load [92, 93, 94], although none consider how root DNS specifically contributes to page load time and how this relates to user experience. Recent work considers placing DNS in the context of other applications but does not look at root DNS latency in particular [95]. Hence, it will be a goal of this thesis to characterize root DNS's (and in particular, anycast's) role in application performance.

Egress Traffic Engineering. Recall that egress traffic engineering refers to the practice of sending traffic from a single Service Provider site to a single client prefix along many paths to the client. Service Provider sites learn many paths to each prefix since Service Providers connect to many networks at each site. There are large scale systems that steer egress traffic, selecting one of multiple paths to client prefixes to either improve performance [9, 8, 10] or optimize peering costs [61, 15]. This thesis addresses problems for ingress traffic, and so is orthogonal to the effects of these systems.

Ingress Traffic Engineering. PECAN issues multiple advertisements to a single ISP to expose routes and uses DNS to steer traffic [96]. However, it is unclear to what extent PECAN is agile enough for modern Service Providers and whether PECAN's methodology scales to that of a Service Provider with millions of users and thousands of peers/providers. Another study used a combination of MPTCP and SD-WAN to steer traffic across tunnels to the cloud [97]. This thesis answers questions that study did not address such as how should those tunnels be set up, in what use cases does it make sense to invest the resources in creating that technology, and how can Service Providers create scalable solutions that use similar ideas. Other work steers traffic by advertising prefixes to different ISPs [98], but does not scale to modern Service Providers with thousands of peers/providers.

Other work aims to overcome the limitations of BGP to perform ingress traffic engineering [29, 34, 41, 35, 24, 36]. This thesis hopes to both contextualize some of these results, and improve on

these methods.

Tango [6] exhaustively exposes paths between endpoints and so may provide resilience to dynamic traffic conditions if the right paths were exposed. However, exposing all the paths does not scale to our setting since there are too many paths to measure.

Other work and companies create overlay networks and balance load through paths in these overlay networks to satisfy latency requirements [99, 100, 101, 102, 48]. However, those companies act as Service Providers of their own and so they suffer from the key problem this thesis addresses.

Prior work built a BGP playbook to mitigate DDoS attacks [103], but it is unclear if those strategies would scale to large service providers (they tested on a few sites and a few providers). This thesis hopes to develop scalable solutions to handle dynamic interdomain traffic conditions.

Microsoft withdraws prefixes to mitigate congestion on links [42]. However, as they demonstrate in their paper, such actions are unpredictable and can lead to further congestion. We hope to find alternative solutions that plan for congestion and other dynamic traffic conditions.

Client-Side Reliability and Last-Mile Technology. Systems exist for measuring client performance at scale [74, 104]. Xlink uses MPQUIC over 5G, WiFi, and LTE to offer improved bandwidth for video streaming services on mobile devices [105]. DChannel uses both of 5G's high bandwidth and low latency channels to optimize web performance [106]. TGaming [107] uses feedback from a 5G network telemetry system to improve performance. These systems address a similar problem to the one this thesis addresses, but take a different approach by modifying the technology near or in the client. This thesis characterizes and addresses performance problems from the perspective of the Service Provider and proposes systems/solutions that can work alongside such client-side technology.

Intradomain Failure Planning. Large Service Providers have shown significant interest in reducing the frequency/impact of failures in their global networks [67, 108, 62, 68]. This thesis proposes solutions that work alongside such systems, as they handle failure in complementary ways.

Other prior work tried to plan intradomain routes to minimize the impact of k-component failures [63, 64, 65, 66, 5]. We hope to address similar problems, but in the interdomain setting.

Chapter 3: Understanding the Performance of Ingress Traffic Engineering Approaches

IP anycast is an approach to routing in which geographically diverse servers known as anycast sites all use the same IP address. It is used by a number of operational Domain Name System (DNS) [109, 82, 110, 111, 112] and Content Delivery Network (CDN) [79, 80, 29, 81, 82] deployments today, in part because of its ability to improve latency to clients and decrease load on each anycast server [113, 114, 115]. Unicast is another common approach to routing where users are routed to a specific site via DNS. The limitations of both of these approaches to ingress routing are not well understood.

Studies have argued that anycast often provides sub-optimal performance compared to the lowest latency one could achieve given deployed sites [26, 27, 28]. Notably, the SIGCOMM 2018 paper "Internet Anycast: Performance, Problems, & Potential" has drawn attention to the fact that anycast can inflate latency by hundreds of milliseconds [27], leaving readers of the paper with a poor impression of anycast. Conversely, other work has shown inflation is quite low in Microsoft's anycast CDN [29] and Google Public DNS [30], but used different coverage, metrics, and methodology, so it is difficult to directly compare results. Perhaps because of the very different takeaways of these studies, we have found that some experts in the community have negative opinions of anycast. It seems surprising that anycast continues to see more adoption and growth in production systems – why continue to use anycast if it causes inflation?

To understand the impact of anycast inefficiency and its wide use in spite of inflation, we step back and evaluate anycast as a component of actual applications/services. User-affecting performance depends on the anycast deployment, how anycast is used within the service, and how users interact with the service. To see these effects, we consider anycast's role within two

real-world systems: the root DNS and Microsoft's `anycast` CDN serving web content. These applications have distinct goals, they are key components of the Internet, and they are two of the dominant, most studied `anycast` use cases.

We analyze root DNS [109] packet traces which are available via DITL [116] and which are featured in existing `anycast` studies [117, 43, 77, 27, 28], with increased coverage compared to prior work. The 13 root letters operate independently with diverse deployment strategies, enabling the study of different `anycast` deployments providing the same service. We analyze two days of unsampled packet captures from nearly all root DNS letters, consisting of tens of billions of queries from millions of recursive resolvers querying on behalf of all users worldwide, giving us broad coverage.

We also examine Microsoft's CDN using the same methodology we use for the root DNS so we can directly compare results. Microsoft's CDN configures subsets of sites into multiple `anycast` "rings" of different sizes, providing deployment diversity, but all operated by one organization. We analyze global measurements from over a billion Microsoft users in hundreds of countries/regions, giving us a complete view of CDN performance.

With these measurements, we present the largest study of `anycast` latency and inflation to date. We first validate and extend prior work on inflation in `anycast` deployments [27]. Whereas that work focused primarily on a single root letter, we analyze almost the whole root DNS. By joining root DNS captures with global-scale traces of user behavior, we find that more users than previously thought experience *some* inflation (on average, more than 95% of requests), and as many as 40% of users experience more than 100 ms of inflation to some root letters (§3.2). However, average inflation per query to the roots is lower than previously thought, since each recursive resolver can preferentially query its best performing root letter – on average, only 10% of users experience more than 100 ms of inflation.

Do recursive resolvers have to implement preferential querying strategies for their users so that inflation does not hurt user performance? Our answer is a resounding "no" – using new methodology that amortizes DNS queries over users who benefit from cached query results, we

find differences in latency and inflation among root letters are *hardly perceived* by users – most users interact with the root DNS once per day (§3.3). Delay is minimal due to caching of root DNS records with long TTLs at recursive resolvers.

The inflated `anycast` routes to root DNS could be a result of latency not mattering, causing root operators to not optimize for it, or inflation could be inherent in `anycast` routing as suggested in prior work. To determine which is the case, we use measurements from Microsoft's CDN and find that, were latency to Microsoft's CDN to be *hypothetically* inflated as to individual root letters, it would result in *hundreds of milliseconds* of additional latency per page load. This increased latency would negatively affect the user's overall experience, especially when compared to root DNS. The key difference is that users incur several RTTs to Microsoft's CDN when fetching web content, whereas users rarely wait for a query to the root DNS because of DNS caching (§3.4.1).

With this context, we then measure *actual* inflation in Microsoft's CDN and find that inflation is kept comparatively small (§3.4.3), especially compared to individual root letters. To explain why inflation is so different in these deployments, we contrast AS-level connectivity and inflation between the users, Microsoft's CDN, and roots. We find that Microsoft is able to drive latency down and control inflation as deployment sizes increase through extensive peering and engineering investment (§3.6.1), even though inefficiency increases with these larger deployments (§3.6.2). Through discussions with operators of root DNS and CDNs, we find recent root DNS expansion has (surprisingly) been driven by a desire to reduce latency and mitigate DDoS attacks, while CDN expansion is driven by market forces (§3.6.3).

The comparison between performance in these two deployments allows us to put results from prior work in perspective [27, 77, 29, 117]. Even though root inflation is large, users rarely experience it, making its impact on the average query quite small. In contrast, users frequently interact with the CDN, and inflation there is small. These inflation results make sense, given the economic incentives of the organizations running Microsoft's CDN and the root DNS. While we expect these results to hold for other latency-sensitive services using `anycast`, as they have similar economic

incentives, a key takeaway from our work is that anycast must be analyzed in the context of the service in which it is used (§3.6.3), and so we cannot make definitive statements about generalizability. Hence, we do not refute past claims that anycast can inflate latencies, but we expand on these studies to show that, where it counts, anycast performance can be quite good.

We then characterize unicast, focusing on how quickly Service Providers can change exactly where clients go by updating DNS records. Towards this goal, we first measure the extent of DNS TTLS violations on a residential network due to caching within clients. Client applications, browsers, and operating systems may inadvertently cache these DNS records. Specifically, we look at how much traffic to popular cloud providers uses a DNS record whose TTL has expired that at least 10% and up to 80% of traffic still uses invalid DNS record five minutes after TTL expiration. This result suggests that it may be difficult to react quickly to failure or changing network conditions using unicast.

One possible way of guarding against failure, even with DNS TTLS violations, is to specify multiple DNS answers in a response to a query for a single record. Clients could then switch to other addresses upon failure, as is suggested in the Happy Eyeballs RFC [118]. However, we find that some popular operating systems, browsers, and applications take tens of seconds to switch to these backup addresses, at which point a user would likely give up trying to connect. Moreover, clients randomly choose which address to connect to, limiting Service Provider precision.

Hence we find that, while unicast admits better control than anycast, it may not provide quite as much control for Service Providers as was previously thought. This lack of control may be okay for some applications (*e.g.,* those with non-stringent reliability SLO*s*), but may not suffice for all applications.

3.1 Methodology and Datasets

We use a combination of DNS packet captures and global CDN measurements to measure latency and inflation. Root DNS data is readily available [116], while CDN data is proprietary. We supplement these datasets with measurements from RIPE Atlas [72]. We summarize our many

data sets' characteristics, strengths, and weaknesses in Section 3.1.3.

3.1.1 Root DNS

The first of the two systems we discuss, the root DNS, is a critical part of the global DNS infrastructure. DNS is a fundamental lookup service for the Internet, typically mapping hostnames to IP addresses [119, 120]. To resolve a name to its result, a user sends DNS requests to a recursive resolver (recursive). The recursive queries authoritative DNS servers as it walks the DNS tree from root, to top-level domain (TLD), and down the tree. Recursives cache results to answer future requests according to TTLs of records. The root DNS server is provided by 13 letters [109], each with a different `anycast` deployment with 6 to 254 `anycast` sites (as of July 2021), run by 12 organizations. A root DNS site can be local or global – local sites serve small geographic areas or certain ASes (controlled by restricting the propagation of the `anycast` BGP announcement from the site), while global sites are globally reachable.

We use three datasets: for end-users, we use long-term packet captures from THE INFORMATION SCIENCES INSTITUTE (ISI) AT USC, and DNS and browser measurements from daily use of two of the authors. For DNS servers, we use 48-hour packet captures at most root servers from Day in the Life of the Internet (DITL) [116].

Packet captures from ISI provide a local view of root DNS queries. The recursive resolver runs BIND v9.11.14. The captures, from 2014 to the present, reflect all traffic (incoming and outgoing) traversing port 53 of the recursive resolver. We use traces from 2018 (about 100 million queries), as they overlap temporally with our other datasets. This recursive resolver received queries from hundreds of users on laptops, and a number of desktop and rack-mounted computers of a network research group, so the results may deviate from a typical population. We found no measurement experiments or other obvious anomalies in the period we use.

We use the 2018 DITL captures, archived by DNS-OARC [116], to obtain a global view of root DNS use. DITL occurs annually, with each event including data from most root servers. The 2018 DITL took place 2018/04/10-12 and included 12 root letters (all except G root). Traces from

I root are fully anonymized, so we did not use them. Traces from B root are partially anonymized, but only at the /24 level. Our analysis does not rely on addresses more specific than /24, so we use all data from B root and all other roots except G and I. Although the 2018 DITL is older than the most recently available, it is significantly more complete than recent DITLs; in Section 3.8.1 we conduct analysis on the 2020 DITL and find none of our main conclusions change.

Since we aim to understand in part how root DNS latency affects users, we filter queries in DITL that do not affect user latency and queries generated by recursives about which we have no user data. We describe this pre-processing of DITL and subsequent joining of root query volumes with Microsoft's CDN user population counts.

Of the 51.9 billion daily queries to all roots, we discard 31 billion queries to non-existing domain names and 2 billion PTR queries. About 28% of non-existing domain name queries are NXDomain hijacking detection from Chromium-based browsers [91, 90, 121], and so involve machine startup and not browsing latency. Prior work suggests the remainder are generated by other malfunctioning, automated software [122]. Similarly, while PTR queries have some uses (traceroutes and confirming hostnames during authentication), they are not part of typical user web latency. In Section 3.3.3, we find that including invalid TLD queries significantly changes the conclusions we can draw about how users interact with the root DNS, and we provide more justification for this step. We next remove queries from prefixes in private IP space [123] (7% of all queries). Finally, we analyze only IPv4 data and exclude IPv6 traffic (12% of queries) because we lack v6 user data.

Sources of DNS queries in DITL are typically recursive resolvers, so the captures alone provide no information about how many DNS queries each user makes. To estimate per-user latency, we augment these traces with the approximate number of Microsoft users of each recursive, gathered in 2019 (the oldest user data we have). This user data is from Microsoft DNS data, which counts unique IP addresses as "users". This definition undercounts multiple human users that use a single IP address with Network Address Translation. Microsoft maps recursives to user IP addresses with an existing technique that instruments users to request DNS records for domains Microsoft

controls when users fetch content [124, 74].

We join the DITL captures and Microsoft user counts by the recursive resolver /24, aggregating DITL query volumes and Microsoft user IP counts, each grouped by /24 prefix to increase the amount of recursives for which we have user data. This aggregation is justified since many organizations use colocated servers within the same /24 as recursives [112, 125]. Prior work has also found that up to 80% of /24's are collocated [126]. We provide additional justification for this preprocessing step in Section 3.8.1, by showing all addresses in a /24 in DITL are almost always routed to the same `anycast` site. For simplicity, we henceforth refer to these /24's as recursives, even though each /24 may contain several recursives. We call this joined dataset of query volumes and user counts by recursive DITL∩CDN.

In an effort to make our results more reproducible, and as a point of comparison, we also use public Internet population user count data from APNIC to amortize root DNS queries [127] (*i.e.,* instead of using proprietary Microsoft data). APNIC obtains these AS user population estimates by first gathering lists of IP addresses from Google's Ad delivery network, separated by country. APNIC converts this distribution of IP addresses to a distribution of ASNs, normalized by country Internet-user populations. We use the TeamCymru IP to ASN mapping to map IP addresses seen in the DITL captures to their respective ASes [128] and accumulate queries by ASN. We were able to map 99.4% of DITL IP addresses to an ASN, representing 98.6% of DITL query volume. The assumption that recursives are in the same AS as the users they serve is obviously incorrect for public DNS services, but we do not make an effort to correct for these cases. Overall, we believe Microsoft user counts are more accurate, but APNIC data is more accessible to other researchers and so provides a useful comparison.

3.1.2 Microsoft's CDN

We also analyze Microsoft's large `anycast` CDN that serves web content to over a billion users from more than 100 sites. Traffic destined for Microsoft's CDN enters its network at a point of presence (site) and is routed to one of the `anycast` sites serving the content (front-ends). Mi-

crosoft organizes its deployment into groups of sites, called rings, that conform to varying degrees of regulatory restrictions (*e.g.,* ISO 9001, HIPAA), each with its own `anycast` address. The rings have the property that a site in a smaller ring is also in all larger rings. Other CDNs have to work with similar regulatory restrictions [129]. Hence, traffic from a user prefix destined for Microsoft's CDN may end up at different front-ends (depending on which ring the application uses), but often will ingress into the network at the same site. Users are routed to rings via `anycast` and fetch web content from a front-end via its `anycast` address. Users are always routed to the largest *allowed* ring given the application's regulatory restrictions (performance differences among rings are not taken into account).

Microsoft's `anycast` rings provide different size `anycast` deployments for study. In Figure 3.2 we show Microsoft's front-ends and user concentrations. Rings are named according to the number of front-ends they contain, and front-ends are labeled according to the smallest ring to which they belong (or else all front-ends would be labelled as R110). We do not show some front-ends too close to each other to improve readability. Circles are average user locations, where the radius of the circle is proportional to the population of users in that region. Figure 3.2 suggests that front-end locations tend to be near large populations, providing at least one low latency option to most users.

In Figure 3.1 we show front-ends in R110, and associated latency users experience to R110 in each region. Transparent circles represent user populations and their radii are proportional to the user population. Population circles are colored according to average median latency users in the metro experience to R110 – red indicates higher latency while green indicates lower latency. Latency generally gets lower the closer users are to a front-end. The CDN has focused on deploying front-ends near large user populations, which has driven latencies quite low for nearly all users.

User locations are aggregated by *region*, a geographic area used internally by Microsoft to break the world into *regions* that generate similar amounts of traffic and so contain similar numbers of users. A region often corresponds to a large metropolitan area. We refer to users at the ⟨`region, AS`⟩ granularity, because users in the same ⟨`region, AS`⟩ location are often routed

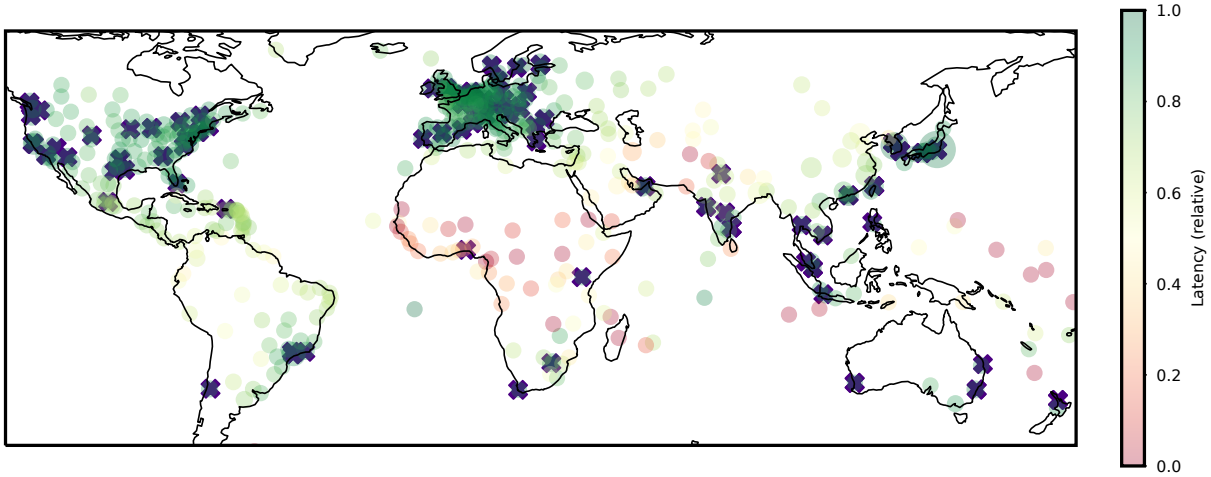


Figure 3.1: A visualization of front-ends in R110 (purple Xs), and user populations (transparent circles). User populations are colored according to the relative latency they experience and have size proportional to user population. Red corresponds to high latency, and green corresponds to low latency. Latency generally gets lower the closer users are to a front-end, and front-ends are concentrated around large user populations.

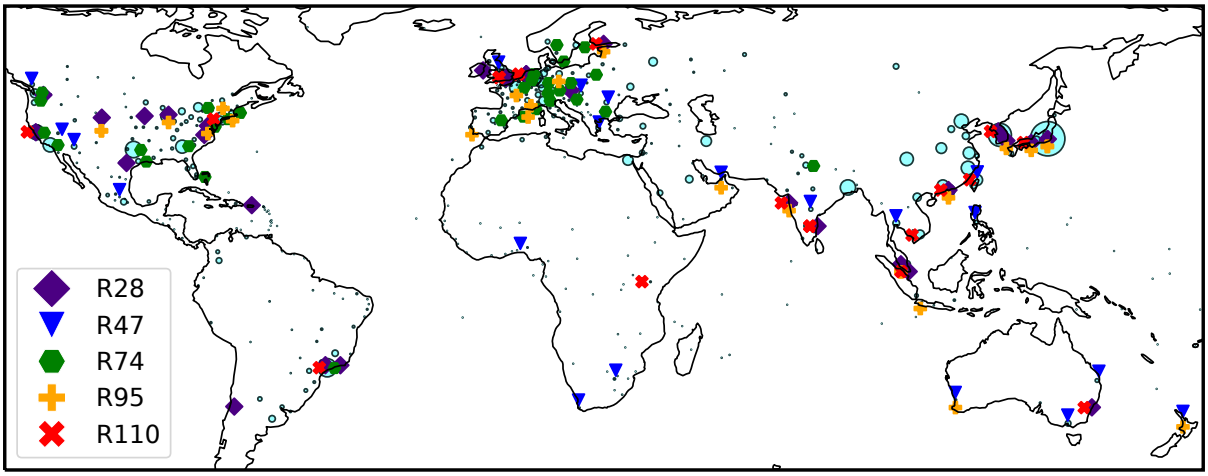


Figure 3.2: Microsoft's CDN rings and user populations. Sites in smaller rings are also in larger rings, and the legend indicates the number of sites in that ring. We do not show some front-ends too close to each other to improve readability. User populations are shown as circles, with the radius of the circle proportional to the number of users in that region, demonstrating that Microsoft has deployed front-ends in areas of user concentration.

to the same front-ends and so (generally) experience similar latency. There are 508 regions in total: 135 in Europe, 62 in Africa, 102 in Asia, 2 in Antarctica, 137 in North America, 41 in South America, and 29 in Oceania.

To study performance in Microsoft's CDN, we use two major data sources: server-side logs and client-side measurements. Server-side logs at front-ends collect information about user TCP connections, including the user IP address and TCP handshake RTT. Using these RTTs as latency measurements, we compute median latencies from users in a ⟨`region, AS`⟩ location to each front-end that serves them. Microsoft determines the location and AS of users using internal databases.

Client-side measurements come from a measurement system operated by Microsoft [74]. Latency measurements are the time it takes for Microsoft users to fetch a small image via HTTP. DNS resolution and TCP connection time are factored out. The measurement system instructs clients using CDN services to issue measurements to multiple rings, which enables us to remove biases in latency patterns due to services hosted on different rings having different client footprints (*e.g.,* enterprise versus residential traffic). Microsoft collects latencies of users populations, noting the location and AS of the user. Since these measurements come directly from end-users, we do not know which front-end the user hit. For both client-side measurements and server-side logs, we collect statistics for over a billion users across 15,000 ⟨`region, AS`⟩ locations.

We also use RIPE Atlas to ping `anycast` rings, because we cannot share absolute latency numbers. We calibrate these results versus our (private) data measuring latency for CDN users. In total, we collect 7,000 ping measurements to rings from 1,000 RIPE Atlas probes in more than 500 ASes to augment CDN latency measurements. (Probes were selected randomly, and measured three times to each ring.)

3.1.3 Summary of Data

We use a myriad of datasets in the work, which is a result of our presenting answers to the questions we pose in several different ways (each with strengths and weaknesses). This approach

Table 3.1: Summary of datasets used to analyze `anycast`.

Dataset	# of Measurements	Duration	Year	# of ASes	Technology/Format
Sampled CDN Server-Side Logs (§3.5)	11.0×10^9	1 week	2019	59 000	Windows TCP/IP, HTTPService (TCP RTT)
Sampled CDN Client-Side Measurements (§3.4.3)	50.0×10^7	1 week	2019	10 600	Odin [74] (HTTP GET)
CDN User Counts (§3.3.3)	—	1 month	2019	39 000	Custom URL DNS Requests
APNIC User Counts (§3.3.3)	—	updated daily	2019	23 000	Google Ad Delivery Network
DITL Packet Traces (§3.1.1)	51.9×10^9	2 days	2018	50 300	Packet Traces
DITL ∩ CDN (§3.2, §3.3.3, §3.6)	18.6×10^9	—	2018–2019	35 500	Root DNS query and user counts
RIPE Atlas (§3.4.3, §3.6.1)	10.0×10^3	1 hour	Various	3 300	ping, traceroute
USC/ISI (§3.3.3)	10.0×10^7	1 year	2018	1	Packet Traces
Local DNS / Activity Measurements (§3.3.3)	68.0×10^4	1 month	2020	2	Packet Traces, Chrome Webtime Tracker

Table 3.2: Strengths/weaknesses of datasets used to analyze `anycast`.

Dataset	Strengths	Weaknesses
Sampled CDN Server-Side Logs (§3.5)	Has client to front-end mappings, global coverage	Cannot hold user population fixed across rings
Sampled CDN Client-Side Measurements (§3.4.3)	Can hold user population fixed across rings, global coverage	Do not know which front-end the client reached, smaller scale
CDN User Counts (§3.3.3)	Precise estimates of user counts, global coverage	Under estimates user counts
APNIC User Counts (§3.3.3)	Global coverage, publicly accessible	Not validated, coarse granularity
DITL Packet Traces (§3.1.1)	Global coverage	Noisy, only above the recursive resolver
DITL ∩ CDN (§3.2, §3.3.3, §3.6)	Global coverage, attributes queries to users	Excludes v6
RIPE Atlas (§3.4.3, §3.6.1)	Historic data, reproducibility	Limited coverage
USC/ISI (§3.3.3)	Precise, below the recursive,	Limited coverage, no information about users
Local DNS / Activity Measurements (§3.3.3)	Precise, at the end user	Limited coverage, small scale

allows us to overcome the limitations of individual datasets, by combining multiple views with different tradeoffs. To aid in comprehensibility, we summarize each of our datasets in Table 3.1 and Table 3.2.

As an example of how we use multiple views with different tradeoffs, consider the differences between the DITL packet traces (containing 51.9 billion queries across 50,000 ASes) and our local DNS / activity measurements (10 thousand measurements, 2 users). DITL allows us to see, globally, how recursive resolvers interact with the root DNS, allowing us to make definitive statements about global inflation and query volumes. However, DITL does not tell us how individual users interact with the root DNS, and translating DITL queries to user experience requires heuristic arguments about caching (§3.3.3). Our local DNS and activity measurements, although limited, give us precise reference points for how root DNS factors into everyday Internet browsing experience, which we find valuable.

3.2 Routes to the Root DNS Are Inflated

Earlier work has found query distance to the root DNS is often significantly inflated [26, 75, 117, 77, 27]. Similar to this work, we find that queries often travel to distant sites despite the presence of a geographically closer site. We extend this understanding in a number of ways. While previous work considered only subsets of root DNS activity and focused on geographic inflation for recursives rather than users, we calculate inflation for nearly all root letters, and place inflation in the context of *users*, rather than recursive resolvers. These contributions are significant for several reasons. First, considering more root letters allows us to evaluate inflation in different deployments, and with most letters we can evaluate the root DNS *system*. Since a recursive makes queries to many root letters, favoring those with low latency [130], *system* performance and inflation can (and does) differ from component performance. Second, we weight recursive resolvers by the number of users, which allows us to see how users are affected by inflation. Finally, we extend prior work by conducting an analysis of latency (as opposed to geographic) inflation with large coverage.

Previous studies of `anycast` have separated inflation into two types, `unicast` and `anycast`, in an attempt to tease out how much latency `anycast` specifically adds to queries [75, 29, 77, 27]. For several reasons, we choose to consider inflation relative to the deployment, rather than try to infer which inflation would exist in an equivalent `unicast` deployment. First, coverage of measurement platforms used to determine `unicast` inflation such as RIPE Atlas (vantage points for `anycast` studies [77, 27]) is not representative [131]. Second, calculating `unicast` inflation requires knowledge of the best `unicast` alternative from every recursive seen in DITL to every root letter, something that would be difficult to approximate with RIPE Atlas because some letters do not publish their `unicast` addresses. Third, we find it valuable to compare latency to a theoretical lower bound, since user routes to the best `unicast` alternative may still be inflated.

We measure two types of inflation for the root DNS, by looking at which sites recursive resolvers are directed to. DITL captures are a rich source of data because they provide us with a

global view of which recursives access which locations (§3.1.1). Our inflation analysis covers 224 countries/regions and 22,243 ASes (Atlas covers about 3,700 ASes as of July 2021).

We calculate the first type of inflation – geographic inflation (Eq. (3.1)) – over 10 of the 13 root letters, omitting G which does not provide data, H which only had one site in 2018 (and so has zero inflation), and I, where anonymization prevents analysis. Geographic inflation measures, at a high level, how users are routed to sites compared to the closest front-end (*i.e.,* efficiency).

We calculate the second type of inflation – latency inflation (Eq. (3.2)) – over the root letters mentioned above by looking at the subset of DNS queries that use TCP, using the handshake to capture RTT [132]. Our latency inflation analysis further excludes D and L root, due to malformed DITL PCAPs. Latency inflation uses measured latencies to determine inflation, so it reflects constraints due to physical rights-of-way and connectivity, bad routing, and peering choices. We calculate median latency over each ⟨`root, resolver /24, anycast site`⟩ for which we have at least 10 measurements, providing us latencies for resolvers representing 40% of DITL query volume to these roots.

3.2.1 Methodology

To calculate geographic inflation, we first geolocate all recursives in our DITL∩CDN dataset using MaxMind [133], following prior methodology which affirmed MaxMind to be suitably accurate for geolocating recursive resolvers in order to assess inflation [27]. We then compute geographic inflation (scaled by the speed of light in fiber) for each recursive sending queries to root server j as

$$GI(R, j) = \frac{2}{c_f}\left(\sum_i \frac{N(R, j_i)d(R, j_i)}{N(R, j)} - \min_k\ d(R, j_k)\right) \tag{3.1}$$

where $N(R, j_i)$ is the number of queries to site j_i by recursive R, $N(R, j) = \sum_i N(R, j_i)$ is the total number of queries to all sites j_i in root j by recursive R, c_f is the speed of light in fiber, the factor of 2 accounts for the round trip latency, $d(R, j_k)$ is the distance between the recursive resolver and site j_k, and both the summation and minimization are over the global sites

in this letter deployment (see Section 3.1.1 for the distinction between local and global). We only consider global sites, since we do not know which recursives can reach local sites. For recursives which can reach a local site but instead reach a global site, Equation (3.1) (and Equation (3.2)) may underestimate actual inflation.

$GI(R, j)$ is an approximation of the inflation one would expect to experience when executing a single query to root deployment j from recursive R, averaged over all sites. The overall geographic inflation of a recursive is then the empirical mean over all roots. Even though queries from the same recursive /24 are usually routed together, they may be routed to different sites due to load balancing in intermediate ASes (see Section 3.8.1 for measures of how often this occurs), so we average geographic inflation across sites for a recursive. Geographic inflation is useful to investigate since it shows how our results compare with prior work, how many users are being inflated, and it gives us a measure of "efficiency" (§3.6.2) .

We also calculate latency inflation, again considering recursive querying patterns seen in DITL. We calculate latency inflation $LI(R, j)$ for users of recursive R to root j as

$$LI(R, j) = \sum_i \frac{N(R, j_i) l(R, j_i)}{N(R, j)} - \frac{3 \times 2}{2c_f} \min_k d(R, j_k) \tag{3.2}$$

where $l(R, j_i)$ is the median latency of recursive R towards root site j_i and the other variables are as in Equation (3.1). Prior work notes that routes rarely achieve a latency of less than the great circle distance between the endpoints divided by $\frac{2c_f}{3}$ [134], so we use $\frac{2c_f}{3}$ to lower bound the best latency recursives could achieve. Latency inflation is a measure of potential performance improvement users could see due to changes in routing or expanding the physical Internet (*e.g.,* laying fiber).

One limitation is that we do not account for the fact that the source addresses of some queries in the DITL traces may be spoofed. Spoofing is more likely to make our calculated inflation larger, especially in cases where the spoofer is far away from the physical interface it is spoofing (*i.e.,* from our perspective, the route looks inflated when actually the source address was spoofed). We do not attempt to correct for these cases since it would be difficult to distinguish between legitimately

poor routing and spoofed traffic.

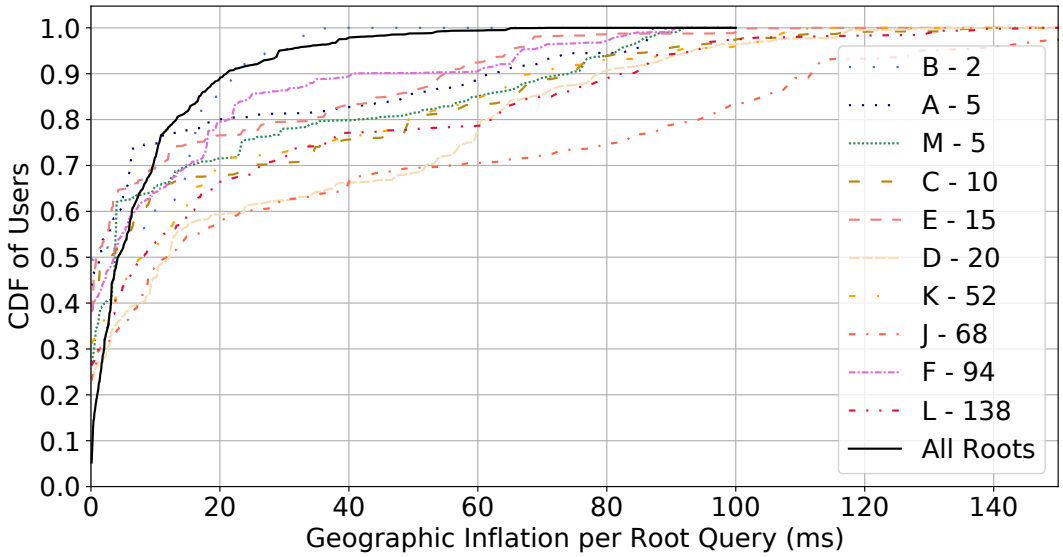
3.2.2 Results

Figure 3.3a demonstrates that the likelihood of a root DNS query experiencing any geographic inflation (Eq. (3.1)) roughly grows with deployment size (y-axis intercept), expanding on results in prior work which presented an orthogonal, aggregated view [27]. The `All Roots` line takes into account that each recursive spreads its queries across different roots. It has the lowest y-intercept of any line in Figure 3.3a, which implies that nearly every recursive experiences some inflation to at least one root and that the set of inflated recursives varies across roots. Hence, our analysis shows that nearly every user will (on average) experience inflation when querying the root DNS, and 10.8% of users are likely to be inflated by more than 2,000 km (20 ms).

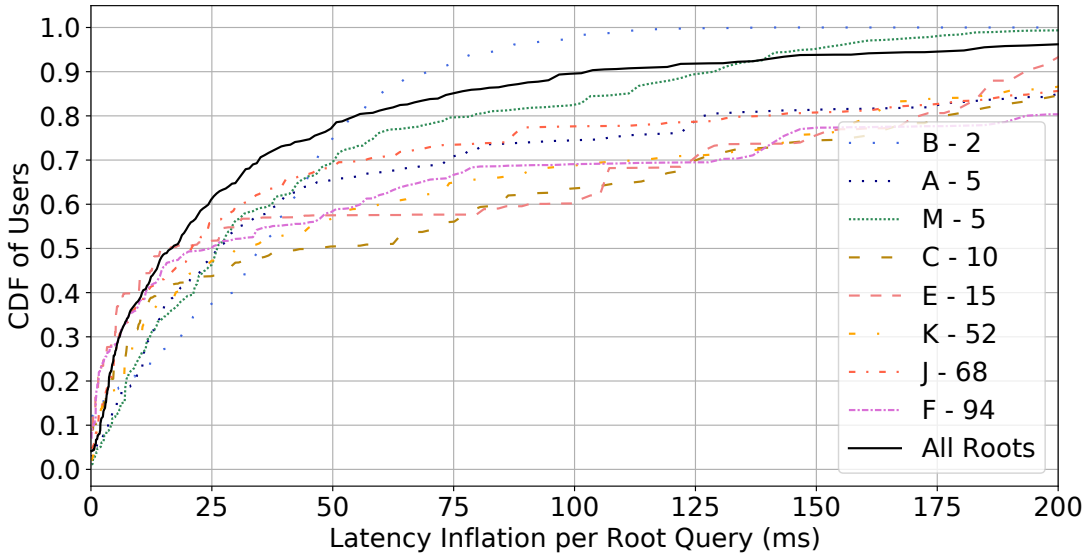
Figure 3.3b shows that queries to these roots experience frequent latency inflation (Eq. (3.2)), with between 20% and 40% of users experiencing greater than 100 ms of inflation (B root is a clear exception, but only had 2 sites, so inflation is less meaningful). Latency inflation starts at approximately zero, which follows from our choice of "optimal" latency (Eq. (3.2)). Compared to geographic inflation, latency inflation is particularly larger in the tail. For example, at the 95^{th} percentile `C` root has 240 ms of latency inflation but only 70 ms of geographic inflation. However, inflation for *the root DNS* as a whole is not as bad as individual root letters as shown by lines `All Roots`, which take into account that recursives can preferentially query low latency root servers [130].

Our latency inflation metric shows `C` root is more inflated than previously thought, inflating 35% of users by more than 100 ms compared to 20% reported in prior work [27] (although the comparison to prior work is not perfect since what was measure is different). Other prior work found significant inflation in the roots, but it is difficult to directly compare results since inflation was presented in different ways [77, 117].

Clearly, routing to individual root letters often is inflated, with many queries traveling thousands more kilometers than needed, and being inflated by hundreds of milliseconds for some users.



(a)



(b)

Figure 3.3: Inflation measured using geographic information (3.3a) and TCP RTT estimates (3.3b). Generally, larger deployments are more likely to inflate paths, and inflation in the roots is quite large. The legends indicate the number of global sites per letter during the 2018 DITL.

3.3 Root DNS Latency and Inflation Hardly Matters

With a richer understanding of inflation in the root DNS, one might wonder why inflation in root letters is large given growing deployments and root DNS's importance in the Internet. We now show that root DNS inflation does not result in much *user-visible* latency.

3.3.1 Measuring Root DNS Latency Matters

The root DNS servers host records for TLDs (*e.g.,* COM, ORG). There are approximately one thousand TLDs, and nearly all of the corresponding DNS records have a TTL of two days. Hence, due to shared caches at local resolvers, one might think root DNS latency *trivially* does not matter for users. Recent work even suggests the root DNS can be done away with entirely [135] or largely replaced by preemptive caching in recursives [136]. We offer several reasons why we found it necessary to explicitly measure root DNS latency's impact on users, rather than use intuition.

First, there is a lot of attention being placed on the root DNS in the professional and research communities. For example, some experts have asked us in conversation why CDNs use `anycast`, when `anycast` inflates latencies in the root DNS so much. The SIGCOMM 2018 paper "Internet Anycast: Performance, Problems, & Potential" has drawn attention to the fact that `anycast` can inflate latency to the root DNS by hundreds of milliseconds [27]. Blog posts from the root letters discuss latency improvements and inflation reductions [137, 138, 139, 140] – why does latency matter to roots? Moreover, over the past 5 years the number of root DNS sites has steadily increased to more than double, from 516 to 1367. Why is there so much investment in more sites?

Second, there is value in quantitatively analyzing systems, especially global systems that operate at scale, even if we can intuitively, qualitatively reason about these systems without conducting analysis. We conduct analysis using data from eleven of thirteen root letters, giving us a truly global view of how users interact with the root DNS. We are aware of only one other study which looked at how caching affects root DNS queries [86], but that study is old, is limited to one recursive resolver, and does not place DNS queries in the context of user experience.

Third, although TTLs of TLD records are two days, recursive resolver implementations can be buggy. We noticed millions of queries per day for TLD records being sent to the root letters by some recursives (§3.3.3), and found a bug in the popular BIND recursive resolver software that causes unnecessary queries to the roots (Section 3.8.3). Hence, making arguments about root DNS latency requires careful analysis.

3.3.2 How We Measure Root DNS

Measuring how root DNS latency affects users poses several challenges. To put root DNS latency into context we must understand (1) how user-application performance is affected when applications make root queries, (2) how often end-hosts and recursive resolvers interact with root DNS, given their caches, (3) what the latency is from the `anycast` deployment, and (4) how these effects vary by location and root letter. These challenges both motivate our subsequent analyses and also highlight the limitations of prior work which do not capture these subtleties of root DNS latency [117, 43, 77, 27].

Therefore, precisely determining how root DNS latency affects users would require global, OS-level control to select recursives and view OS DNS caches; global application-level data to see when DNS queries are made and how this latency affects application-performance; global recursive data to see caches, root queries, and their latencies; and global root traces to see how queries to the roots are routed. As of November 2024, only Google might have this data, and assembling it would be daunting.

To overcome these challenges we take two perspectives of root DNS interactions: local (close to the user) and global (across more than a billion users). Our local perspective precisely measures how root DNS queries are amortized over users browsing sessions, while our global analysis estimates the number of queries users worldwide execute to the roots.

3.3.3 Root DNS Latency Hardly Matter

Local Perspective: To obtain a precise measure of how root DNS queries are amortized over a small population, we use packet captures of a recursive resolver at ISI (§3.1.1). We also measure from two authors' computers to observe how an individual user interacts with the root servers (with no shared cache), since ISI traces do not give us context about user experience. Data from two users is limited, which is a reflection of the challenges we identified in Section 3.3.2. However, these experiments offer *precise* measures of how these authors interact with root DNS (which no prior work has investigated), supplementing the global-scale data used for most of this chapter.

Using traces gathered at ISI, we calculate the number of queries to any root server as a fraction of user requests to the recursive resolver. We call this metric the root cache miss rate, as it approximates how often a TLD record is not found in the cache of the recursive in the event of a user query. It is approximate because the resolver may have sent multiple root requests per user query, and some root requests may not be triggered by a user query. The daily root cache miss rates of the resolver range from 0.1% to 2.5% (not shown), with a median value of 0.5%. The overall cache miss rate across 2018 was also 0.5%. The particular cache miss rate may vary depending on user querying behavior and recursive resolver software, but clearly the miss rate is small, due to shared caches. Section 3.8.2 shows the minimal impact root DNS latency has on users of ISI and a CDF of DNS latency experienced by users at ISI.

Since the measurements at ISI can only tell us how often root DNS queries are generated, we next look at how root DNS latency compares to end-user application latency. On two authors' work computers (in separate locations), we direct all DNS traffic to local, non-forwarding, caching recursive resolvers running BIND 9.16.5 and capture all DNS traffic between the user and the resolver, and between the resolver and the Internet.

We run the experiment for four weeks and observe a median daily root cache miss rate of 1.5% – similar to but larger than the cache miss rate at ISI. The larger cache miss rate makes sense, given the local users do not benefit from shared caches. We also use browser plugins to measure median daily active browsing time and median daily cumulative page load time, so we can place

DNS latency into perspective. Active browsing time is defined as the amount of time a user spends interacting with the page (with a 30 second timeout), whereas page load time is defined as the time until the window.onLoad event. Median daily root DNS latency is 1.6% of median daily page load time and 0.05% of median daily active browsing time, meaning that root DNS latency is barely perceptible to these users when loading web pages, even without shared caches. In general, we *overestimate* the impact of DNS and root DNS latency since DNS queries can occur as a result of any application running on the authors' machines (not just browsing).

Global Perspective: Towards obtaining a global view of how users interact with the root DNS, we next look at global querying behavior of recursives. As discussed in Section 3.3.2, it is difficult to model caching at resolvers and how caching saves users latency, since caching hides user query patterns (by design) and differs with recursive implementation. To overcome this challenge, we use a new methodology that amortizes queries over large user populations, by joining DNS query patterns with user data.

Given query volumes towards root servers from recursives and user counts using each recursive from the DITL captures (§3.1.1), we estimate the number of queries to the roots that users wait for per day. Figure 3.4 is a CDF of the expected number of queries per user per day, where lines `CDN` and `APNIC` use a different user-count dataset (§3.1.1), and line `Ideal` uses hypothetical assumptions which we describe below. Figure 3.4 demonstrates that most users wait for no more than one query to the roots per day, regardless of which user data we use.

To generate each line in Figure 3.4, we divide (*i.e.,* amortize) the number of queries to the root servers made by each recursive by the number of users that recursive represents. We weight this quotient (*i.e.,* daily queries per user) by user count and calculate the resulting CDF. We calculate the number of queries per day each recursive makes from DITL by first calculating daily query rates at each site (*i.e.,* total queries divided by total capture time) and subsequently summing these rates across sites. We include nearly every root query captured across the root servers, so Figure 3.4 provides a truly global view of how users interact with the root DNS.

The two lines `CDN` and `APNIC` correspond to amortizing DITL queries over Microsoft and

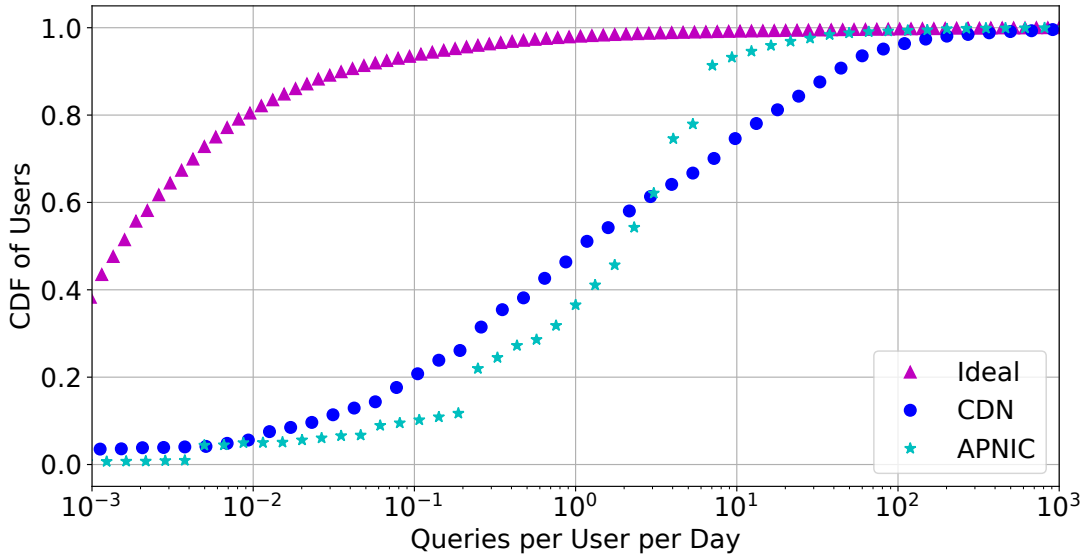


Figure 3.4: A CDF of the number of queries each user executes to the roots per day. The CDN and APNIC lines represent different user-count datasets. The Ideal line presents an idealized assumption about recursive query behavior. Most users wait for less than one query to the roots per day, regardless of which user data we use.

APNIC user counts, respectively. Hence, the set of 'users' each line represents is technically different, but we place them on the same graph for comparison. Even though the two methodologies of estimating user counts behind root queries are very different (CDN uses an internal measurement system, while APNIC uses Internet population estimates by country), amortizing queries over these sets of users still yields the same high level conclusions about how users interact with the root DNS, suggesting that our methodology and conclusions are sound – users *rarely* interact with the root DNS executing about one query per day at the median. Users in the tail are likely either spammers, have buggy recursive software, or represent recursives with more users than DITL∩CDN suggests (*e.g.,* cellular networks). APNIC user estimates are not affected by NATs, and APNIC has a smaller tail.

The line labeled Ideal does not use DITL query volumes to calculate daily user query counts, but instead represents a hypothetical scenario in which each recursive queries for all TLD records exactly once per TTL, and amortizes these queries uniformly over their respective user populations (we use Microsoft user counts for Ideal). The resulting hypothetical median daily

query count of 0.007 could represent a future in which caching works at recursives optimally – not querying the roots when not necessary. `Ideal` also demonstrates the degree to which the assumption that recursives only query once per TTL *underestimates* the latency users experience due to the root DNS (§3.3.2) – the assumption is orders of magnitude off from reality.

Effect of Removing Invalid TLD Queries

In Section 3.3 we estimate the number of queries users experience due to the root DNS by amortizing queries over user populations. Out of 51.9 billion daily requests to all roots, we observe 31 billion daily requests for bogus domain names and 2 billion daily requests for PTR records. We choose to not count these towards user query counts, because we believe many of these queries do not lie on the critical path of user applications and so do not cause user-facing latency. This decision has a significant effect on conclusions we can draw, decreasing daily query counts to root DNS resolution by 20×.

We base this decision on prior work which investigated the nature of queries with invalid TLDs landing at the roots. ICANN has found that 28% of queries for non-existent domains at L root result from captive-portal detection algorithms in Chromium-based browsers [121]. Researchers at USC have found that more than 90% of single-string (not separated by dots) queries at the root match the Chromium captive-portal pattern [91]. We remove captive-portal detection queries from consideration since they occur on browser startup and network reconnect, not during regular browsing, and they can occur in parallel with browsing.

Some might argue that queries for invalid TLDs *are* associated with user latency because typos for URLs (when typing into a browser search bar, for example) cause users to generate a query to the root servers. However, typos only generate a query to the root server if the TLD is misspelled (as opposed to the hostname). Hence typos, in general, cause users latency, but only specific typos will cause users *root* latency. Moreover, prior work has found that approximately 60% of queries for invalid TLDs reaching root servers are for domains such as `local`, `no_dot`, `belkin`, and `corp` [122]. It is unlikely these queries are caused by typos, since they are actual

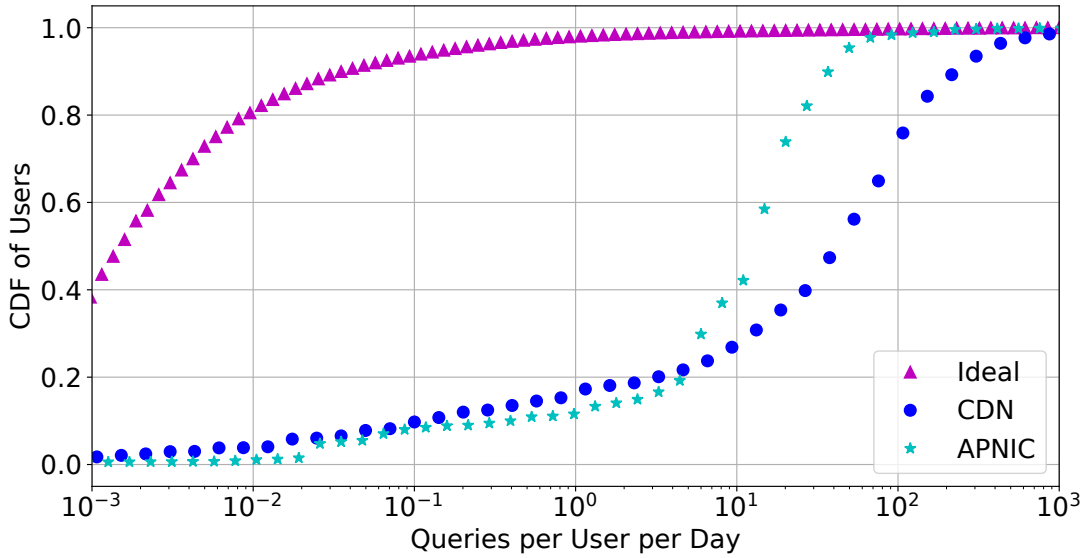


Figure 3.5: Daily queries by users to the root DNS, calculated by amortizing root DNS requests over user populations, when including or excluding queries for invalid TLDs. Counting invalid queries drastically increases median daily query counts to 22 (CDN), a 20-fold increase, or to 6 (APNIC), a 6-fold increase, depending on which user data we use.

(as opposed to misspelled) words and resemble domains often seen in software or in corporate networks. Chromium queries and queries for a certain set of invalid TLDs therefore account for around 86% of all queries for invalid TLDs at the roots, suggesting the vast majority of queries we exclude are not directly associated with user latency.

Nevertheless, it is still valuable to assess how including these queries for invalid TLDs changes the conclusions we can make about root DNS latency experienced by users. Figure 3.5 shows daily user latencies due to root DNS resolution when we include requests for invalid TLDs and PTR records in daily query volumes. Using CDN user counts, users experience a median of 22 queries to the root DNS each day – about 20× more than when we exclude requests for invalid queries (§3.3). This drastic 20-fold increase is surprising given we only (roughly) double the amount of queries by including invalid queries. The difference is best explained by the fact that a majority of invalid queries are generated by /24s with a large number of users. Since the y-axis of Figure 3.5 is the number of users (not /24s), counting invalid queries shifts the graph far to the right. Hence, counting invalid queries drastically affects the conclusions we can draw. There is

a less severe 6-fold increase in the number of queries per user per data calculated using APNIC data. Overall, including invalid TLD queries drastically changes our quantitative conclusions about user interaction with the root DNS but may not change our qualitative conclusions, since 20 queries a day to the roots is still small.

We have shown root DNS latency, and therefore inflated routes to the roots, makes no difference to most users. This result raises the question – are paths to the roots inflated because anycast intrinsically results in inflation? Or rather, does latency not mattering in this setting lead to anycast deployments that are not optimized for latency and hence tend to have inflated routes? To answer these questions, we turn to a new system using anycast to serve latency-sensitive content – Microsoft's CDN.

3.4 Latency Matters For Microsoft's CDN

We demonstrate that latency (and hence inflation) *does* matter for Microsoft users when fetching web content, unlike for most users in the root DNS, principally due to the number of RTTs users incur when fetching web content.

3.4.1 RTTs in a Page Load

To estimate the latency a user experiences when interacting with Microsoft's CDN (§3.4.3), we first estimate the number of RTTs required to load a typical web page hosted by Microsoft's CDN.

The number of RTTs in a page load depends on a variety of factors, so we aim to lower bound the number. We lower bound the number of RTTs since a lower bound is a conservative measure of the impact of CDN inflation, as the latency inflation accumulates with each additional RTT, and larger pages (more RTTs) would be impacted more. We provide an estimate of this lower bound based on modeling and evaluation of a set of web pages hosted by Microsoft's CDN using Selenium (a headless web browser), finding that 10 RTTs is a reasonable estimate. We now detail these experiments.

3.4.2 Number of RTTs in a Page Load

To estimate the latency a user experiences when interacting with Microsoft's CDN (§3.4.3), we first estimate the number of RTTs required to load a typical web page hosted by Microsoft's CDN. The number of RTTs in a page load depends on a variety of factors, so we aim to find a reasonable *lower bound* on the number of RTTs users incur for typical pages. A lower bound on the number of RTTs to load pages is a conservative measure of the impact of CDN inflation, as latency inflation accumulates with each additional RTT, and larger pages (more RTTs) would be impacted more. We provide an estimate of this lower bound based on modeling and evaluation of a set of web pages hosted by Microsoft's CDN using Selenium (a headless web browser), finding that 10 RTTs is a reasonable estimate. We scale latency by the number of RTTs in Section 3.4.3 to demonstrate how improvements in latency help users (and, conversely, how inflation hurts users).

Users incur latency to Microsoft's CDN when they download web objects via HTTP. We calculate the number of RTTs required to download objects in each connection separately, and sum RTTs over connections while accounting for parallel connections. For a single TCP connection, the number of RTTs during a page load depends on the size of files being downloaded. This relationship is approximated by

$$N = \left\lceil log_2 \frac{D}{W} \right\rceil \tag{3.3}$$

where N is the number of RTTs, D is the total number of bytes sent by the TCP connection from the server to the user, and W is the initial congestion window size in bytes [141, 142]. Although W is set by the server, Microsoft and a majority of web pages [143] set this value to approximately 15 kB so we use this value. We do not consider QUIC or persistent connections across pages in detail here, but larger initial windows will result in fewer RTTs. We test mostly landing pages, for which persistent connections are uncommon. Moreover, such considerations likely would not change our qualitative conclusions about how users experience CDN latency.

We make the following assumptions to establish a *lower bound* on N: (1) we do not account

for connections limited by the receive window or the application, as the RTT-based congestion window limitation we calculate is still a lower bound, (2) TCP is always in slow start mode, which implies the window size doubles each RTT and serves as a lower bound on the actual behavior of Microsoft's standard CUBIC implementation, and (3) all TCP and TLS handshakes after the first do not incur additional RTTs (*i.e.,* they are executed in parallel to other requests).

Modern browsers can open many TCP connections in parallel, to speed up page loads. Summing up RTTs across parallel connections could therefore drastically overestimate the number of RTTs experienced users. To determine the connections over which to accumulate RTTs, we first start by only considering the connection with the most data. We then iteratively add connections in size-order (largest to smallest) that do not overlap temporally with other connections for which we have accumulated RTTs. The 'data size' of a connection may represent one or more application-layer objects.

We load nine web pages owned by Microsoft, twenty times for each page. We choose popular pages hosted on Microsoft's CDN with dynamic content suggested to us by a CDN operator. We use Selenium and Chrome to open web pages and use Tshark [144] to capture TCP packets during the page load. When the browser's `loadEventEnd` event fires, the whole page has loaded, including all dependent resources such as stylesheets and images [145]. So, to calculate the total data size for each connection, we use the ACK value in the last packet sent to the server before `loadEventEnd` minus the SEQ value in the first packet received from the server. We then calculate the number of RTTs using Equation (3.3), and add a final two RTTs for TCP and TLS handshakes. We find only a few percent of CDN web pages are loaded within 10 RTTs, and 90% of all page loads are loaded within 20 RTTs, so 10 RTTs is a reasonable lower bound.

3.4.3 Microsoft's CDN User Latency

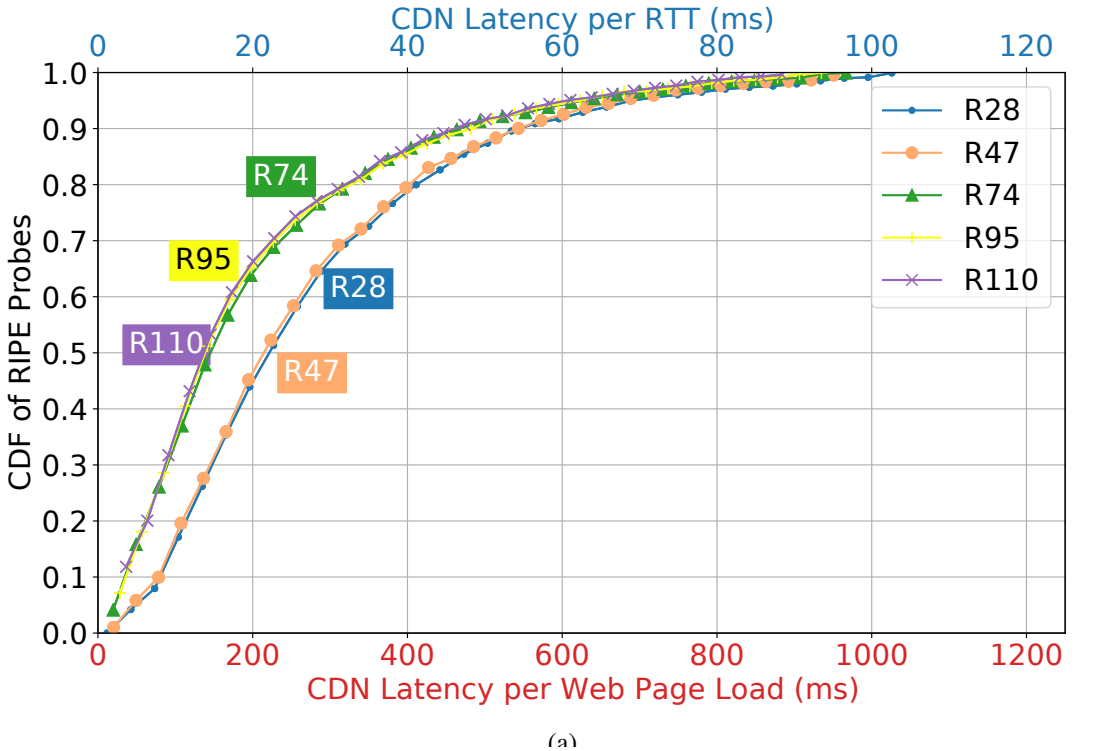
We now measure how users are impacted by latency of Microsoft's CDN. First, using measurements from RIPE Atlas probes, we demonstrate that CDN latency results in significant delay to users when fetching web content. Then, using both client-side measurements and server-side

logs, we also show that latency usually decreases with more sites. Consequently, Microsoft has a major incentive to limit inflation experienced by users, and investments in more `anycast` sites positively affect user experience much more in the case of Microsoft's CDN than in the roots. The positive effect on user experience has been a major reason for recent expansion (§3.6.3).

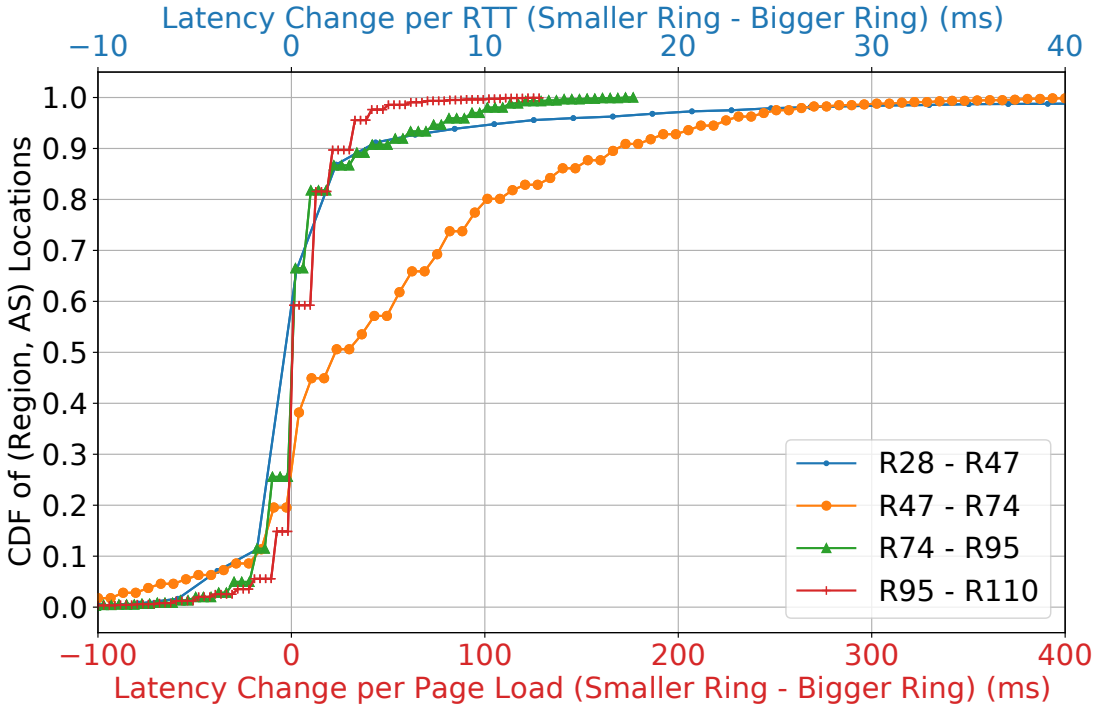
Microsoft's CDN has groups of sites called *rings* (§3.1.2). Each larger ring adds some sites to those of the smaller ring. Each ring provides an IP `anycast` CDN, so we report results for each of the rings individually. Different ring sizes reflect some of the benefit of additional `anycast` locations, but a user's traffic usually ingresses to Microsoft's network at the same site regardless of ring, since all routers announce all rings. Users experience latency from Microsoft's as they retrieve web objects (*e.g.,* web pages or supporting data) hosted by Microsoft's CDN. Hence, in order to assess how Microsoft users experience latency, we must measure what the RTT is from users to front-ends and how many RTTs are incurred when fetching web content. We use our estimate from Section 3.4.1 that users incur *at least* 10 RTTs in a page load. To obtain per-page-load latency, we scale `anycast` latency by the number of RTTs.

In Figure 3.6a, we show latencies to rings. Figure 3.6a uses latencies measured from RIPE Atlas probes (§3.1.2), as we cannot share absolute latencies from Microsoft measurements since Microsoft considers this data proprietary. Although RIPE Atlas has limited coverage [131], we compare (but cannot share) to CDN measurements, which contain latencies from all ⟨`region, AS`⟩ locations to all rings. We observed that the distribution of RIPE Atlas probe latencies is overall somewhat lower than that of Microsoft's users globally (not shown in figure), so Figure 3.6a likely underestimates the latency users typically experience.

Users can experience up to 1,000 ms in `anycast` latency per page load, and, for large deployments (*e.g.,* `R95`), half of RIPE Atlas probes experience approximately 100 ms of latency per page load (Fig. 3.6a). Therefore, unsurprisingly, latency to Microsoft's CDN factors into user experience, and so Microsoft has an incentive to decrease latency for users. The difference in median latency per page load between `R28` and `R110` is approximately 100 ms, which is a measure of how investments in more front-ends can help users. Similarly, a root deployment with more sites



(a)



(b)

Figure 3.6: RTTs and latencies per web page load from RIPE probes to CDN rings (3.6a), and change in median latency for Microsoft users in ⟨`region, AS`⟩ locations when transitioning rings (3.6b). Axes with per-RTT latencies are blue, while axes with per-page-load latencies are red. Latencies per page load can be significant, so Microsoft has an incentive to reduce inflation.

tends to have lower latency than a root deployment with fewer sites (§3.6.2), but such reductions in latency hardly affect user experience (§3.3).

Latency benefits with more sites are not uniform, and performance falls into one of two "groups" – R28 and R47 have similar aggregate performance, as do R74, R95, and R110. This grouping corresponds to the way rings "cover" users – R74 provides a *significant* additional number of Microsoft users with a geographically close front-end over R47 (§3.6.2).

To show how adding front-ends tends to help individual ⟨region, AS⟩ locations (in addition to aggregate performance), Figure 3.6b shows the difference in median latency for a ⟨region, AS⟩ location from one ring to the next larger ring, calculated using CDN measurements (as opposed to RIPE Atlas probes). Most ⟨region, AS⟩ locations experience either equal or better latency to the next largest ring, with diminishing returns as more front-ends are added. A small fraction of users experience small increases in latency when moving to larger rings – 90% of users experience a decrease of at most a few millisecond increase and 99% experience less than a 10 ms increase. Hence, Microsoft does not sacrifice fairness for performance improvements.

We next investigate if Microsoft's clear incentive to reduce latency (and therefore inflation) translates to lower inflation from users to Microsoft's CDN than from users to the root DNS.

3.5 Anycast Inflation Can Be Small

We next investigate whether Microsoft's incentive to reduce inflation translates to an anycast deployment with less inflation than in the roots, representing the study of anycast CDN inflation with the best coverage to date – measurements are from billions of users in hundreds of countries/regions and 59,000 ASes. Critically, we are able to directly compare inflation between root DNS and Microsoft's CDN, since we use the same methodology with broad coverage.

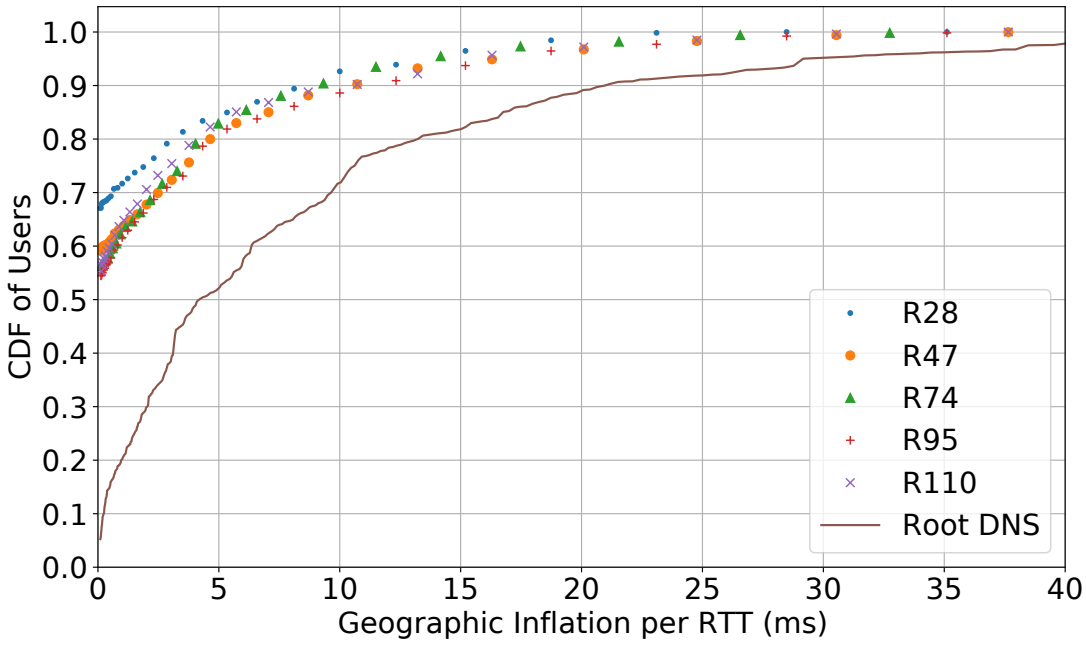
To measure anycast inflation for Microsoft's CDN we use geographic information and server-side measurements (§3.1.2). Server-side logs give us a global view of which clients hit which front-ends and the latencies they achieved. Latency is measured via server-side logging of TCP round-trip times. Front-ends act as TCP proxies for fetching un-cached content from data

centers. Routing over the global WAN is near optimal [3], so measuring inflation using latency to front-ends (as opposed to measuring inflation using end to end latency) captures all routing inefficiency. We also use Microsoft user locations, which are determined using an internal database.

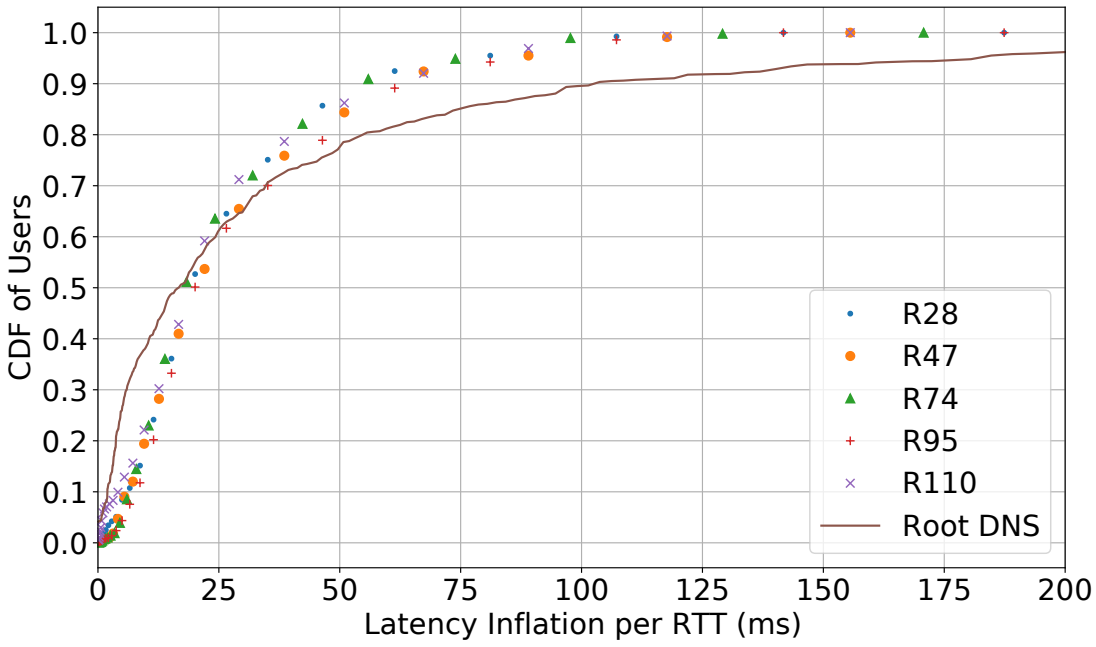
As in Section 3.2, we calculate both geographic and latency inflation. We calculate geographic inflation as in Equation (3.1), except all users in a ⟨`region, AS`⟩ location are assigned the mean location of users in the ⟨`region, AS`⟩ location. `Anycast` inflation results in extra latency for every packet (and corresponding ACK) exchanged between a client and an `anycast` service, resulting in a per RTT cost, so we refer to inflation as "per RTT". Application-layer interactions may incur this cost multiple times (as in the case of loading a large web object from a CDN) or a single time (as in the case of typical DNS request/response over UDP).

Microsoft users usually experience no geographic inflation (Fig. 3.7a, y-axis intercepts), and 85% of users experience less than 10 ms ($1,000\ km$) of geographic inflation per RTT for all rings. Conversely, 97% of root DNS users experience some geographic inflation, and 25% of users experience geographic inflation more than 10 ms ($1,000\ km$) per RTT. The fact that geographic inflation is larger and more prevalent in the roots than in Microsoft's CDN (at every percentile) suggests Microsoft optimizes its deployment to control it (§3.6).

We next calculate latency inflation for each ring as in Equation (3.2). We calculate median latencies over user populations within a ⟨`region, AS`⟩ location hitting a front-end in a given ring, the assumption being that measurements from some users in a ⟨`region, AS`⟩ location hitting the same site are representative of all users in that ⟨`region, AS`⟩ location hitting that site. More than 83% of such medians were taken over more than 500 measurements, so our observations should be robust. There is roughly constant latency inflation as the number of front-ends grows (Fig. 3.7b), which highlights that even though users have more low latency options (front-ends), they can still take circuitous routes to close front-ends. However, Microsoft is able to keep latency inflation below 30 ms for 70% of users in *all* rings and below 60 ms for 90% of users. In Microsoft's CDN, 99% of users experience less than 100 ms of inflation, but 10% experience more than 100 ms to the roots.



(a)



(b)

Figure 3.7: Inflation measured using geographic information (3.7a) and CDN server side logs (3.7b). Inflation is more prevalent for larger deployments but is still small for most users.

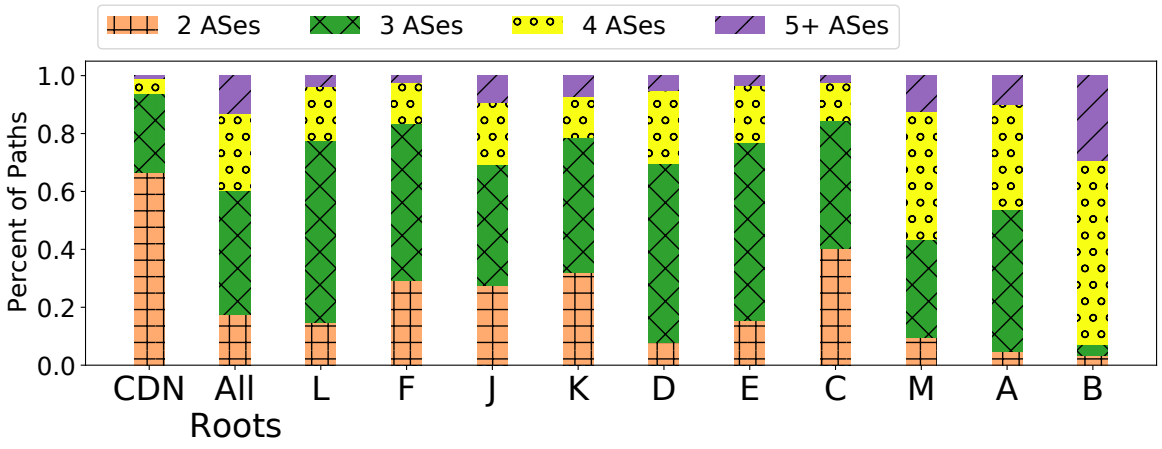
An interesting takeaway from Figure 3.7b is that system-wide per-query root DNS inflation is quite similar to CDN inflation, a fact that is not clear from prior work [27, 29] since prior work used different methodology and looked at fewer root letters. However, inflation in *individual* root letters is quite worse than in Microsoft's CDN (Fig. 3.3b). Although inflation in the roots does not matter to most users (§3.3.3), it is still interesting to see how recursive resolvers can take advantage of the thirteen independent deployments of root letters, and choose which letter is the best for them, in a way that is not possible in Microsoft's CDN.

Compared to prior work which also studied inflation in Microsoft's CDN [29], we find an improvement – 95% of users experience inflation under 80 ms now compared to 85% 5 years ago. This improvement (representing millions of users) is despite the fact that Microsoft's CDN has more than doubled in size and that we use a stricter measure of inflation, and is evidence that expansion reduces efficiency (in terms of % of users at their closest site) but inflation can be kept low through careful deployment (§3.6.2). Figure 3.7b also offers a complementary view of inflation compared to prior work [29], which does not take into account that routing from a ⟨`region, AS`⟩ location to *all* front-ends might be sub-optimal.

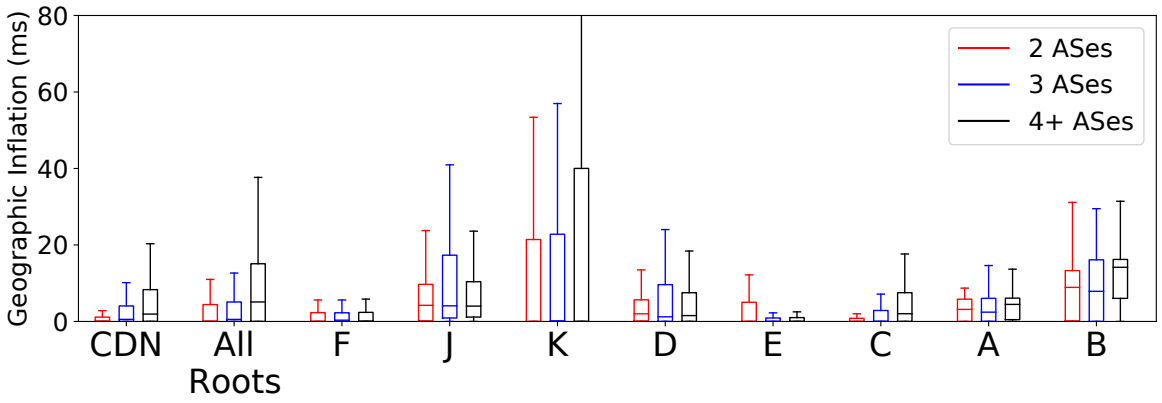
Compared to Figure 3.7a, Figure 3.7b demonstrates there is room for improvement – at least half of users visit their closest front-end, but those users might take circuitous routes to those front-end as shown by the low y-axis intercepts in Figure 3.7b. There is still room for latency optimization in `anycast` deployments, which is an active area of research [84, 146, 31].

3.6 Incentives and Investment Shape Deployments and Paths

We have definitively answered the questions regarding inflation that we posed at the end of Section 3.3.3. We now investigate why inflation is so different in root DNS and Microsoft's CDN by looking at path lengths (§3.6.1), investigate how geographical differences in deployments affect inflation (§3.6.2), and present reasons behind the expansion of both root DNS and CDNs (§3.6.3).



(a)



(b)

Figure 3.8: Distribution of the number of ASes traversed to reach various destinations (3.8a) and the correlation between the AS path length towards a destination and geographic inflation (3.8b). Microsoft is closely connected to many eyeball ASes, and this connectivity correlates with lower inflation. We group paths towards roots and Microsoft by ⟨`region, AS`⟩ locations, except for 'All Roots' which groups paths by ⟨`region, AS, root`⟩ locations.

3.6.1 Microsoft's CDN Has Shorter AS Paths, and Short AS Paths are More Direct

CDNs have a financial incentive to keep latency low for users and have the resources to build efficient systems. Microsoft deploys state-of-the-art network routing automation [9, 8], a global SDN WAN [4, 3], and expensive peering agreements when they make economic sense and/or help user experience. These strategies result in short, low latency routes between users and Microsoft.

We can capture some of these engineering efforts by measuring how Microsoft connects to users. CDNs peer widely with end-user networks and so have direct paths to many users [147, 28]. With fewer BGP decision points, paths are often less inflated [20]. This intuition motivates the following investigation of AS path lengths towards roots and Microsoft and of how path lengths relate to inflation, which is summarized by Figure 3.8. Figure 3.8 quantifies one key difference between root DNS and CDN deployments, but publicly available data cannot capture all of Microsoft's optimizations.

To quantify differences in AS path length between Microsoft and roots, Figure 3.8a shows AS path lengths to roots and Microsoft from RIPE Atlas probes. We use the maximum number of active RIPE Atlas probes for which we can calculate AS paths to all destinations, amounting to 7,200 RIPE Atlas probes in 158 countries/regions and 2,400 ASes. Although RIPE Atlas probes do not have representative coverage [131], it is the best publicly available system, and we are only interested in qualitative, comparative conclusions.

Lengths towards Microsoft's CDN are based on traceroutes from active Atlas probes in August 2020, whereas lengths towards the roots are based on traceroutes from RIPE Atlas probes in April 2018 (the time of DITL). We use AS path lengths from traceroutes towards the roots measured in 2018 in Figure 3.8, so that we can pair AS path length directly with 2018 DITL inflation data. We perform IP to AS mapping using Team Cymru [128], removing IP addresses that are private, associated with IXPs, or not announced publicly by any ASes. We merge AS siblings together into one 'organization'. We derive sibling data from CAIDA's AS to organization dataset [148]. We group paths by ⟨`region, AS`⟩ location, except for 'All Roots', for which we group paths by ⟨`region, AS, root`⟩ location. We assign each ⟨`region, AS`⟩ location equal weight; when

a given ⟨region, AS⟩ location hosts multiple RIPE Atlas probes that measure different path lengths to a given destination, the location's weight is split evenly across the measured lengths.

Figure 3.8a shows shorter paths to Microsoft than to the roots. (Weighting by traffic volumes yielded similar results.) 69% of all paths to Microsoft only traverse two ASes (direct from RIPE Atlas probe AS to destination AS), and only 5% of paths to Microsoft traverse four or more ASes. Conversely, between 5% and 44% paths to root letters only traverse two ASes, and between 12% and 63% of paths to roots traverse four or more ASes.

To demonstrate how short AS paths tend to have lower inflation, Figure 3.8b shows the correlation between AS path length and geographic inflation. We compare to geographic (as opposed to latency) inflation since we are able to calculate it for more root letters. For the inflation towards destinations in Figure 3.8b, we use the geographic inflation associated with that ⟨region, AS⟩ location calculated for Figure 3.3 and Figure 3.7a. The AS path length towards each destination is the most common AS path length measured across RIPE Atlas probes in the same ⟨region, AS⟩ location. Figure 3.8b demonstrates that paths that traverse fewer ASes tend to be inflated less. All Roots shows that this is true globally, across root letters, and the results for each individual root letter shows geographic inflation is less for paths traversing 2 ASes than it is for paths traversing more (except for B and E root). The relationship between inflation and AS path length is very different across root letters, which is evidence of different deployment strategies.

Overall, our results demonstrate that shorter paths tend to have less inflation, users have shorter paths to Microsoft than towards the roots, and Microsoft tends to have less inflation across path lengths. We believe these observations are a result of strategic business investments that Microsoft puts toward peering and optimizing its routing and infrastructure. In addition to shorter AS paths generally being less inflated [20], direct paths to Microsoft's CDN in particular sidestep the challenges of BGP by aligning the best performing paths with the BGP decision process [11]. Direct paths will usually be preferred according to BGP's top criteria, local preference and AS path length (because by definition they are the shortest and from a peer, and ASes usually set local preference to prefer peer routes in the absence of customer routes, which for Microsoft will only exist during

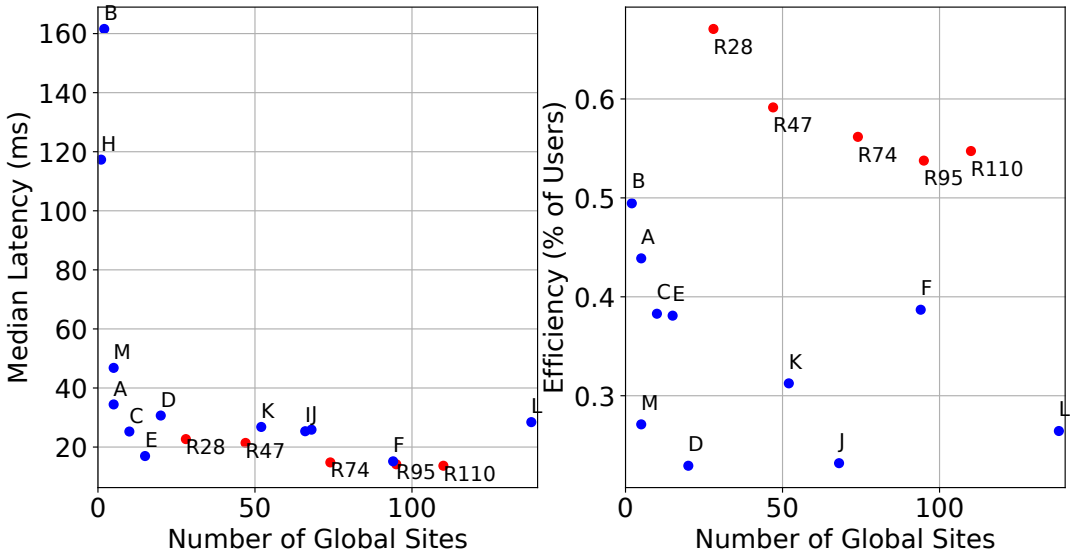
a route leak/hijack). Among the multiple direct paths to Microsoft that a router may learn when its AS connects to Microsoft in different locations, the decision will usually fall to lowest IGP cost, choosing the nearest egress into Microsoft. Microsoft collocates `anycast` sites with all its peering locations, and so the nearest egress will often (and, in the case of the largest ring, always) be collocated with the nearest `anycast` site, aligning early exit routing with global optimization in a way that is impossible in the general case or with longer AS paths [20]. At smaller ring sizes, Microsoft can use traffic engineering (for example, not announcing to particular ASes at particular peering points) when it observes an AS making poor routing decisions.

3.6.2 Larger Deployments are Less Efficient but Have Lower Latency

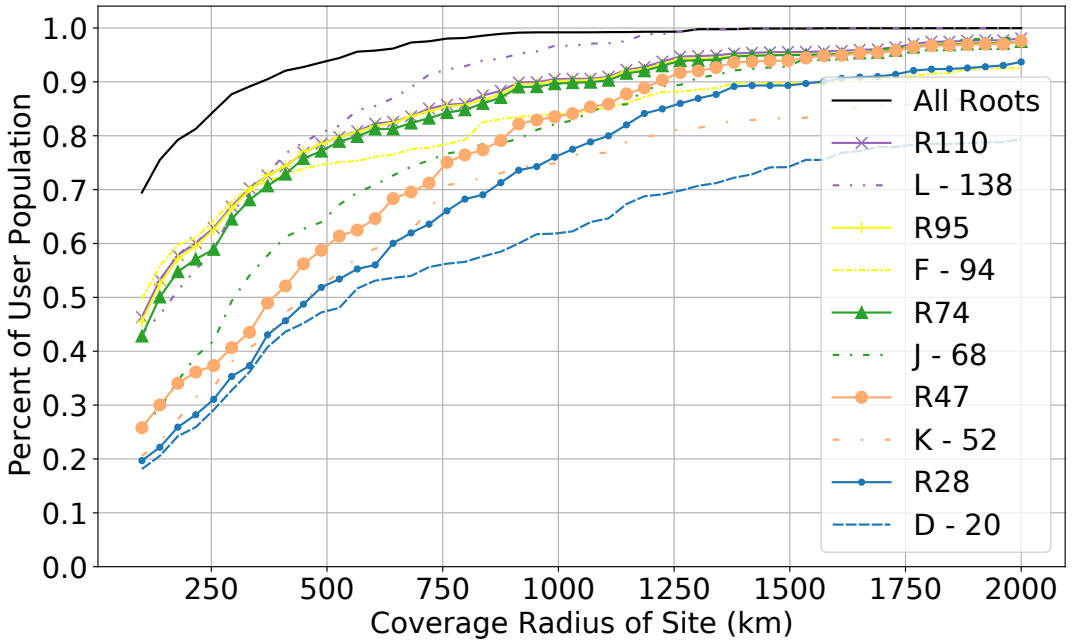
CDN latency in Figure 3.6a and inflation in Figure 3.7 reveal a relationship that some may find non-intuitive – as deployment size increases, inflation increases (less efficiency) but median latency decreases. We observe a similar effect in Figure 3.3 – larger root deployments tend to have more inflation but have lower latency. Intuitively, larger deployments are less efficient since BGP will make "wrong" decisions about which routes to export more often, and have lower latency since there are more low-latency options available to users. These results suggest efficiency may not be a useful metric for assessing performance.

We make these relationships explicit in Figure 3.9a which shows median latency and efficiency for each root letter and Microsoft ring. We define efficiency as the percentage of users with zero geographic inflation (*i.e.,* y-axis intercepts in Figure 3.3a and Figure 3.7a) since it is a rough measure of how optimal routing is (routing may not actually be optimal even if there is zero geographic inflation if users take a circuitous route to their closest site). Latency to root letters in Figure 3.9a is the median latency across all RIPE Atlas probes over an hour in 2018 (time of DITL) (*i.e.,* median per probe, then a median across probes), and latencies to rings are medians in Figure 3.6a.

The trend that efficiency decreases with deployment size is less clear in the root DNS than in Microsoft's CDN, likely since the root letters are run by different organizations and so have different deployment strategies which also impact latency and inflation. A counterexample to the



(a)



(b)

Figure 3.9: Larger deployments lead to lower latency (Fig. 3.9a-left) since they offer more low-latency options to users (Fig. 3.9b). However, fewer users visit their closest site (Fig. 3.9a-right) leading to more inflation.

trend is F root which had the lowest median latency (15 ms) in 2018 and good efficiency (39%). F root likely bucks the trend since F root partners with Cloudflare (a global CDN) and so benefits from a deployment tuned to lowering user latency. It is interesting that R95 and F root have similar number of sites and (low) median latency (approximately 15 ms), but that F root has considerably lower efficiency; hence, low efficiency is not necessarily bad. Conversely, high efficiency does not result in low latency; for example, 49% of users reach their closest B root site, but users still experience a high median latency to B root of 160 ms. Prior work looked at similar metrics to those in Figure 3.9a (right) for root letters using data from RIPE Atlas and arrived at very different conclusions [27], possibly since RIPE Atlas has limited coverage.

Part of what contributes to low latency is that organizations place sites close to users. Figure 3.9b shows what percent of Microsoft users are "covered" by a site in each ring and in a root letter of similar size, where "covered" means the closest site is within X km of users (x-axis). Hence, coverage implies there is a reasonably low latency option for users. Figure 3.9b is quite surprising – first, the root DNS as a whole (All Roots) has impressive coverage – 91% of Microsoft users are within 500 km of a root site (not even counting local sites!). Moreover, individual root letters can have even better coverage of Microsoft users than rings (L root has 94% of users within 1,000 km whereas R110 has 90%), which is interesting since L root, unlike R110, was not deployed specifically for Microsoft users. Figure 3.9b also demonstrates that approximating root DNS users with Microsoft users (Fig. 3.3) was fair, since root letters have decent coverage of Microsoft users. An exception is D root which did not have global sites in India at the time, where Microsoft has both `anycast` sites and a large user population to serve.

3.6.3 Differing Incentives Lead to Different Investments and Outcomes

We now discuss how incentives have shaped deployments and how our findings may extend to other `anycast` deployments.

Table 3.3: Survey results from root DNS operators. Most root letters indicate DDoS resilience and (surprisingly) latency have been major factors for growth, and that future growth will likely slow.

Past		**Future**	
Reason for Growth	**Number of Orgs**	**Future Growth Trend**	**Number of Orgs**
DDoS Resilience	9	Decceleration of Growth	4
Latency	8	Acceleration of Growth	1
ISP Resilience	5	Maintain Growth Rate	4
Other	3	Cannot Share	1

Drivers for Growth

We reached out to operators of both root DNS and Microsoft asking what fueled their recent growth and whether they think it will continue. Of the twelve organizations running a root DNS letter, 11 responded, and we summarize the main reasons root DNS letters expand in Table 3.3. Principally, roots grew to reduce latency and improve DDoS resilience.

Over the past 5 years the number of root DNS sites has more than doubled from 516 to 1367, steadily increasing. Surprisingly, Table 3.3 demonstrates latency *was* a primary reason for expansion for nearly all root letters. Our results suggest this reasoning does not stem from caring about user experience (§3.3.3) but perhaps from establishing a competitive benchmark with other root letters.

Root operators also indicated that growth was driven by a desire to improve resilience in two dimensions: DDoS and ISP resilience. DDoS resilience refers to increasing overall capacity so root letters can provide service in the face of DDoS attacks. ISP resilience refers to offering root sites in certain locations and networks so that service can still be offered even if connectivity to the rest of the Internet is severed. According to both operator responses and publicly available sources, growth additionally stems from open hosting policies (almost any AS can volunteer to host a new site) [149, 150, 151], and from teaming up with large CDNs like Cloudflare. Root operator responses about future plans for growth suggest that the increase of root DNS sites will slow in the coming years.

With such decentralized deployment (in part by design to promote resilience), coordinated

optimization of root DNS latency is difficult, even if latency optimization were a goal. By contrast, Microsoft's CDN is latency-sensitive and is centrally run. Operators optimize and monitor latency, thereby minimizing inflation (§3.5) with direct paths to many users (§3.6.1). Unlike some root letters, Microsoft does not publicly compare latency with other CDNs and instead considers client latency proprietary information. Construction of new front-ends often follows business needs to support new markets. These commercial motivations contrast with the above root DNS reasons for expansion, yet the number of front-ends for Microsoft's CDN has more than doubled in the past five years.

Other `Anycast` Systems

A key takeaway from our results is that one cannot generalize anycast performance results (including ours) to other systems using `anycast`. `Anycast` must be assessed in the context of the system in which it resides. Prior work took the results of one system (root DNS) and assumed it applied generally to a technique (`anycast`) which resulted in misleading conclusions [27]. It would be difficult to even extend our results to systems with similar deployments, since the degree to which performance improvements are due to the deployment and the degree to which they are due to tuning of route configurations is unknown [152].

Other systems using `anycast` include Akamai DNS authoritative resolvers [110], Google Cloud VMs [153], and Google Public DNS [112]. All of these services have different performance requirements for users; *i.e.,* they all want inflation to be "low" but how "low" it needs to be depends on the application. For example, Google Cloud VMs can host game engines which have much stricter latency requirements than fetching HTTP objects. We hope that future work will take these considerations into account when assessing `anycast`.

3.7 `Unicast` Introduces Reliability Concerns

`Unicast` is an alternative to `anycast` where Service Providers advertise one prefix per site and then direct clients to individual sites. Prior work suggests that using this strategy alongside

[32] or on top of [28, 29, 35] anycast can improve performance for users.

However, one reason why Microsoft's CDN uses `anycast` is reliability. If a site fails, BGP usually finds a route to a new site within tens of seconds [24], restoring availability for affected clients. The question we now ask is to what degree using `unicast` would change these availability properties. By analyzing residential traffic traces, we determine that these availability concerns can be quite significant.

3.7.1 Residential Network Data

To assess the ability of clouds to quickly redirect network traffic using DNS as a mechanism, 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 [154]—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 [155], tshark to extract non-sensitive information (*e.g.,* protocol, TLS SNI, DNS A record) [144], and cryptoANT to anonymize privacy-sensitive fields [156]. 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.

We passively captured all traffic sent to and from residential units during two separate 1 hour windows a day, and all DNS traffic from two separate 6 hour windows overlapping those two one-hour windows of data capture. We capture this traffic from 2022 to the present day. Packets with the same 5-tuple in the IP header (transport protocol, source IP, destination IP, source port,

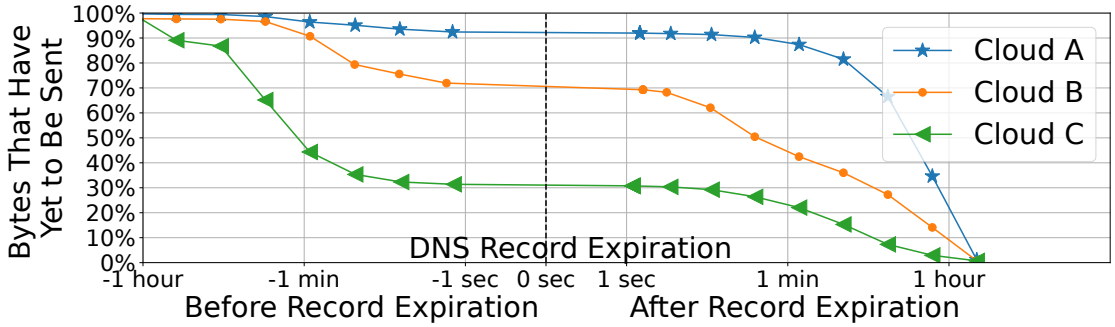


Figure 3.10: Of all traffic sent to Cloud A, 80% is sent at least 5 minutes after TTL expiration.

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. Our methodology for matching traffic to DNS records is similar to that in prior work [95].

Not Respecting DNS TTLs

A reliability problem that DNS introduces is that DNS records can live in caches for much longer than their TTLs (§2.2.4). Prior work demonstrates that many flows start after the corresponding DNS record has expired [33, 34, 157, 158], leaving these flows outside the control of the cloud to steer based on performance, availability, or load-balancing. However, we find that the situation is even bleaker than this prior work suggests. Even flows that start while a DNS record is valid may be extant after DNS expiration, further restricting cloud control over traffic. That is, if a cloud desires to shift traffic it not only has to wait for TTLs to expire but it also has to wait for existing connections to close.

Our analysis captures the impact of flows that send traffic after the DNS record has expired. We find that *most* traffic to many large Service Providers 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 [95]. More about our DNS measurement and analysis pipeline are in Section 3.7.1.

Figure 3.10 shows that, out of all the traffic sent to one of the largest cloud providers, 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 two other 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).

Using Multiple DNS Answers

Some queries have multiple DNS answers for a single query. Hence it is possible that, to avoid reliability problems, Service Providers can return more than one answer so that clients have more than one path. Efforts such as Happy Eyeballs standardize this technology [118].

However, Service Providers lack control over how clients use these multiple answers. For example, it is unclear which answer clients will choose to use by default, and it is unclear if/when they will switch among answers in the presence of either performance problems or failure. This behavior depends on a combination of the application, the operating system, and, in the case of web-based services, the browser.

To assess the degree to which Service Providers have control over this behavior, we measure how popular browsers, applications, and operating systems used multiple DNS records in the face of simulated failure (*i.e.,* whether they switch to a different IP address, and how long that switch takes). Specifically, we conduct measurements on 6 sample devices each running a different operating system, 3 browsers, and 4 popular applications. We find highly variable behavior depending on the operating system, application, and browser, suggesting that Service Providers cannot reliably use multiple DNS records as a way of guarding against failure or, in general, controlling exactly where traffic goes and when it goes there.

We test both the browser and application behavior of Hulu, WeChat, Slack, Grammarly, ChatGPT, and Netflix. We test these applications since they are popular, since they are of a few different types, and since they consistently use hostnames whose queries for corresponding A records contained multiple IP addresses in the response. We test failover behavior of these applications on

	Linux	Windows	MacOS	iOS (iPhone and iPad)	Android
Google Chrome	[3s, 80s]	[22s, 130s]	[75s, 120s]	<100 ms	[20s, 30s]
Firefox	[140s, 160s]	[20s, 30s]	<100 ms	<100 ms	<100 ms
Dedicated Applications	[40s, 75]	[40s, 50s]	<100 ms	<100 ms	<100 ms
Safari	N/A	N/A	<100 ms	<100 ms	N/A
Edge	[3s, 311s]	[21s, 22s]	[1s, 77s]	[250ms, 1s]	N/A

Table 3.4: Ranges of possible failover times to backup IP addresses for operating systems and browser/native-application pairings. Safari could only be tested for Apple devices, and so some entries have "N/A" for Safari. Behavior varies widely depending on the client's environment, suggesting that specifying multiple DNS answers in a response may not help failure resilience.

each browser, on the iPad application, on the iPhone application, on the Android application, and on the Desktop application. We tested all combinations that were possible, but did not test every combination since, for example, the Safari browser is only supported on Apple devices. We route all traffic through a Mac (*i.e.,* it acted as a WiFi AP), and used traffic filtering rules to block addresses/prefixes of our choice. To simulate failure, we set a rule on the Macbook to drop all traffic received from one or more of the IP addresses used by the application (*i.e.,* returned in a DNS response for that application).

We simulate failure at two different points of the experiment, one where the address is never responsive, and one where the address is responsive at first but then became unresponsive, and record how long it takes before a secondary IP address was used using `tcpdump`. We observe similar behavior in both sets of failure experiments and so do not report different results for each set of experiments.

We observe similar (although not identical) behavior regardless of the specific application/service, suggesting that behavior is mostly dictated by the operating system and browser. The high-level results, also summarized in Table 3.4, are that failover between addresses often occurs after more than a minute, but that behavior varies significantly depending on operating system, browser, and application. For example, iPhone applications switch to backup addresses in less than 100 ms (*i.e.,* on the order of a round-trip time), likely resulting in little performance impact to a user. Windows consistently takes tens of seconds to switch to a backup address. Failover times were generally worse on non-Apple devices except for Google Chrome, for which failover was nearly two minutes on Mac. Mobile devices tended to show better performance than non-mobile, except

for Google Chrome on Android.

Although we could not identify a root-cause for every behavior, blog posts and Chromium's open-source code base give us hints as to why failover times are so large. One key determination of why timeouts are so large is that they are dictated by default timeouts set in calls to the operating system. Usually these functions are of the form `connect`, to set up a TCP connection. As an interesting aside, clients generally do not respect TTLs when using backup addresses and often use records that are stale by the time they switch to a backup address even if the record was valid when the application tried to use the primary address.

Finally, there was no predictable pattern of which address clients would choose to use out of the multiple given in the DNS response. Hence, this multiple DNS response mechanism introduces an inherent tradeoff between controlling exactly where a client is directed and whether that client has a good backup option.

This behavior demonstrates that Service Providers cannot rely on giving the client multiple answers to protect against failover and that clients could be left without service for more than a minute (at which point, realistically, a user would give up). Moreover, Service Providers may be discouraged from using this functionality even with fast failover times, since Service Providers lack control of exactly where clients end up. Changing this behavior could require significant coordination among multiple parties and so seems unlikely to occur in the short term.

A Wider, Less-Precise View In addition to conducting tests on a few devices and for a few services, we also parse the campus traces (§3.7.1) to see if we can infer how services globally shift among IP addresses. These traces have the caveats that we can only see flow-level traces, rather than application behavior, and that we cannot determine with certainty if particular flows are suffering from performance problems.

To assess if clients in these traces switch among multiple DNS answers during failure or performance problems, we parse traces from September and October 2024 and look for flows with a relatively large number of retransmissions, as such flows are more likely to experience performance problems. To count the number of retransmissions for a flow, we count the number of

repeated TCP `SEQ, ACK` tuples seen during that flow's lifetime. We say flows with more than 30 retransmissions suffer from performance problems, where 30 was chosen since it is the 95^{th} percentile of the number of retransmissions across all flows for both months. We also save a subset of flows with fewer than 30 retransmissions as a control group.

We then map flows to flow groups, where a flow group consists of all flows from a given client to a destination where that destination is one of many IP addresses seen in a given DNS response for that client. For example, if a given DNS response to a client contained the IP addresses `1.2.3.4` and `5.6.7.8`, we would group flows from that client to those addresses into a flow group until a new DNS response for either address is observed (after which we would create a new flow group). For each flow group, we compute the fraction of addresses the client communicated with, the time between connection establishments for each of the various addresses (assuming the client opened many connections), and the distribution of traffic multiple over addresses in each flow group.

We compared these statistics between flows with and without a large number of retransmissions and found no significant differences between the distributions across flow groups for two months. We might expect, however, that flows with a large number of retranmissions would open up connections to more addresses, do so with less time between connection establishments, and send more traffic to more addresses than flows with a small number of retransmissions would. The lack of difference between these two groups may suggest that there is no widespread significant difference in behavior across many clients and services. These results are not definitive, however, since performance problems are rare, and our proxy signal for performance problems (many retransmits) is not perfect, and so it may be difficult to extract a reliable signal from a small number of flows with performance problems. We leave further investigation to future work.

Table 3.5: Statistics displaying the extent to which the recursives of users in Microsoft's CDN overlap recursives seen in the 2018 DITL captures without users and volumes by /24. Also shown in parentheses are corresponding statistics when joining by /24. Joining the datasets by /24 increases most measures of representation by tens of percents, with some measures increased by up to 64%.

dataset	Statistic	Percent Overlap (by /24)
DITL ∩ CDN	DITL Recursives	2.45% (29.3%) of DITL Recursives
	DITL Volume	8.4% (72.2%) of DITL Query Volume
	CDN Recursives	41.9% (78.8%) of CDN Recursives
	CDN Volume	47.05% (88.1%) of CDN Query Volume

3.8 Supplementary Analysis

3.8.1 Quantifying the Impact of Methodological Decisions

Representativeness of Daily Root Latency Analysis

In Section 3.3 we estimate the number of queries users experience due to the root DNS by amortizing queries over user populations. To obtain estimates of user populations, we obtain counts of Microsoft users who use recursives (§3.1.1). Naturally recursives used by Microsoft users and recursives seen in DITL do not overlap perfectly. To increase the representativeness of our analysis, we aggregate Microsoft user counts and DITL query volumes by resolver /24, and join the two datasets on /24 to create the DITL∩CDN dataset. The intuition behind this preprocessing step is that IP addresses in the same /24 are likely colocated, owned by the same organization, and act as recursives for similar sets of users. We now justify this decision and discuss the implications of this preprocessing step on the results presented in Section 3.3.3.

In Table 3.5 we summarize the extent to which the recursives seen by Microsoft are representative of the recursives seen in DITL, and vice-versa, without aggregating by /24. We also display corresponding statistics when aggregating by /24 for comparison in parentheses. Clearly joining by /24 makes a significant difference, increasing various measures of overlap by tens of percents and in certain cases by up to 64%.

As an analogy to Figure 3.4, in Figure 3.11 we show the number of queries each Microsoft user

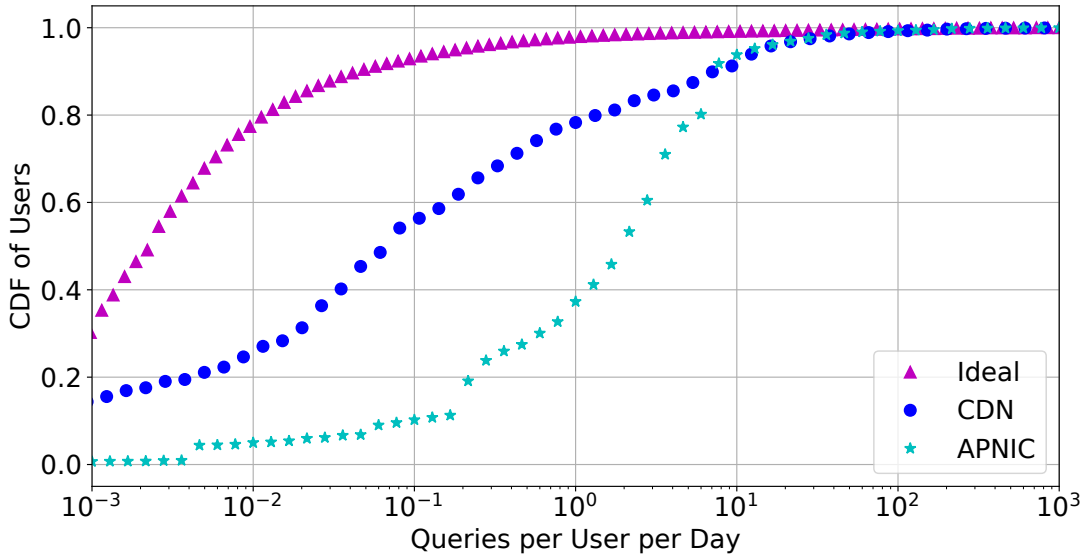


Figure 3.11: A CDF of the number of queries Microsoft users experience due to root DNS resolution, per day, without joining recursives by /24 in DITL with recursives seen by Microsoft (CDN). This unrepresentative analysis yields an estimate of daily user queries far, far lower than in Section 3.3.3.

executes to the roots per day *without* aggregating query and user statistics by /24 (CDN). We also show APNIC as in Figure 3.4 for comparison, even though APNIC is not affected by /24 volume aggregation. Users of CDN only send 0.036 queries to the roots each day at the median – roughly one 30^{th} of the estimate obtained when aggregating statistics by /24. This small daily user latency makes sense, given that we only capture 8.4% of DITL volume without joining the datasets by /24 (Table 3.5).

Table 3.5 and Figure 3.11, demonstrate that the decision to aggregate statistics and join DITL captures with Microsoft user counts by /24 led to both much greater representativeness of the analysis and very different conclusions about user interactions with the root DNS. We would now like to justify this decision using measurements. If, as we assume, IP addresses in the same /24 are colocated, they are probably routed similarly. Prior work has shown that only a small fraction of `anycast` paths are unstable [159], and so we expect that, over the course of DITL, IP addresses in the same /24 reach the same `anycast` sites.

As a way of quantifying routing similarity in a /24, in Figure 3.12 we show the percent of queries from each /24 in DITL that do not reach the most "popular" `anycast` site for each /24 in

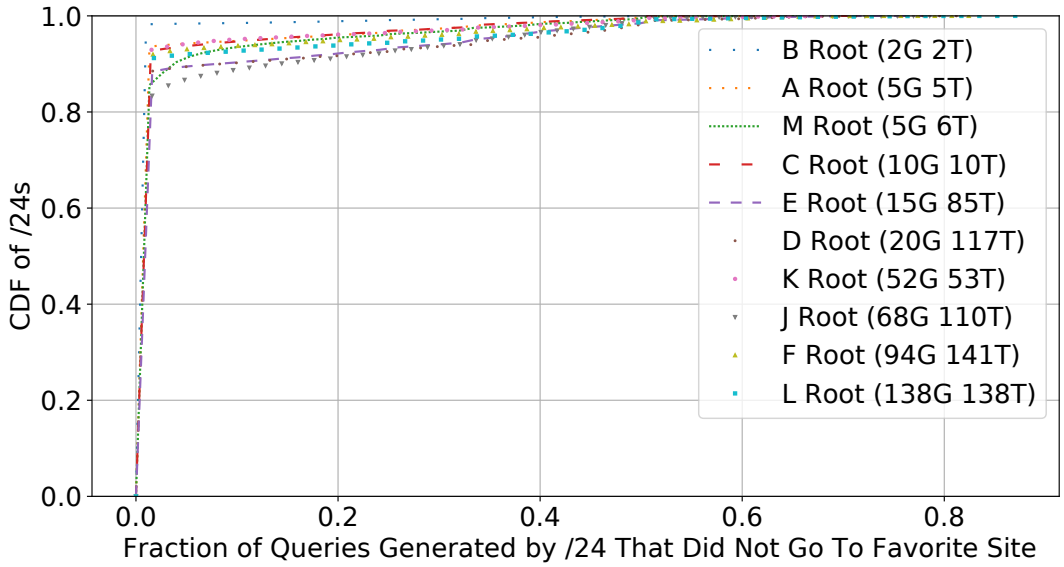


Figure 3.12: Fractions of queries generated by /24s that do not hit the most popular site for each /24 and for each root letter in question. The legend indicates the number of global sites (G) and total (global and local) sites (T). For all root letters, more than 80% of /24s have all queries visit the most popular site, suggesting queries from the same /24 are usually routed similarly.

each root deployment. We label root letters alongside the total number of sites (local and global) that they had during the 2018 DITL. For each root letter and for each /24 that queried that root letter in DITL, we look at how queries from the /24 are distributed among sites.

Let q_{ij}^k be the number of daily queries from IP i in /24 k toward `anycast` site j. We then calculate the fraction of queries that do not visit the most "popular" site as

$$f^k = 1 - \sum_i \frac{q_{ij_F}^k}{Q^k} \tag{3.4}$$

where j_F is the favorite site for /24 k (*i.e.,* the site the /24 queries the most), and Q^k is the total number of queries from /24 k. We plot these fractions for all /24s in DITL, and for each root deployment. (We do not include /24s that had only one IP from the /24 visit the root letter in question.)

For more than 80% of /24s, all queries visit only one site per root letter, suggesting that queries from the same /24 are routed similarly. This analysis is slightly biased by the size of the root deployment. For example, two IP addresses selected at random querying B root would hit the

same site half the time, on average. However, even for L root, with 138 sites, more than 90% of /24s direct all queries to the most popular site. We believe Figure 3.12 provides evidence that recursives within the same /24 prefix are located near each other, and hence serve similar sets of users.

Even queries from a single IP address within a /24 may reach multiple sites for a single root over the course of the DITL captures. Such instability can make routing look less coherent across IP addresses in a /24, even if they are all routed the same way. Controlling for cases of changing paths for the same IP makes intra-/24 routing even more coherent. If we let the distribution of queries generated by an IP address to a root be a point mass, with all the queries concentrated at that IP addresses' favorite site, all queries from more than 90% of all /24s to all roots are routed to the same site (not shown).

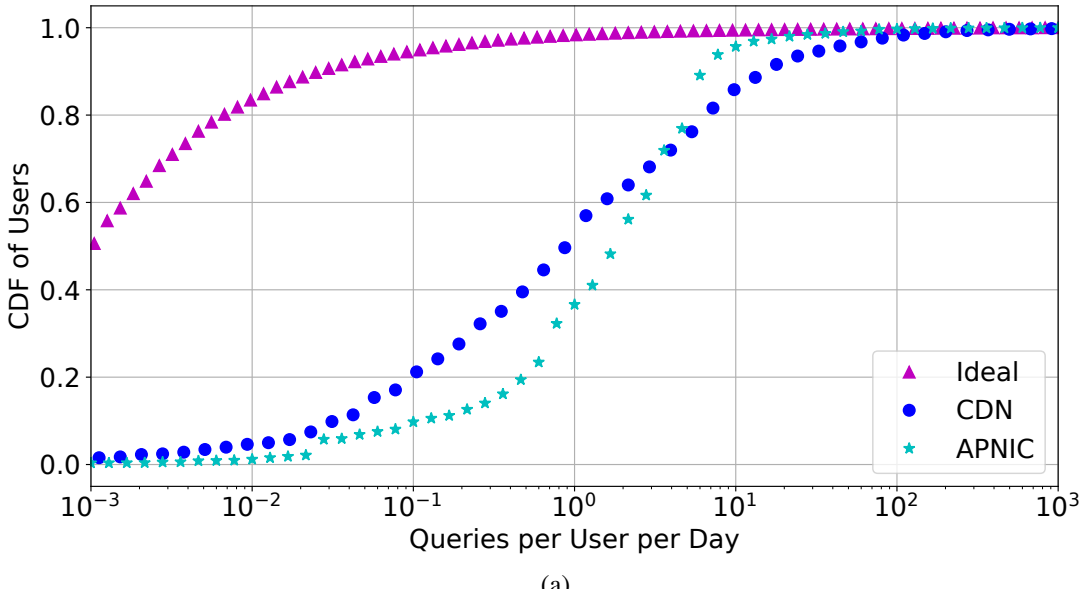
Implications of Using the 2018 DITL

At the time of writing, the 2020 DITL was available to use in the study, but we chose to use the 2018 study since the 2018 study had better coverage of root letters. (Neither has perfect coverage – for both 2018 and 2020 DITLs, G root is not included and I root is completely anonymized.)

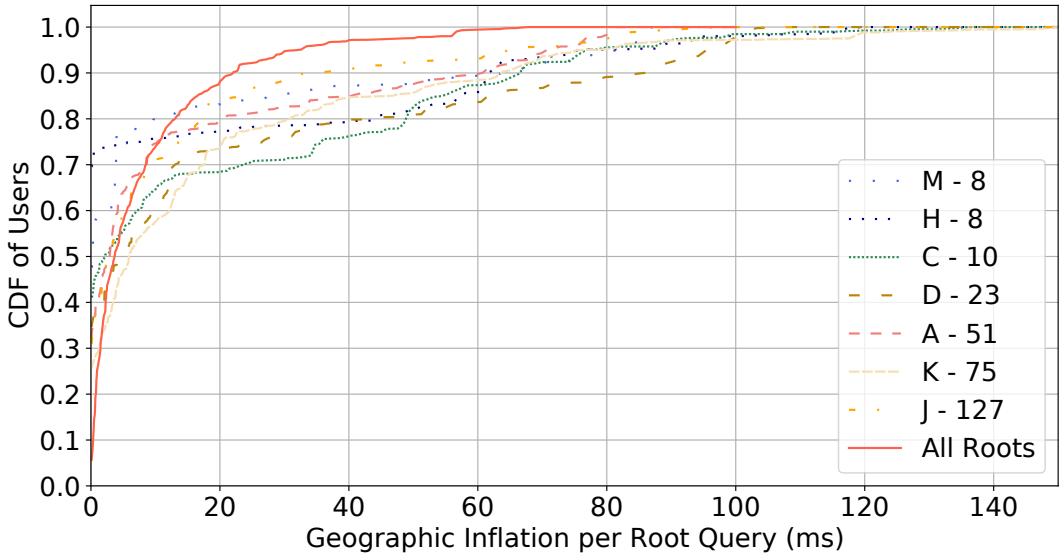
For the 2020 DITL specifically, B root was not available at the time of writing (but may be in the future), E root includes only one site (out of 132), F root does not include any Cloudflare sites (more than half the volume), and L root is completely anonymized (hence unusable). The 2018 DITL has none of these limitations, and so our results apply to more letters. Studying the root DNS system as a whole is a key strength of our analysis compared to prior work, so we feel coverage is more important than having the most up-to-date results for only a subset of root letters.

For completeness, and to demonstrate that our larger takeaways about root DNS latency and inflation do not change significantly from year to year, we calculate queries per day (as in Figure 3.4) and inflation (as in Figure 3.3) for the root letters for which we have data, and the results are shown in Figure 3.13.

Our high level conclusions about root DNS latency do not change when looking at the 2020



(a)



(b)

Figure 3.13: Queries per user per day to the root DNS and inflation of root letters calculated using the 2020 DITL. Our high level conclusions about how much inflation is in the root DNS and the number of queries users experience per day do not change depending on the year.

DITL – most users still experience about one DNS query per day, and the number of root queries sent by recursives is still far from the 'ideal' querying behavior of one record per TTL. Inflation results are also similar – individual root letters have less inflation (for example, D root improved). Average geographic inflation is almost exactly the same as in 2018, with approximately 10% of

users experiencing more than 20 ms (2,000 km) of inflation.

3.8.2 Latency Measurements at a Recursive Resolver

To obtain a local perspective of how users experience root DNS latency, we use packet traces from ISI. Here, we characterize DNS and root DNS latencies users experience at the resolver, along with a useful visualization of how inconsequential root DNS latency is for users at this resolver. This analysis complements our global view of how users interact with the root DNS in Section 3.3.3, as it demonstrates how often everyday users might send queries to the root relative to other DNS queries.

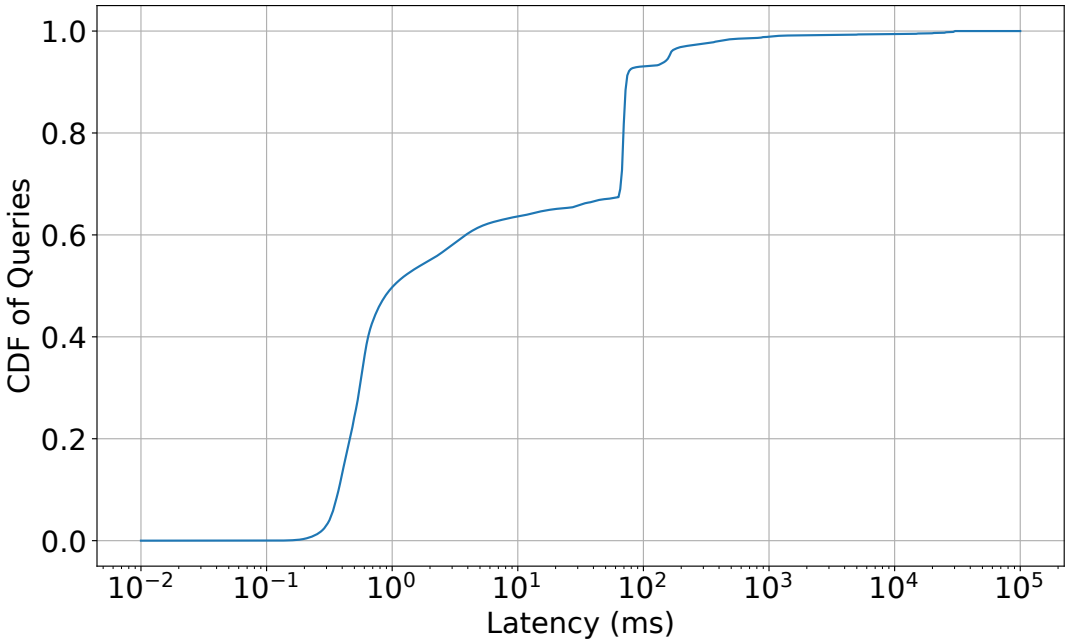


Figure 3.14: CDF of user DNS query latencies seen at a recursive resolve at ISI, over the course of one year. Latencies are measured from the timestamp when the recursive resolver receives a client query to the timestamp when the recursive sends a response to that client query. The sub-millisecond latency for more than half of queries suggests most queries to this recursive are served by the local cache.

Figure 3.14 shows the latencies of all queries seen at the recursive resolver over one year, where latencies are measured from the timestamp when the recursive resolver receives a client query to the timestamp when the recursive sends a response to that client query. Latencies are divided into (roughly) 3 regions: sub-millisecond latency, low latency (millisecond - tens of milliseconds),

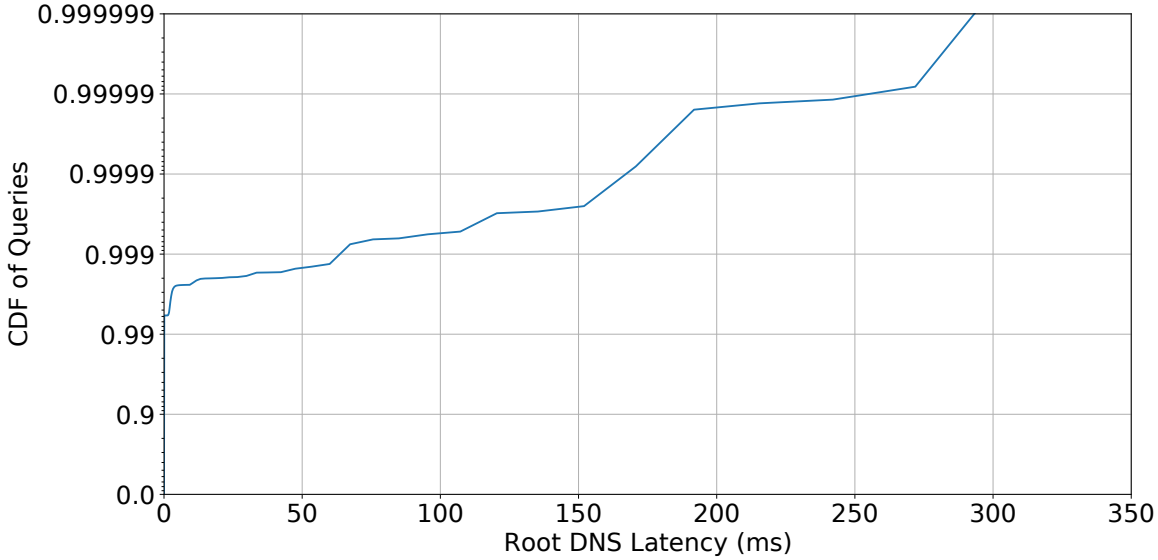


Figure 3.15: Root DNS latency for queries made by users of ISI recursive resolver during 2018. This plot demonstrates the benefits of caching and high TTLs of TLD records – fewer than 1% of queries generate a root request, and fewer than 0.1% incur latencies greater than 100 ms. User queries that did not generate a query to a root server were given a latency of 0.

and high latency (hundreds of milliseconds). The first region corresponds to cached queries, so roughly half of queries are (probably) cached. The second region corresponds to DNS resolutions for which the resolving server was geographically close. Finally, the third region likely corresponds to queries that had to travel to distant servers, or required a few rounds of recursion to fully resolve the domain. The sub-millisecond latency for more than half of queries suggests most queries to this recursive are served by the local cache. These latencies are similar to those presented in previous work that also studied a recursive resolver serving a small user population [87]. Queries in the second and third regions include queries that did not query the root (since those records were cached) but did query other parts of the DNS hierarchy.

As discussed in Section 3.3, root DNS queries make up a small fraction of all queries shown in Figure 3.14. To visualize just how small this fraction is, Figure 3.15 shows a CDF of root DNS latency experienced for queries over 2018. Requests that do not generate a query to a root server are counted as having a root latency of 0. Figure 3.15 demonstrates the benefits of shared caches and high TTLs of TLD records – fewer than 1% of queries generate a root request, and fewer than

Table 3.6: Redundant root DNS requests. The last five requests to J root are redundant which may be caused by an unanswered request in step 4.

Step	Relative Timestamp (second)	From	To	Query name	Query type	Response
1	0.00000	client	resolver	bidder.criteo.com	A	
2	0.01589	resolver	192.42.93.30 (g.gtld)	bidder.criteo.com	A	
3	0.02366	192.42.93.30 (g.gtld)	resolver	bidder.criteo.com	A	ns23.criteo.com ns22.criteo.com ns25.criteo.com ns26.criteo.com ns27.criteo.com ns28.criteo.com.
4	0.02387	resolver	74.119.119.1 (ns25.criteo.com)	bidder.criteo.com	A	
5	0.82473	resolver	182.161.73.4 (ns28.criteo.com)	bidder.criteo.com	A	
6	0.82555	resolver	192.58.128.30 (j.root)	ns22.criteo.com	AAAA	
7	0.82563	resolver	192.58.128.30 (j.root)	ns23.criteo.com	AAAA	
8	0.82577	resolver	192.58.128.30 (j.root)	ns27.criteo.com	AAAA	
9	0.82584	resolver	192.58.128.30 (j.root)	ns25.criteo.com	AAAA	
10	0.82592	resolver	192.58.128.30 (j.root)	ns26.criteo.com	AAAA	
11	0.82620	resolver	192.58.128.30 (j.root)	ns28.criteo.com	AAAA	

0.1% incur latencies greater than 100 ms.

3.8.3 Case Study: Redundant Root DNS Queries

When we investigate the traffic from a recursive resolver to the root servers in Section 3.3, we see as many as 900 queries to the root server in a day for the COM NS record. Given the 2 day TTL of this record, this query frequency is unexpectedly large. This large frequency motivated us to analyze why these requests to roots occurred. We consider a request to the root to be redundant if a query for the same record occurred less than 1 TTL ago. Prior work has investigated redundant requests to root servers as well, and our analysis can be considered complementary since we discover different reasons for redundant requests [122].

To observe these redundant requests in a controlled environment, we deploy a BIND instance (the resolver in Section 3.1.1 runs BIND v9.11.17) locally and enable cache and recursion. We do not actually look up the cache of the local BIND instance to see which records are in it. Instead, we save the TTL of the record and the timestamp at which we receive the record to know if the record should be in BIND's cache. We use BIND version 9.11.18 and 9.16.1. Because 9.16.1 is one of the newest releases and 9.11.18 is a release from several years ago, we can assume that pathological behavior is common in all versions between these two releases. After deploying the instance, we simulate user behavior by opening the top-1000 web pages according to GTmetrix [160] using Selenium and headless Chrome. While loading web pages, we collect network packets

on port 53 using Tshark [144].

For these page loads, we observe 69,215 DNS A & AAAA-type requests generated by the recursive resolver. 3,137 of these requests are sent to root servers, and 2,950 of these root DNS queries are redundant. Over 70% of redundant requests are AAAA-type. After investigating the cause of these redundant queries, we find over 90% of these redundant requests follow a similar pattern. This pattern is illustrated by the example in Table 3.6.

In Table 3.6, we show queries the recursive resolver makes when a user queries for the A record of bidder.criteo.com. In step 1, the recursive resolver receives a DNS query from a client. According to TTL heuristics, the COM A record is in the cache. In step 3, the TLD server responds with records of authoritative nameservers for "criteo.com". Then, the recursive chooses one of them to issue the following request to. However, for some reason (*e.g.,* packet loss), the recursive resolver does not get a response from the nameserver in step 4. Hence, the resolver uses another nameserver in step 5, which it learned in step 3. At the same time, as seen in step 6 to 11, the recursive sends (redundant) DNS requests to root servers, querying the AAAA-type records for these nameservers. These requests are redundant since the AAAA record for COM was received less than two days ago.

From the pattern demonstrated in Table 3.6, we hypothesize that redundant requests to the root servers will be generated for certain records when the following conditions are met.

1. A query from the recursive resolver to an authoritative nameserver times-out.

2. The record queried for by the resolver to the root DNS server was not included in the Additional Records section of the TLD's response.

The second condition is also why we were seeing more AAAA-type redundant requests, because usually there are more A-type records in the Additional Records section than AAAA-type records.

To see how much traffic is caused by our hypothesis in a real scenario, we analyze packet captures on a recursive resolver (BIND 9.11.17) serving users at ISI. To keep consistent with

the other analysis we do on this dataset (§3.3), we use packet captures from 2018. 79.8% of requests to roots are redundant and in the pattern we described. The other 20.2% consists of necessary requests and requests for which we have no hypothesis as to how they were generated. We contacted developers at BIND, who said this may be a bug.

Software behavior as described here can lead to orders of magnitude more root DNS requests than would be necessary if recursives queried for the record once per TTL. As demonstrated in Figure 3.4, focusing on reducing the number of these queries could both improve user experience and reduce load on the root server.

3.8.4 Visualization of Microsoft CDN Performance

3.9 Summary

While `anycast` performance is interesting in its own right, prior studies have drawn conclusions primarily from `anycast` for root DNS. We have shown that `anycast` operates differently in CDNs, with less inflation. Differences stem from the impact the `anycast` service's latency and inflation has on user-perceived latency. Our results show the importance of considering multiple subjects in measurement studies and suggest why `anycast` continues to see wide, growing deployment. However, `anycast` inevitably leads to some inflation which can hurt performance for some users, and fixing this inflation may require constant effort from operators (for example, updating peering strategies or BGP announcements). `Unicast`, on the other hand, gives Service Providers more control over where users go, but may introduce reliability problems due to caching. These two methods of directing traffic are sufficient for many use cases, but in the remainder of this dissertation we explore whether that performance will suffice for emerging Internet use cases.

Chapter 4: Improving Steady State Interdomain Routing with Path Exposure and Selection

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 [161]. 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 [162]. 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 [163].

For example, investigation of a 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` site 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 site through another ISP (Transit ISP), there was no mechanism for detecting such paths and re-directing customer traffic (see Figure 4.1). Hence Figure 4.1 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

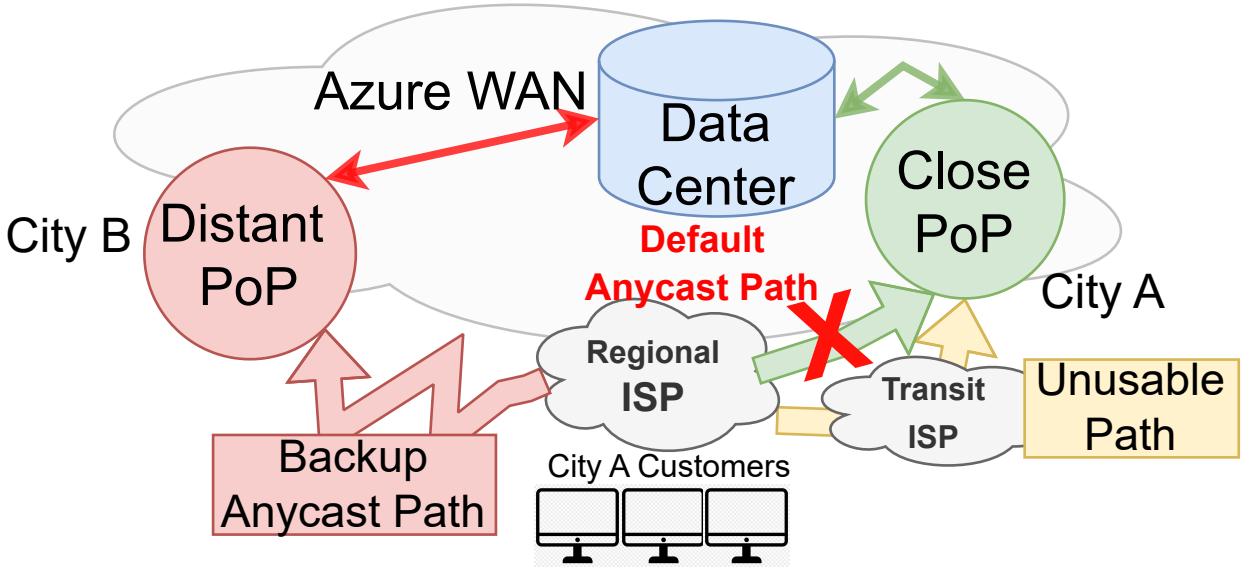


Figure 4.1: A difficult routing problem for an enterprise cloud customer.

availability and performance. Clouds use `anycast` for availability, but may sometimes use `unicast` for performance since `anycast` can inflate paths [29, 12] (chapter 3). `Unicast` routing is enabled by DNS which directs users to different sites, but at coarse granularities (per-recursive resolver), cannot select among different paths to a single site, and cannot react quickly during failure due to clients/recursives not respecting DNS TTLs [33, 34, 157, 158] (chapter 3). All these limitations of DNS hurt availability. BGP limits clouds further since it may pick poorly performing paths, it only exposes one path, and failover between paths is slow [164].

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 [165], cloud-edge network stacks [166, 167, 168, 169], mobile apps with full stack control [170, 171], and integrated VPNs such as Apple Private Relay [172, 173]. 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.

Here 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 [166, 167, 168, 169]. We found this type of edge presence to be the most deployable, most powerful solution for clouds like Azure (more details in Section 4.4.2). However, PAINTER could use other edge presences such as MPTCP-enabled clients [174] (§4.2.2).

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 [72], and build a prototype using the PEERING testbed [38], which (at the time of writing) was deployed at 25 Vultr cloud locations [73]. 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 (§4.4.1), 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 than SD-WAN with multihoming (§4.4.2).

We show PAINTER's steering mechanism is far more deployable than other solutions (§4.4.2) and that this mechanism steers traffic at finer (per-flow) granularities than other steering mechanisms (DNS/BGP updates) [32, 103]. 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 (§4.4.2). Our prototype on the PEERING testbed [38] demonstrates a helpful use case of PAINTER that these other approaches fail to address—failover at RTT-timescales (§4.4.2).

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 can be computed, installed, and selected directly in single-domain settings, the interdomain

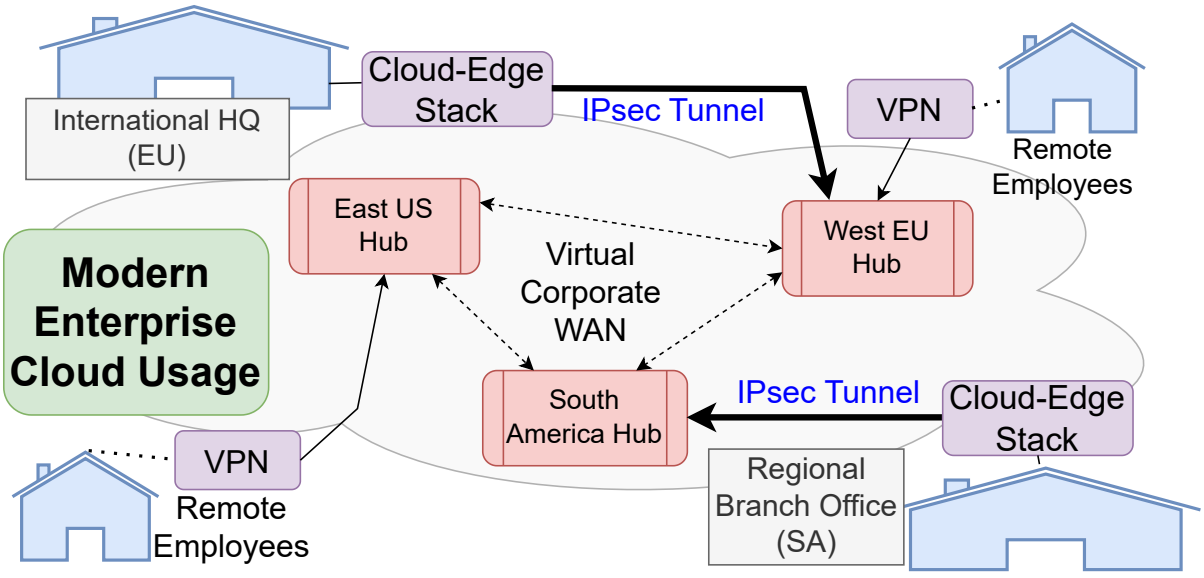


Figure 4.2: Modern enterprise integration with the cloud.

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 [163, 1]. 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.

4.1 Motivation and Challenges

4.1.1 Modern Enterprises, Old Protocols

Even though paths to the cloud are often low latency [1, 175] and near-optimal [31, 29, 12], occasional networking problems still affect critical enterprise operations (*e.g.,* Fig. 4.1). 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 4.2 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 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 [176, 177] 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 [163, 1]. For example, US mobile providers actively collaborate with clouds to provide 5G network functions [178, 179, 180, 181, 182]. 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 4.1.2). Finally, IP addresses are mapped to Internet paths via BGP which is not performance-aware and is slow to reconverge after failure [24]. These factors combine to form a mapping process that is coarse, slow to react, and often not performant.

4.1.2 Insufficiencies of Existing Techniques

IP Anycast and/or DNS

IP anycast is an approach to routing where distinct sites all advertise the same IP prefixes. This strategy is used by Service Providers for its simplicity and resilience [29, 43, 79, 80, 81, 82].

`Anycast` offers limited control over paths, leading to path inflation or unpredictable mappings from clients to sites (as evidenced in our aforementioned customer issues) [29, 27, 12, 42], 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 site [8, 183, 32]. However, we demonstrated in Section 3.7 that `unicast` introduces reliability problems due to client caching. 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 [99, 100, 101, 102], which may use a combination of overlay routing (with sites/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 [184] which, by itself, does not solve our problem and has been around for decades [185]. We compare our proposal to an `SD-WAN` multihoming solution in Section 4.4.2. 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 [186] 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.

4.1.3 Opportunity: Cloud Control at the Edge

Current solutions to directing traffic suffer from a lack of control (§4.1.2), 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 [187, 188], cloud-edge network stacks [166, 167, 168, 169], recent OS versions that give application developers more control over the network stack (*e.g.,* MPTCP/MPQUIC, the Skype app) [170, 171, 189, 190, 191, 192, 193, 105, 165], and integrated VPNs such as Apple Private Relay [172, 173]. 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 [166, 167, 168, 169], and since both parties (enterprise and cloud) would benefit from this synergy.

4.1.4 Key Challenges

Realizing a solution that overcomes the limitations of current protocols (§4.1.1) 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) [194] 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 [17, 195, 27, 18, 16, 191, 105, 171], 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 [196]) and their impact on global BGP routing tables.

BGP routing tables are growing for both v4 and v6 address spaces [197], for which the only solutions are to reject advertisements (bad) or to buy expensive routers (also bad) [198]. The importance of this problem is evidenced by the large body of research on compressing routing tables [199, 200, 201, 202]. Moreover, other work proposes advertising multiple prefixes to enhance performance [96, 98, 35, 6] 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 [203].

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 4.2.1 for more details).

4.2 System Description

PAINTER (Precise, Agile INgress Traffic Engineering & Routing), summarized in Figure 4.3, consists of the Advertisement Orchestrator (§4.2.1) and the Traffic Manager (§4.2.2). 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 4.3, see Section 4.2.1 for more details) and the Traffic Manager tunnels traffic along best paths at fine traffic granularities (§4.2.2). Azure still advertises the anycast prefix (1.1.1.0/24 in Figure 4.3) since anycast offers low latency for most users [29, 12].

We designed PAINTER to be a service run by Azure, with nodes in edge proxies that we call TM-Edges and nodes in Azure sites 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 (§4.1.3), but we believe cloud-edge network stacks are the most sensible proxy technology for our setting (§4.2.2). TM-PoP can be integrated with Azure front-ends in sites. 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.

4.2.1 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 4.1.4, we have to find a good *subset* of paths to expose. To

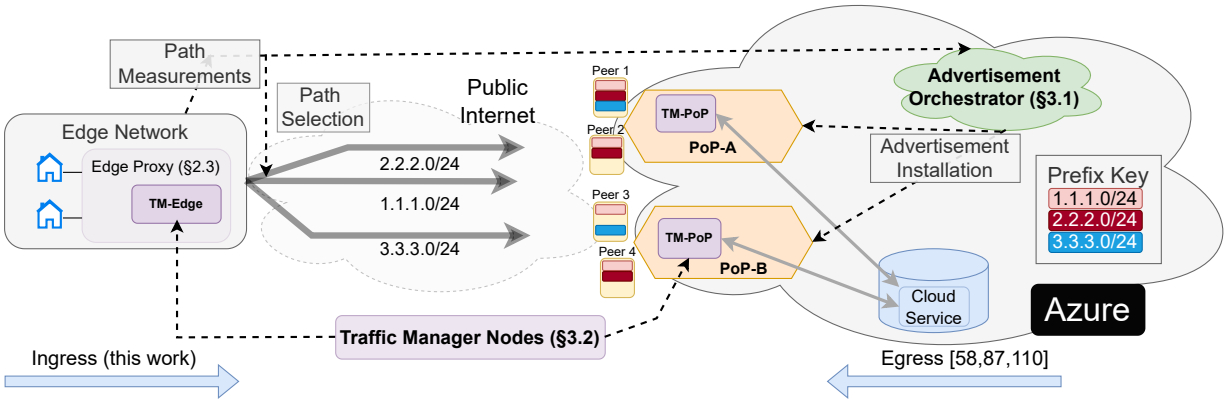


Figure 4.3: Overview of PAINTER. TM-Edges in edge proxies measure paths and tunnel traffic to TM-PoPs. TM-PoPs relay traffic destined to many prefixes to appropriate cloud services. The Advertisement Orchestrator uses path measurements to decide how to update PAINTER prefix advertisements. Prefix advertisements are represented by colored boxes inside peers at sites.

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 (§4.4.2). 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 [20]). Over time the Advertisement Orchestrator learns from these instances of poor routing, computing strategies that get more benefit with fewer prefixes with each iteration.

Unicast vs Anycast

Anycast commonly refers to announcing the same prefix via all sites and all peerings while unicast often refers to announcing the same prefix via *one* site via all peerings at that site

[29, 24]. The Advertisement Orchestrator, however, may advertise a single prefix from multiple sites via a small subset of peerings at each site. Therefore the Advertisement Orchestrator 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 [35].

Maximizing Benefit

We model an advertisement configuration A as a set of ⟨peering, prefix⟩ pairs where ⟨peering, prefix⟩ $\in A$ means we advertise that prefix via that peering. More complex advertisement configurations (*e.g.,* BGP community tagging) are out of the scope of this dissertation. 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

$$B(A; D) = \sum_{\text{UG}} w(\text{UG}) \cdot I(A, \text{UG}; D) \tag{4.1}$$

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 UG*s* with fine-grained control and includes anycast as an option, all UG*s* 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 (4.1).

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 UG*s* to Azure *ingresses*, and correct incorrect heuristics over time. TM-Edges (§4.2.2) 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 UG*s*. 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 [204] 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 [205]. 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 UG*s* 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 UG*s* to each policy-compliant ingress individually (there are many examples in the literature [74, 35, 206, 31]). Hence, to predict UG latency we predict the *ingress*. We detail our measurement methodology in Section 4.4.1.

Since it is difficult to predict ingresses [35, 42], 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. (4.2)).

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 site and ISP B at a Miami site 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 (§4.4.1). Preference models of routing are used in prior work [35].

Second, we exclude ingresses that would result in a UG reaching an Azure site more than D_{reuse} km (reuse distance) from the closest site advertising that prefix, as prior work found large path inflation is rare [29, 12]. D_{reuse} is a configurable parameter. For example, given that we advertise a prefix at a Central US site and a Tokyo site, we assume a UG near Eastern US (1,500 km to Central US, 11,200 km to Tokyo) would reach the Central US site 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 Section 4.4.1.

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

$$\tilde{I}(A, \texttt{UG}; D) = \max_{P \in \mathscr{P}} \left(\min_{P' \in \mathscr{P}} \mathbb{E}_D(l(\texttt{UG}, P'))\right) - \mathbb{E}_A(l(\texttt{UG}, P)) \tag{4.2}$$

where $\mathscr{P}$ is the set of prefixes being advertised, and $\mathbb{E}_A(l(\texttt{UG}, P)$ 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 (4.2) approximates true improvement in Equation (4.1) due to this expectation.

Solving For Good Advertisement Configurations

Summarizing, we wish to maximize Equation (4.1), which we model using Equation (4.2). Maximizing Equation (4.1) 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 sites, and notify the Traffic Manager about available prefixes via a control channel.

Algorithm 1 takes two hyperparameters: the minimum reuse distance, D_{reuse}, which implicitly affects improvement calculations (Eq. (4.2)), 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

Algorithm 1 Algorithm for selecting advertisements.

Input Prefix Budget PB, minimum reuse distance D_{reuse}

```
RM ← []                                                          ▷ Routing model—how users route to deployment
while learning do                                                ▷ Terminate learning when little marginal benefit increase
    CC ← []                                                      ▷ 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 ← calc_improvements(CC, RM)    ▷ Equation 2
            ranked_peerings ← sort(peering_improvements)         ▷ Rank
            found_peering ← False                                ▷ Can we find a peering?
            for next_best_peering in ranked_peerings do          ▷ Greedy search
                NP ← (p, next_best_peering)                      ▷ Proposed new prefix,peering
                if B(NP; CC) > 0 then                            ▷ Require positive benefit.
                    found_peering ← 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 ← execute_advertisement(CC)                               ▷ Advertise CC and update routing model
end while
return CC
```

improvements relative to the current configuration (Eq. (4.2)). We add a ⟨`peering, prefix`⟩ pair to the current configuration if the advertisement provides positive benefit (Eq. (4.1)). 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. (4.1)). 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 (§4.3).

4.2.2 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.

Traffic Manager Tunneling Mechanism

The Traffic Manager uses a tunneling mechanism similar to some used by SD-WAN solutions [207] since it requires no endpoint modification.

We show a typical packet's journey through PAINTER using our tunneling mechanism in Figure 4.4. 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) (§4.2.2). 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

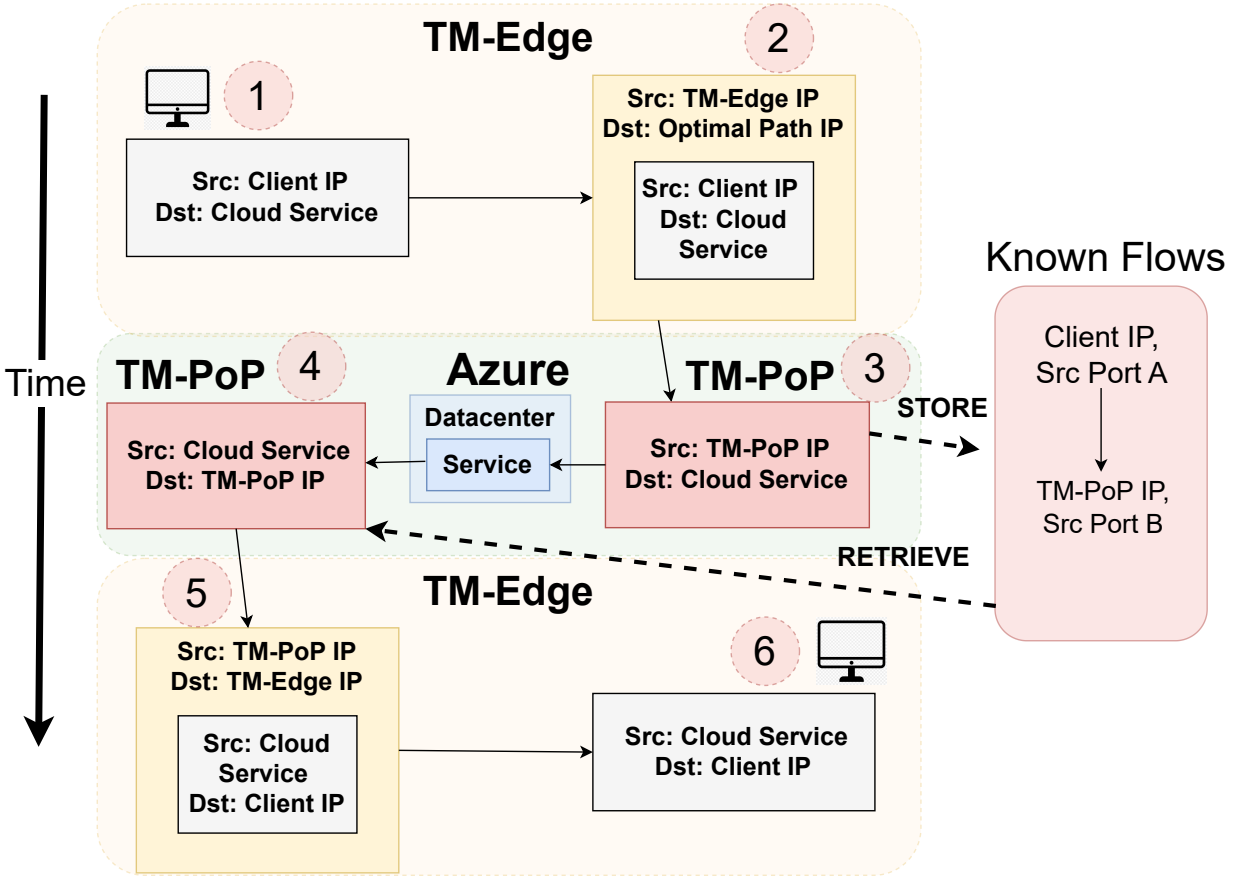


Figure 4.4: Packet journey through PAINTER. TM-Edge tunnels client traffic along different ingress paths (1-2), TM-PoP NATs client traffic (3-4), and sends the traffic back to TM-Edge who forwards it to the client (5-6).

the 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 sites since there need only be one TM-PoPs per site.

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 iden-

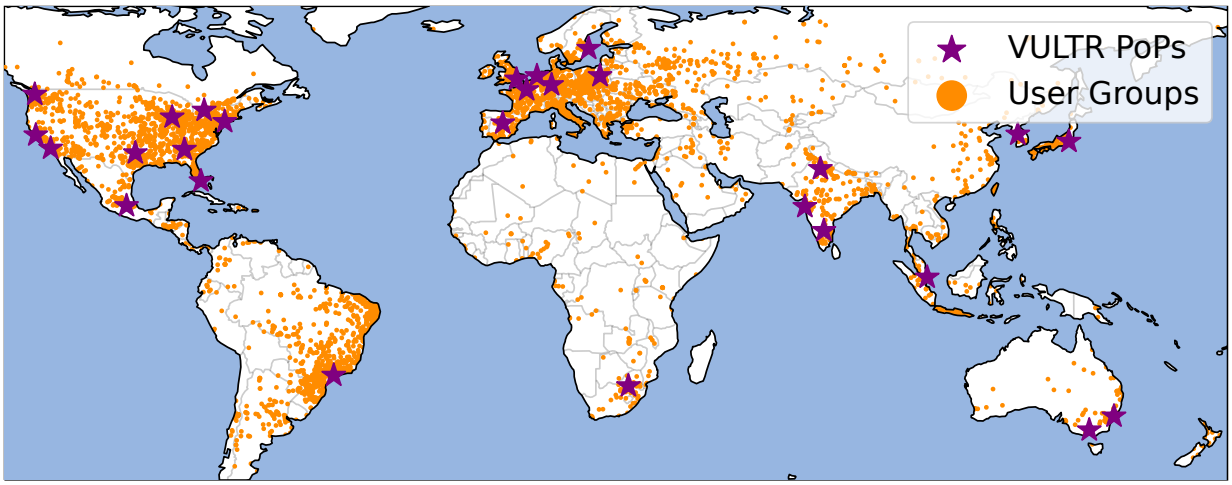


Figure 4.5: PAINTER prototype deployed globally at Vultr's sites, and UG*s* we probed.

tifies the TM-PoP it communicates with along that tunnel. Hence, each TM-Edge maintains a mapping of destination prefixes to sites (which is difficult to compute apriori, as prefixes may be advertised via multiple peerings at multiple sites). Although a TM-Edge may have paths to every TM-PoP, available sites may vary depending on the service since each service may only be served from certain sites 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 [208].

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.,* site), 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 [209, 210].

4.2.3 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 [9, 8, 10]), 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 [104]).

4.3 PAINTER Implementation

4.3.1 Advertisement Orchestrator

The Advertisement Orchestrator takes measurements from TM-Edges and hyper-parameters 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 (§4.4.1).

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 4.4.1 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 [38], which is deployed at Vultr cloud locations [73]. 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 used 25 Vultr sites on 6 continents with 5,000 neighbor ASes and 9,000 ingresses (Fig. 4.5), offering us a rich platform for testing advertisement strategies.

We also implement a partial Advertisement Orchestrator prototype on Azure. We

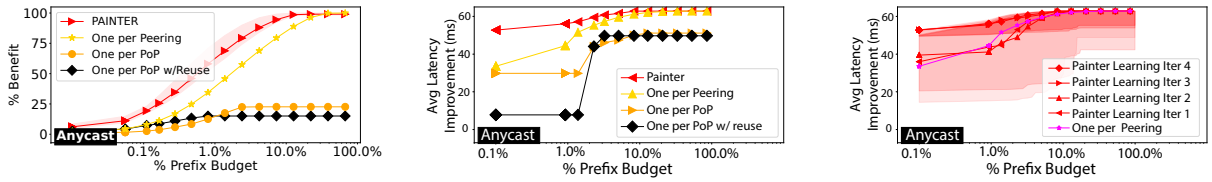
could not change BGP announcements from Azure for operational reasons, nor could we conduct measurements to all UG*s*. Instead, we use a combination of real and simulated measurements to evaluate the Advertisement Orchestrator on Azure— we detail this methodology in Section 4.4.1. Azure is a global cloud with 200 data centers interconnected by 175,000 miles of lit fiber whose traffic is managed by a software-defined WAN [2]. Traffic enters and leaves Azure's WAN through roughly 200 sites which are often in major metropolitan areas [2]. Azure's WAN connects sites to data centers. sites also have peering routers which connect Azure to more than 4,000 networks [13] Some networks connect at multiple sites, most only at one [7].

4.3.2 Traffic Manager

The Traffic Manager steers traffic between TM-Edges in edge proxies and TM-PoPs at Azure sites. We prototyped the Traffic Manager on cloud VMs using a lightly modified version of FlexiWAN [211] 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 Section 4.2.2.

4.4 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* (§4.4.1, Fig. 4.6b). We also show that the Traffic Manager fails over from a unicast path to a backup at RTT timescales using our prototype (§4.4.2, Fig. 4.13).



(a) (b) (c)

Figure 4.6: PAINTER achieves more benefit with less budget compared to other advertisement strategies shown using simulated measurements from Azure (4.6a) and from an actual global cloud (4.6b). PAINTER learns from incorrect assumptions over iterations (4.6c). Shaded regions show PAINTER's uncertainty prior to testing a strategy.

4.4.1 Advertisement Orchestrator

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 (§4.3) 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 UG*s* would experience to Azure ingresses and computed a range of latencies UG*s* could experience to each prefix. To estimate the latency UG*s* would experience through an ingress to a prefix, we measure latency from UG*s* 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 site. We verify IP address locations using speed of light constraints from RIPE Atlas probes [72] with known locations.

We use RIPE Atlas probes to measure latencies to Azure peerings. Considering paths from all UG*s* 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 Section 4.4.1 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 (§4.2.1). We group measurements from RIPE Atlas probes in the same UG, yielding measurements from 4k UG*s*. Since RIPE Atlas covers a relatively small number of UG*s* (only 47% of Azure traffic volume), we then simulate measurements using a methodology that extrapolates RIPE Atlas ingress latencies to nearby UG*s* 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 UG*s*.

We simulate measurements from UG*s* for which there is no available RIPE Atlas probe. We consider the set of UG*s* that represent 99% of Azure traffic volume, as this significantly reduces computational complexity (*i.e.,* the number of UG*s* we need to consider).

We first tabulate the set of all policy-compliant ingresses from UG*s* 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 [74].

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, 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 UG*s* with "good" routing while areas with bad routing will induce simulated measurements with bad routing in those areas. Simulated measurements are from hundreds of thousands of UG*s* to thousands of ingresses.

PEERING Measurements. For our PEERING prototype (§4.3), we measured client latencies by pinging clients from the deployment as in prior work [35, 206]. We do not need to estimate latency here since we can conduct actual advertisements with our prototype. We measure from PEERING to 40k UG*s*. From our measurements, latency gains are heavily concentrated among its ingresses—we only saw latency improvement for approximately 8k UG*s* through 250 out of 9,000 ingresses which were mostly transit providers. We show the global scale of both our deployment and UG*s* with which we evaluated PAINTER in Figure 4.5.

Ingress Latency Estimation

Azure connects to other organizations at over 200 sites around the world [2]. 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

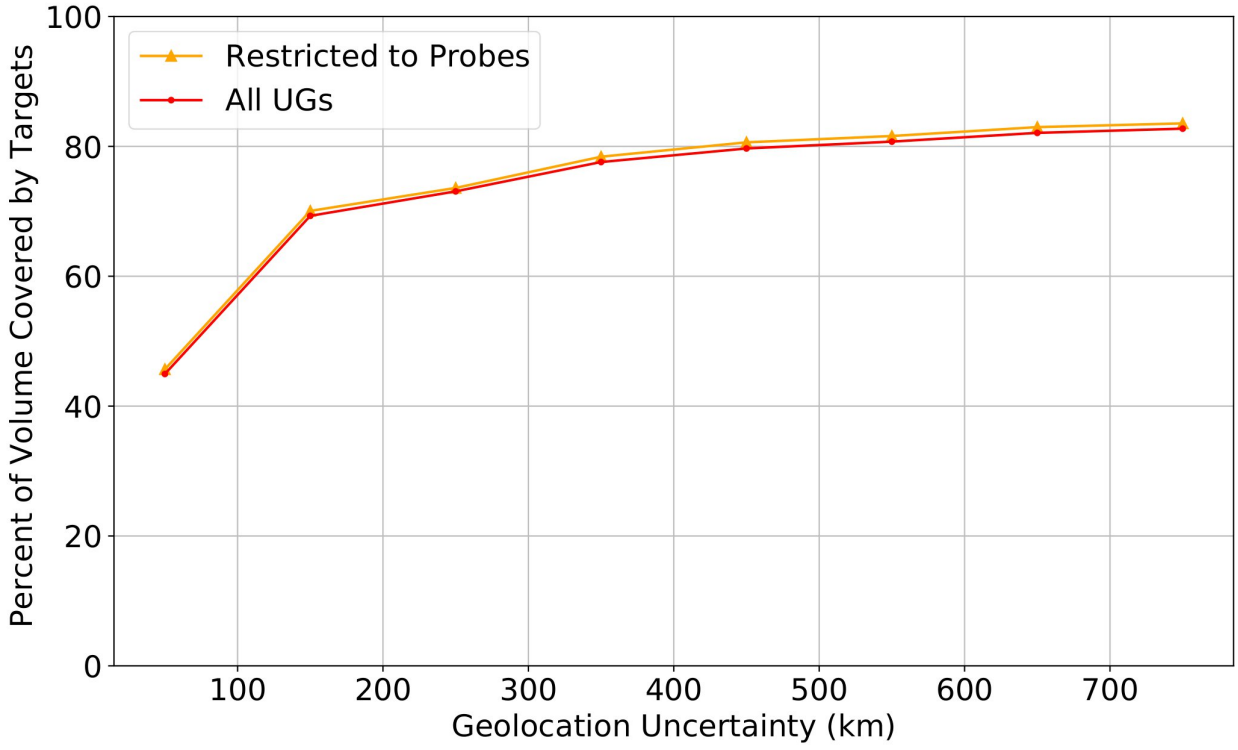
these addresses at all sites (hence the route is the same as the anycast route). In this way, we were only able to obtain target addresses for fewer than 500 ingresses as many addresses were unresponsive.

To find targets for the remaining (vast majority of) ingresses, we obtained lists of candidate IP addresses by (a) crawling RIPE Open IPMap [212] and Maxmind [133], (b) parsing Azure traceroutes from clients, and (c) using RDNS hints (specifically using Hoiho [213]). We exclude addresses known to be anycast according to a publicly available list [214]. 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 site. 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.

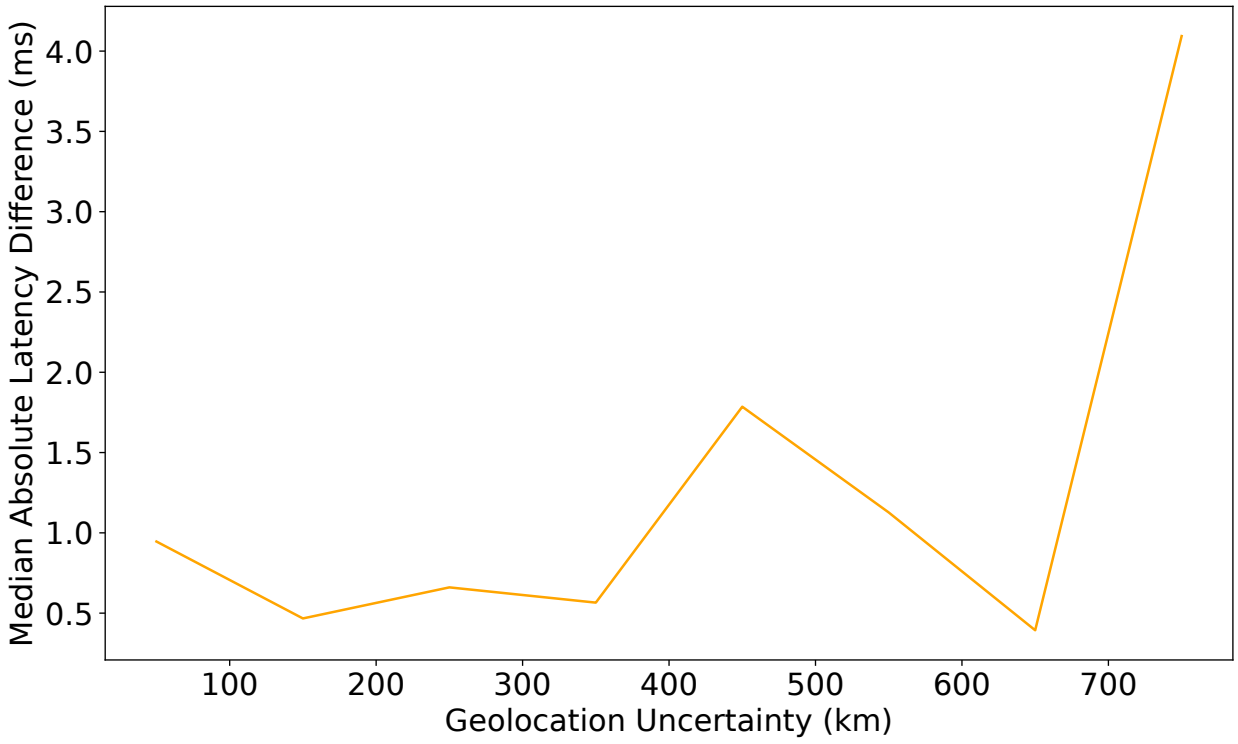
We were only able to find a subset of ingress targets at a given geolocation uncertainty due to the limitations (*i.e.,* coverage) of our geolocation methodology. To quantify how representative ingress targets were for a given geolocation uncertainty, we tabulated all ⟨UG, ingress⟩ tuples for which a path from the UG through the ingress could be policy-compliant (§4.2.1) 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 site is unlikely to provide latency benefits if the anycast latency from that UG is lower than the distance from the UG to the site converted to the speed of light in fiber (*i.e.,* the best possible latency to the site). For example, we do not count policy-compliant ingresses at Azure's Chicago site 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



(a)



(b)

Figure 4.7: Coverage of policy-compliant ingresses at various geolocation uncertainties (4.7a) and median absolute difference in estimated and actual latency (4.7b).

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 4.7b plot the median difference for each bucket.

Results. In Figure 4.7a we show our coverage of ingresses (weighted according to the above metric) at different maximum geolocation uncertainties. In Figure 4.7b we show how ingress target latencies compared to actual latencies through those ingresses at different geolocation geolocation uncertainties.

Figure 4.7a 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 UG*s* are similar, likely since RIPE Atlas probes tend to be in UG*s* that generate lots of Azure traffic volume.

Figure 4.7b shows the median absolute difference between our targets and the actual latency to Azure. Figure 4.7b 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.

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 site its own prefix that it announces to all peerings (One per Site) since prior work explored using per-site announcements to lower latency [29, 35]. 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 site, but allows prefix reuse when sites are more than D_{reuse} km apart (One per Site 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 UG*s* have a route to their best ingress.

For each strategy, we compute a range of possible latency benefits since UG*s* 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 sites 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 UG*s* all the latency benefit since it would expose all the paths.

Results. Figure 4.6a shows the *estimated* benefit each strategy attains as a percent of the total possible benefit (Eq. (4.1)) 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 Section 4.4.1. We show benefit as a percent of the total possible for anonymity. For example, 50% benefit corresponds to UG*s* achieving half the total possible latency decrease over anycast, on average.

The One per Site w/Reuse and One per Site strategies do not consider advertising different prefixes to different peerings at a single site, so both strategies fail to uncover the routes necessary to offer as good estimated benefits as PAINTER. In practice, this means UG*s* have several policy-compliant ingress options at each site 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 4.6b plots average latency improvement over clients that have non-zero improvement for our prototype on PEERING (§4.3), 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 4.6c 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).

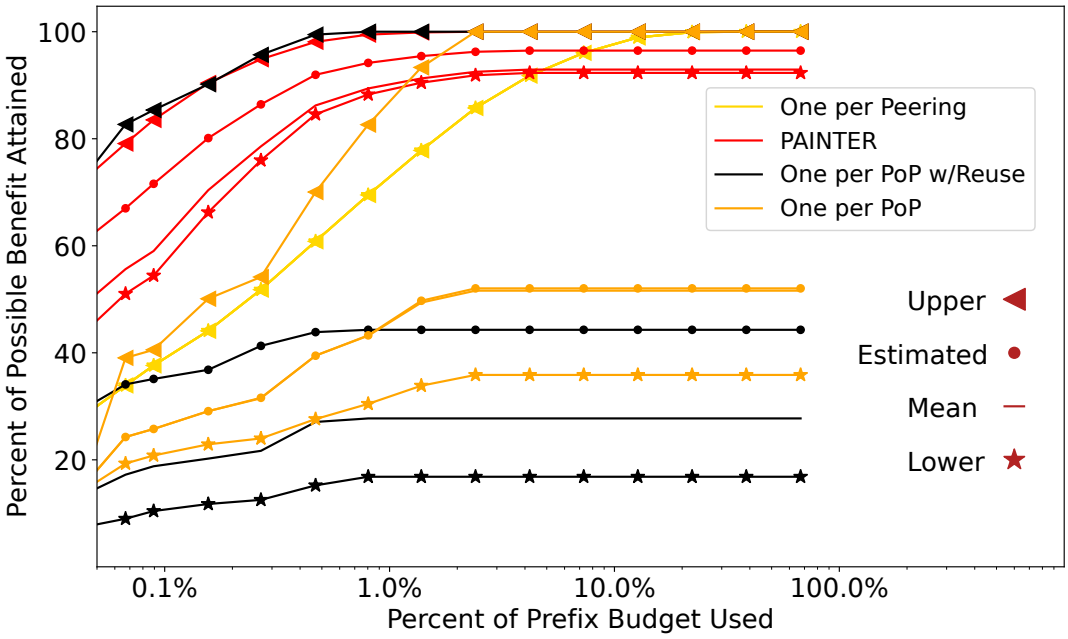
Our prototype outperforms all other strategies, even using only one prefix since, with other strategies, too many UG*s* get a *bad* route to some sub-optimal ingress, with little or no benefit over anycast. PAINTER identifies which subsets of routes offer improvement and only advertises those, refining 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 [203]. 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 [215] advertise at least 500 /24 prefixes according to Routeviews [216] and Figure 4.6a 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 Site (with or without reuse).

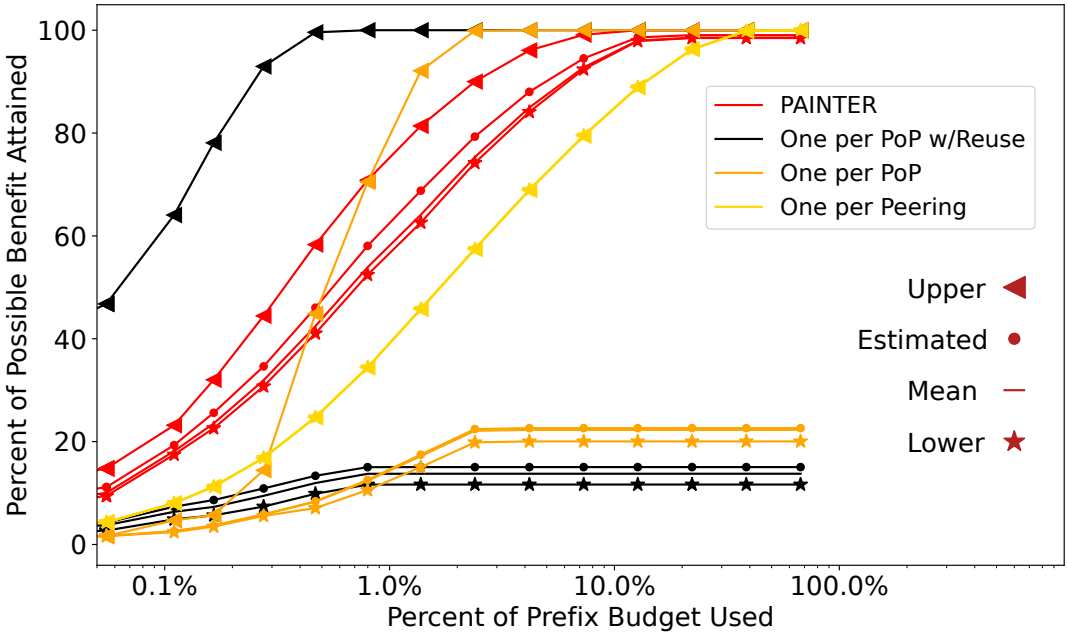
Benefit Ranges over Budget

In Section 4.4.1 we showed *estimated* latency benefits for each advertisement strategy. In Figure 4.8, 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 (§4.4.1). 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 UG*s* selects. UG*s* select the highest 'Mean' prefix over all their prefix choices (Eq. (4.2)).

One per Site strategies have very large ranges of possible benefits since they advertise prefixes via all peerings at sites, and so expose many (possibly poor) valid ingresses for UG*s*. 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



(a)



(b)

Figure 4.8: Range of latency benefits for each strategy over prefix budget, computed using real (4.8a) and simulated (4.8b) measurements.

UG*s* may be routed to worse ingresses. In practice, larger ranges mean that some UG*s* will be routed optimally while some will not, as was observed in prior work [29].

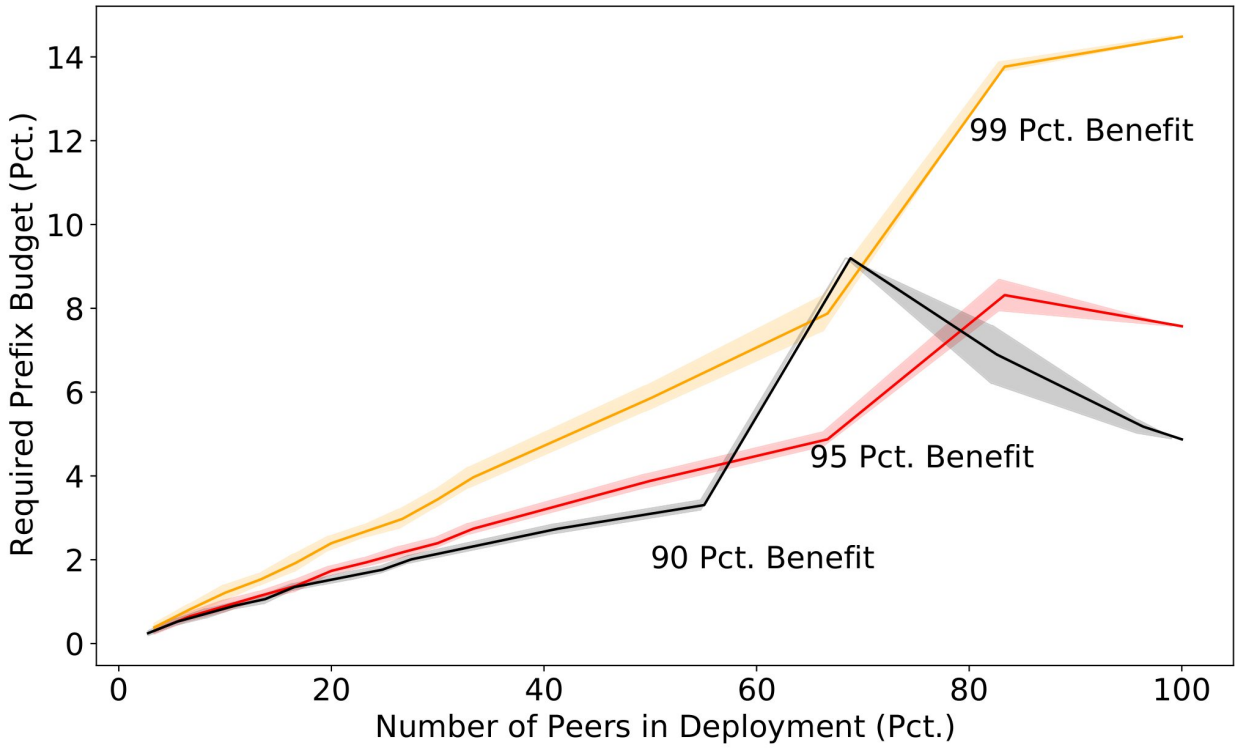
PAINTER's intelligent prefix reuse from far-away sites and via peerings with non-overlapping customer cones allows it to quickly achieve most latency benefit with little uncertainty. The One per Peering strategy has no uncertainty since it advertises a unique prefix via each peering, but it uses at least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

Scale and Uncertainty

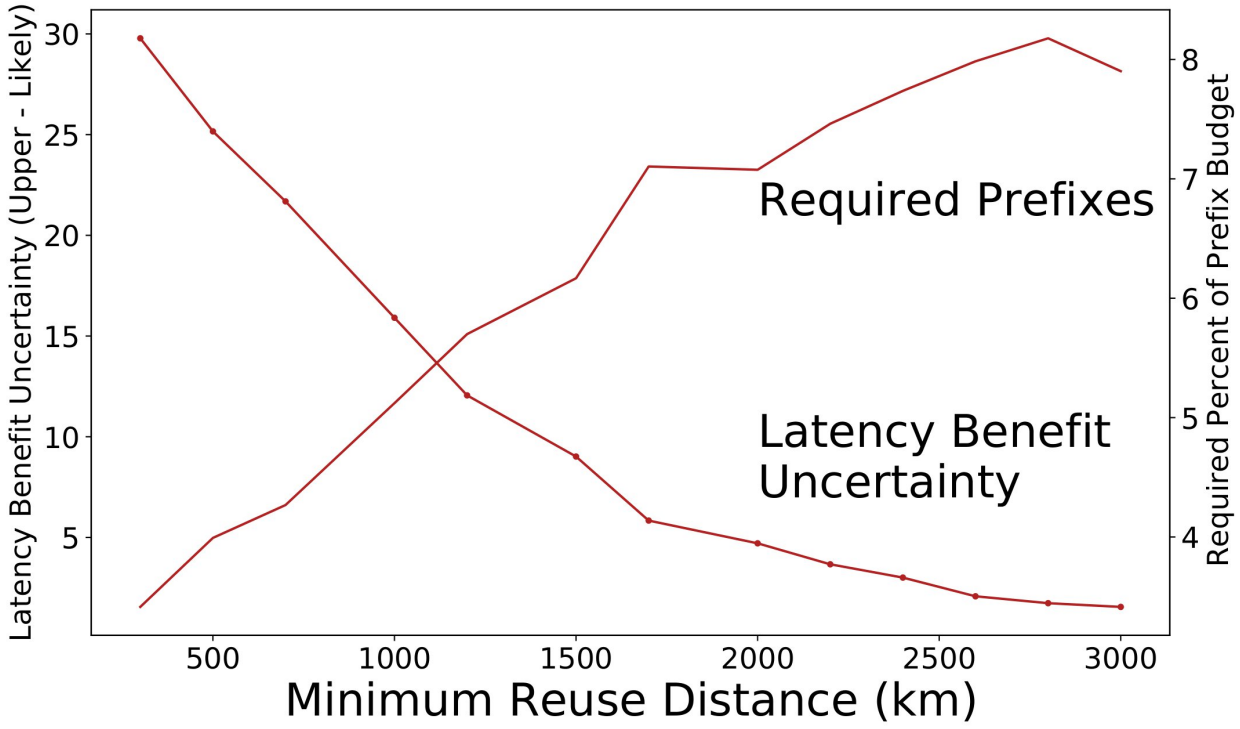
Figure 4.9 summarizes the Advertisement Orchestrator convergence properties with respect to two parameters: deployment size and the minimum reuse distance (D_{reuse}). Figure 4.9a shows that the required number of prefixes to obtain various percent benefits scales linearly with deployment size, so we 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 [7], unlike prior work [96, 35].

Figure 4.9b 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 4.6a 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 4.6a as it provides a decent tradeoff between uncertainty and cost. This is in contrast to other solutions (*e.g.,* One per Site) which provide no way to control benefit uncertainty (and have quite high uncertainty).



(a)



(b)

Figure 4.9: PAINTER scales linearly in the number of required prefixes with deployment size (4.9a), and naturally provides a tradeoff between benefit uncertainty and prefix cost via a tunable parameter (D_{reuse}) (4.9b).

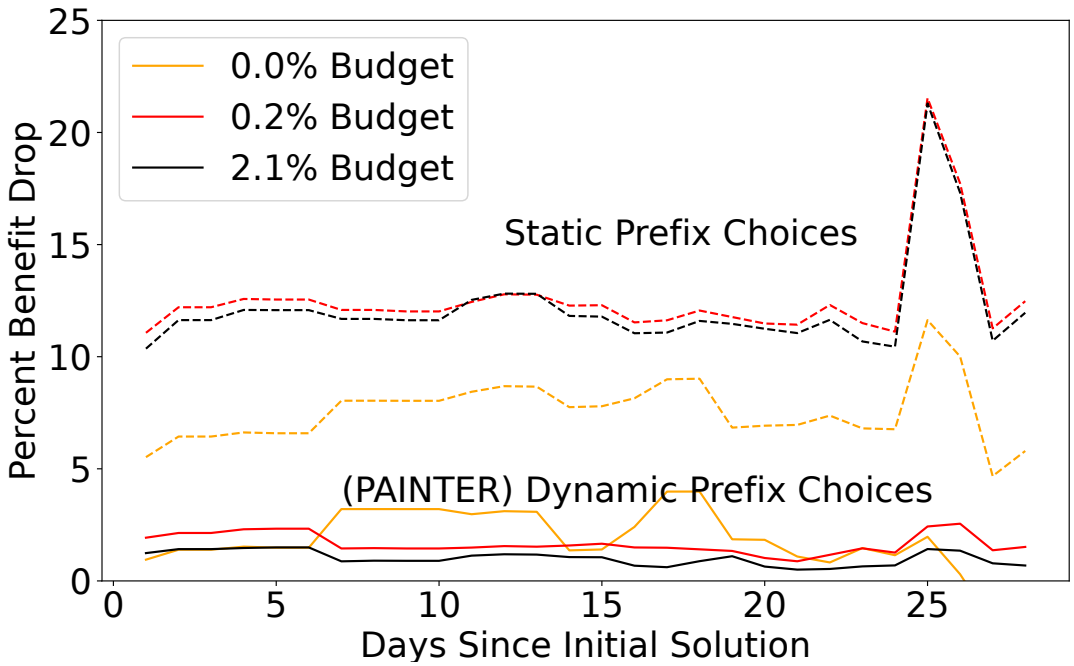


Figure 4.10: PAINTER advertisement benefits remain for at least a month whether users change their prefix choice over time (Dynamic Prefix Choices) or they do not (Static Prefix Choices).

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 4.10 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 [217], 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 UG*s* to different prefixes dynamically. Benefit loss when UG*s* do not switch prefixes over time (shown by the dashed lines) is approximately 10% worse (*i.e.,* UG*s* 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 UG*s*, 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.

4.4.2 Traffic Manager

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 4.11. A solution is more deployable if it can direct more traffic with less deployment effort. A solution is more precise if it can control finer traffic quantities at finer time granularities and direct them over more paths to clouds. Figure 4.11 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 [193, 171]. Deploying the Traffic Manager (specifically TM-Edge) into applications could similarly face deployability obstacles since the framework may have to be deployed and main-

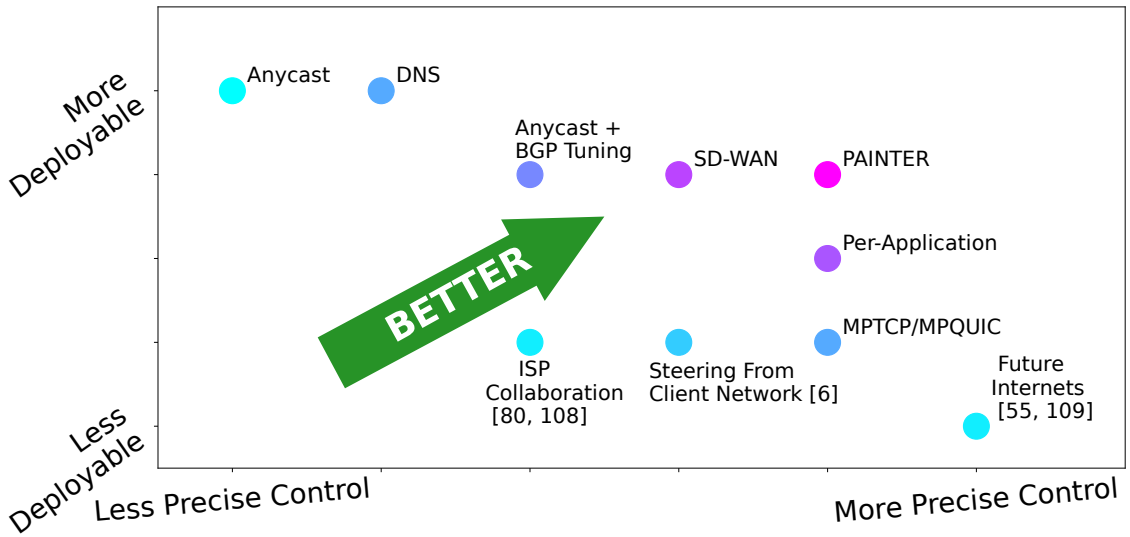


Figure 4.11: Situating PAINTER partially on cloud-edge network stacks gives clouds precise, deployable traffic control.

tained independently for each application. Enterprise networks that use SD-WAN devices with multihoming to optimize routing have fewer path options than PAINTER (§4.4.2), making them less precise. Section 4.5 discusses related systems that involve ISP Collaboration [18, 17], future Internets [16, 195], and steering solely from the client network [218], but they generally offer precise traffic control at the cost of unrealistic/unscalable assumptions (sacrificing deployability).

Fine-Grained Traffic Control

Figure 4.12 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 sites. To obtain an optimistic bound on the granularity at which BGP can control traffic (with-

out 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. 4.6a) (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 [71]. 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 4.12a we show the granularity at which each solution affects different volumes of traffic overall (column 'All') and for the top 10 sites (site-X) by volume. Each site 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 site A, 64% of ingress traffic comes from (⟨peering, user AS⟩) pairs responsible for between 10% and 100% of all traffic arriving at site 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 site 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 sites—for example, DNS controls 100% traffic arriving at site A at granularities finer than 1%, whereas DNS only controls 43% of traffic at this granularity at site B. PAINTER could control all traffic at the finest granularity, since PAINTER controls individual flows.

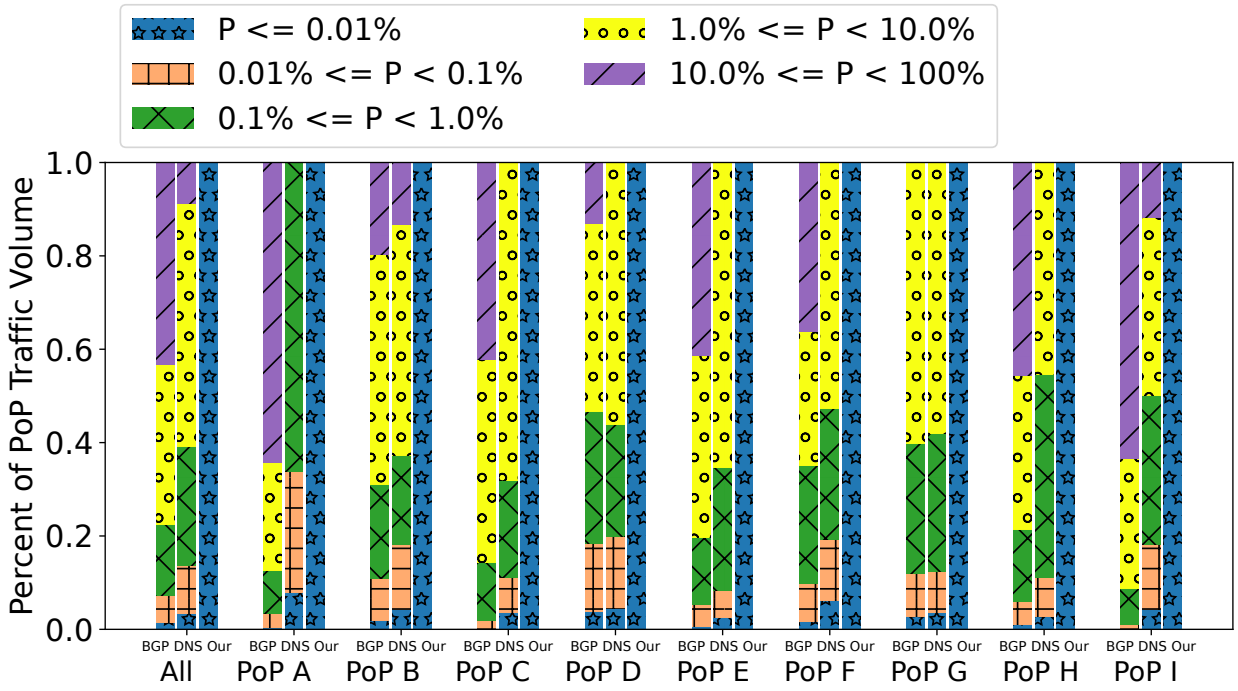
Figure 4.12b quantifies one drawback of coarse control—inability to fully benefit from the Advertisement Orchestrator advertisement strategies. Using DNS sacrifices roughly half the benefit as is possible with fine-grained redirection, since some DNS resolvers serve diverse UG*s* for which no single 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.

Quick, Agile Reactions

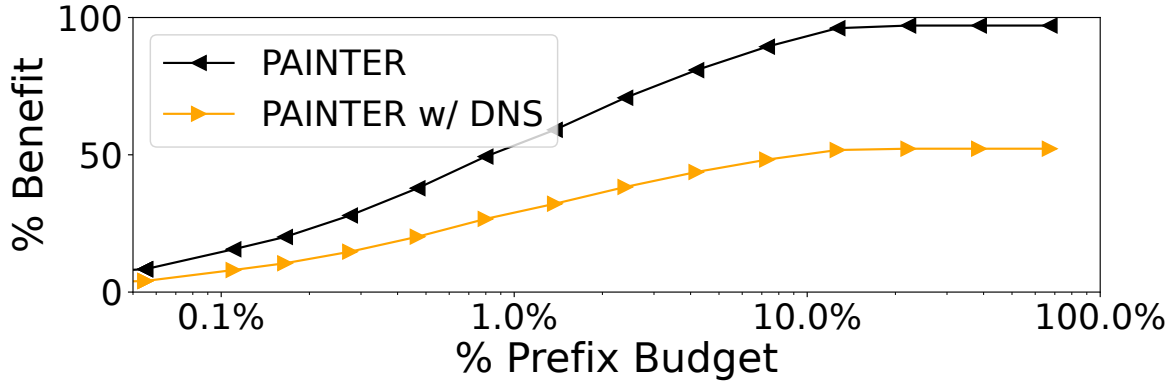
Methodology. Prior work proposed announcing unicast prefixes and directing users to them via DNS to improve latency [29, 41]. Using DNS to improve latency raises *availability* concerns since DNS reaction times are slow (§4.1.2). We show PAINTER realizes the latency improvements of unicast while retaining availability.

We advertise an anycast prefix (1.1.1.0/24) at two sites and one prefix to each ISP at those two sites. Figure 4.13a 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 site-A (2.2.2.0/24) since the path to it is lower latency than the default anycast path. At 60 seconds, we withdraw all prefixes at site-A, which is meant to model a failure in the site advertising PAINTER's chosen prefix.

Results. The time series graph in Figure 4.13b illustrates typical system operation during

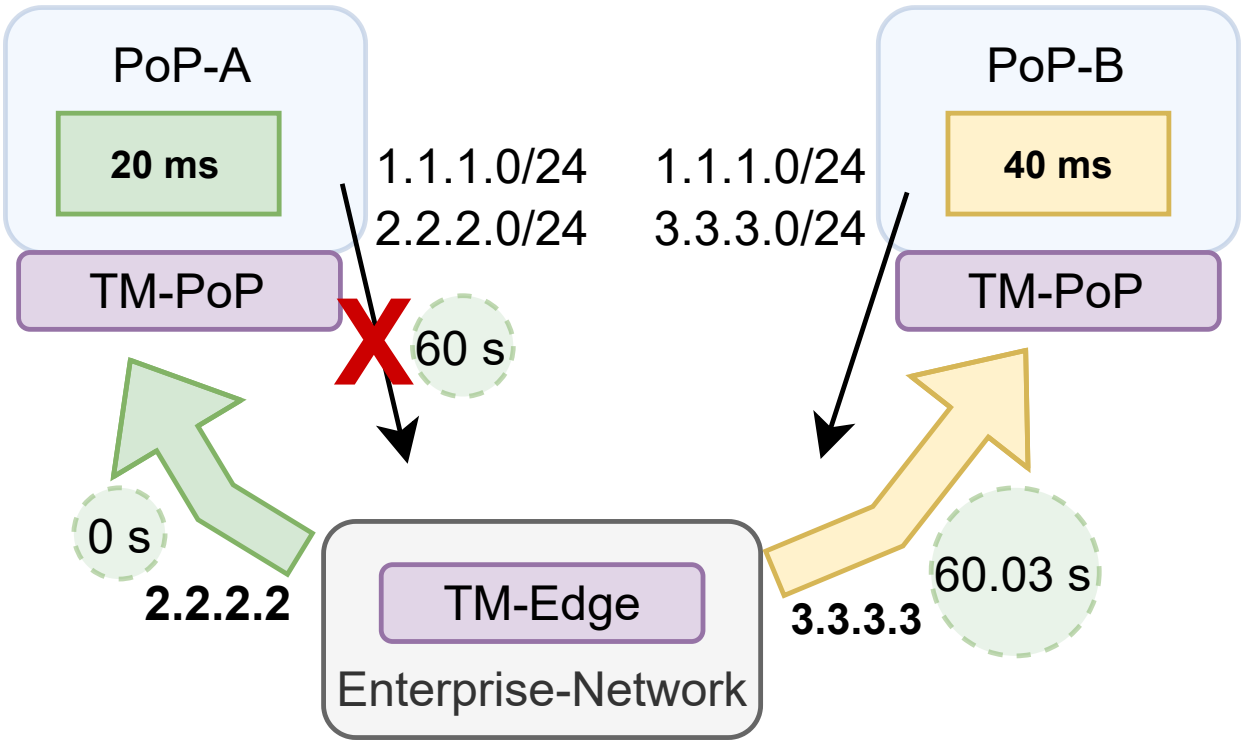


(a)

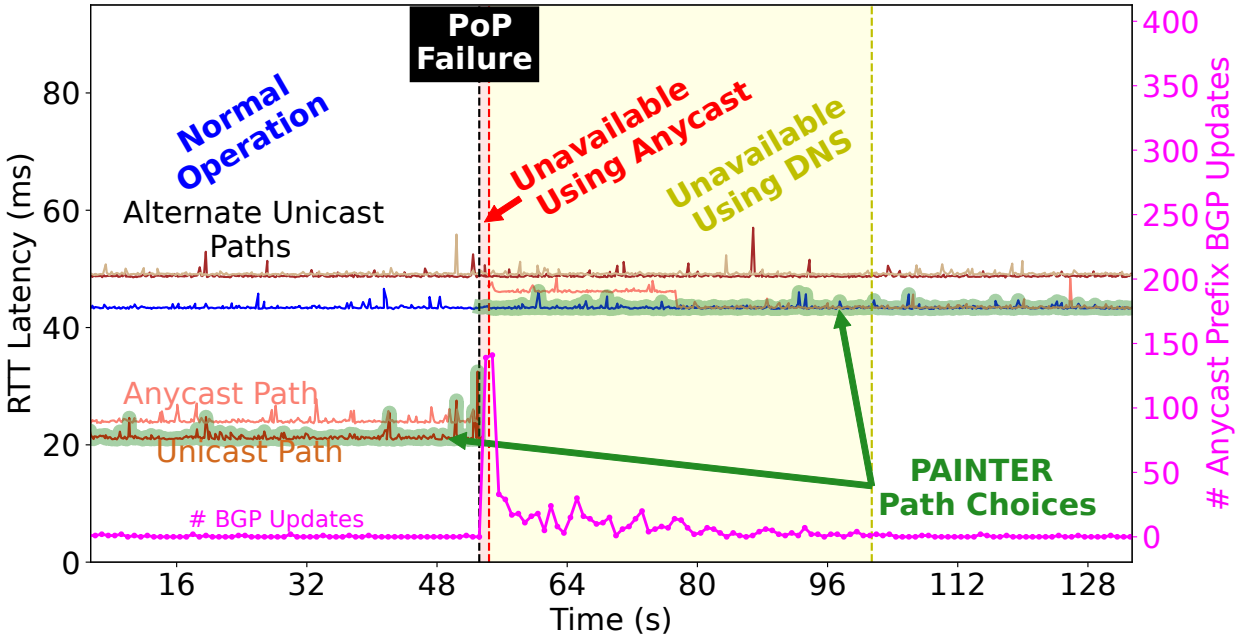


(b)

Figure 4.12: Alternate approaches to steering ingress traffic have coarse control (Fig. 4.12a) which limits the advertisement effectiveness (Fig. 4.12b).



(a)



(b)

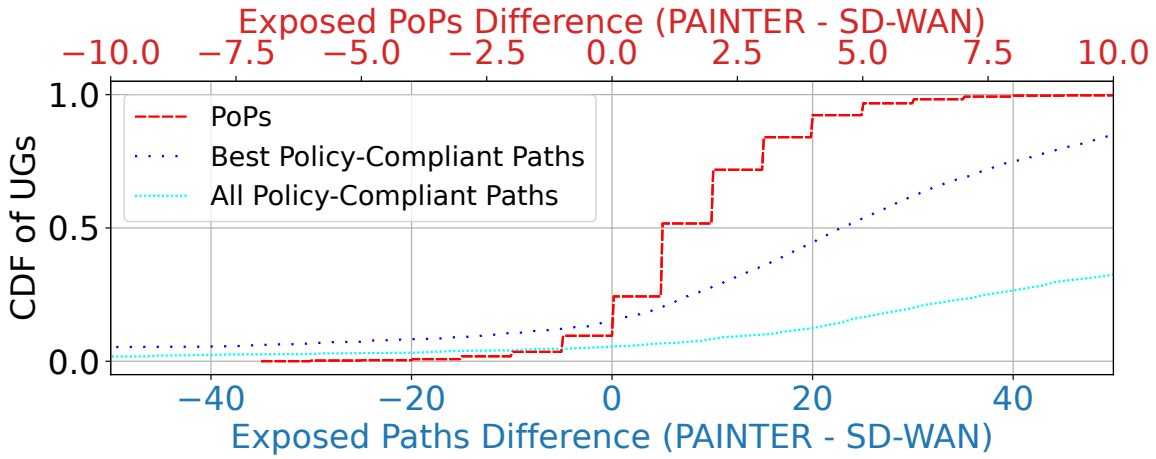
Figure 4.13: PAINTER switches between two paths during site failure much faster than other solutions. First (0 s) PAINTER chooses the prefix 2.2.2.0/24. Second (60 s) site-A fails, so prefix 2.2.2.0/24 is withdrawn and 1.1.1.0/24 reconverges. Third (60.03 s) PAINTER switches over to prefix 3.3.3.0/24 at site-B in approximately 1 RTT.

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 [219] 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 [220]. 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 site-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 site-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).

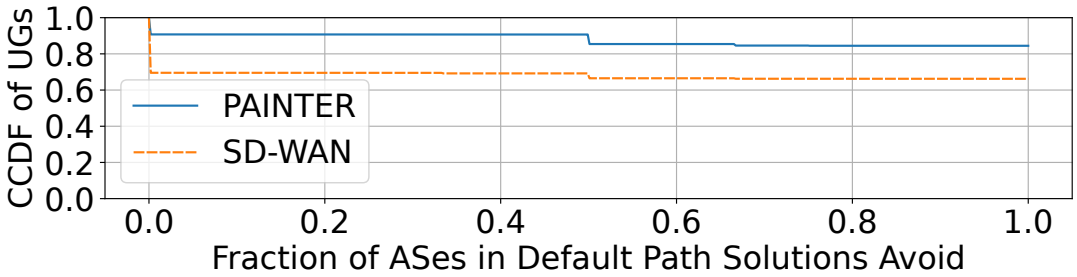
PAINTER's reaction times represent an order of magnitude of improvement over existing techniques—typical BGP convergence times are on the order of minutes [24] and DNS TTLs are usually between 1 and 10 minutes [157, 95]. It took one second for the anycast address to become reachable after withdrawal (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 [95]). Although anycast availability is good compared to DNS (1 second loss), PAINTER's is near optimal (30 ms).

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 4.14a shows that PAINTER exposes more paths



(a)



(b)

Figure 4.14: PAINTER exposes more paths/sites than multihoming (Fig. 4.14a) enhancing resilience to intermediate AS failure (Fig. 4.14b).

and sites than selecting among ISPs, enhancing resilience.

Methodology. We compute paths that SD−WAN and PAINTER would have to Azure for all UG*s*. 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 site 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 sites that Azure clients

may ingress in by looking at the sites at which 90% of user traffic in that UG's geographic region ingress according to Azure logs. We do not consider all sites to remove high-latency routes. This restriction likely does not overestimate low-latency site choices, given that prior work found 90% of traffic in a large CDN reaches a site within 1,000 km of the closest possible [12]. We then count the number of policy-compliant paths from that AS through peerings at each of these sites to Azure according to the common definition of policy-compliant [221] 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 [96].

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 [74].

Results. Most networks have only 2 or three ISPs and so only have 2 or 3 paths to choose from with SD-WAN. Figure 4.14 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 4.14a. All Policy-Compliant Paths indicates that PAINTER *could* expose far more by manipulating advertisements. PAINTER offers policy-compliant routes to 4 more nearby sites than SD-WAN for 10% of UG*s* (sites).

Having more paths could help route around congestion or failures in intermediate ASes. Figure 4.14b 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.

4.5 Related Work

Ingress Traffic Engineering One company uses tight coordination to map hypergiant traffic to ISP ingresses [18]. 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 (§4.1.3). `PAINTER` never requires coordination with networks in between the proxy and the cloud. TIPSY infers where traffic will end up if announcements are withdrawn [42], an orthogonal problem. Anyopt exposes paths using advertisements, but does not scale to deployments with thousands of ingresses [35].

Contemporaneous work called Tango exposed multiple paths by advertising multiple prefixes and tunneled traffic in real-time over the best one [6]. 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 [13]. This difference in focus and scale introduces different challenges that Tango does not address (§4.1.4).

The IETF's Path Aware Networking research group proposed several ingress traffic engineering solutions [194]—`PAINTER` incorporates those lessons (*e.g.,* immediate deployability) into its design. Masque is an IETF effort which could enable performance/security-enhancing QUIC proxies [172]; `PAINTER` uses "proxies" but to enhance route diversity. Emerging/futuristic routing technologies [218, 195] or architectures [16] could facilitate ingress traffic engineering. Miro also exposes more paths and tunnels traffic over these paths [17], 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.

4.6 Summary

PAINTER lowers latency and enhances resilience for cloud services. It is widely deployable due to the growing prevalence of edge proxies, which we demonstrate through extensive data-driven evaluations and by deploying a prototype on the Internet. By embracing 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.

PAINTER, however, does not completely solve the problem of enabling Service Providers to satisfy generic SLO*s*. PAINTER helped lower steady-state latency, but it is unclear if and how that methodology extends to cases where Service Providers wish to satisfy other objectives. We explore this idea in the next chapter.

Chapter 5: Objective-Based Ingress Interdomain Routing

Cloud/content providers (hereafter Service Providers) enable diverse Internet applications used daily by billions of users. Traditionally, Service Providers used DNS and static BGP advertisements to define a single path from a user to a service, leaving the specific path up to the Internet to determine.

Increasingly, however, these services have diverse requirements that can be challenging to meet with a single path that Service Providers have little control over. For example, enterprise services have tight reliability requirements [25, 39], and new applications such as virtual reality require $\leq$ 10 ms round trip latency [1] and $\leq$ 3 ms jitter [40]. Complicating matters, Service Providers must meet these requirements subject to changing conditions such as peering link/site failures [41, 42], DDoS attacks [43, 44, 45], flash crowds [46, 47, 48], new applications/business priorities (LLMs), and route changes [49, 50, 51, 52, 53, 42]. Critically, such changes can cause *overload* if the Service Provider cannot handle new traffic volumes induced by the change, and DNS/BGP are slow mechanisms with which to change how traffic flows over paths. This overload can lead to degraded service for users, hurting reliability [54, 55, 56, 39], and may require manual intervention.

To try to meet different goals and respond to changing conditions, Service Providers adopt solutions that are either (a) too costly, (b) too reactive, or (c) too specific. For example, Service Providers proactively overprovision resources [222, 58, 68], but we demonstrate using traces that bursty traffic patterns can require costly overprovisioning rates as high as 70% despite sufficient global capacity (§5.1.3). `TIPSY` responds to changing loads to retain reliability [42], but only does so reactively, which could lead to short-term degraded performance. `AnyOpt` [35] and `PAINTER` (chapter 4) proactively set up routes to optimize specific objectives such as steady-state latency , but it is challenging to extend those approaches to other objectives (§5.4.4). It is also unclear both how to combine these approaches (*e.g.,* retaining low latency under changing traffic loads) and

how to apply approaches from one domain to another (*e.g.,* applying egress traffic cost reduction systems [15] to ingress traffic).

We present Service Providers with a flexible framework, SCULPTOR (**S**couring **C**onfigurations for **U**tilization-**L**oss-and **P**erformance-aware **T**raffic **O**ptimization & **R**outing), which accepts as input cost, performance, and reliability objectives and outputs BGP advertisements and traffic allocations that help achieve those desired objectives. SCULPTOR is the first system that optimizes ingress interdomain routing objectives such as maximum link utilization, transit cost, and latency for *interdomain* traffic (existing systems optimize for intradomain traffic, *e.g.,* [4, 3, 57, 5]). SCULPTOR computes BGP advertisements proactively, only placing live traffic on them after convergence, and hence does not run the risk of outages.

To solve each optimization problem, SCULPTOR efficiently searches over the large BGP advertisement search space ($> 2^{10,000}$ possibilities) by modeling how different strategies perform, without having to predict the vast majority of actual paths taken under different configurations since predicting interdomain paths is hard and measuring them is slow (§5.2.3). SCULPTOR then optimizes these (modeled) performance metrics using gradient descent, which is appropriate in our setting due to the high dimension of the problem and the parallelism that gradient descent admits (§5.2.4). This modeling enables SCULPTOR to assess > 20M configurations (10,000× more than other solutions [35], Chapter 4) while only measuring tens in the Internet (§5.4.4). Like other work that uses gradient descent with success (*e.g.,* deep learning), we sacrifice the ability to provide a formal characterization of which objective functions are possible for an approach that lets us optimize for multiple criteria (§5.4)

We prototype and evaluate our framework at Internet scale using the PEERING testbed [38] (§5.3), which is now deployed at 32 Vultr cloud locations [73]. Vultr is a global public cloud that allows our prototype to issue BGP advertisements via more than 10,000 peerings. We evaluate our framework on two specific objectives (computed separately): (a) optimizing latency under unseen traffic conditions, and (b) routing different traffic classes. We compare SCULPTOR's performance on both problems to that of an unreasonably expensive "optimal" solution (computing the actual

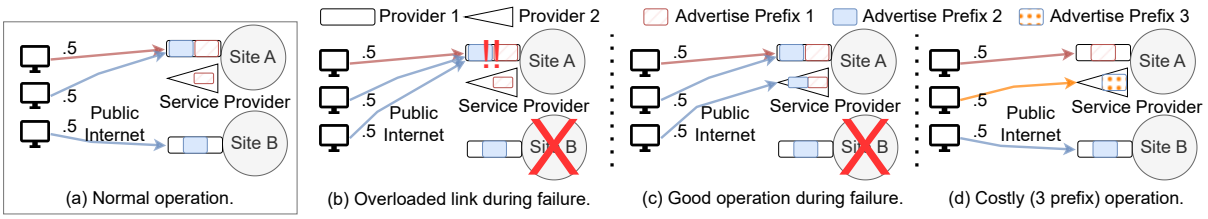


Figure 5.1: In normal operation traffic is split between two sites by directing half the traffic to each prefix (a). Even though there is enough global capacity to serve all traffic when site B fails, there is no way to split traffic across multiple providers given available paths, leading to overload (b). A proactive solution is to advertise prefix 2 to an additional provider at site A, allowing traffic splitting across the two links (c). A simpler but prohibitively expensive solution is to advertise one prefix per peering (d).

optimal is infeasible).

For the first objective, we found that, compared to other approaches, SCULPTOR increases the amount of traffic within 10 ms of the optimal by 19.3% in steady-state (meeting that target for 95% of traffic) (§5.4.2), by 11% during link failure, and by 17% during site failure. SCULPTOR also reduces overloading on links during site failures by up to 41% over PAINTER, giving Service Providers more confidence that services will still be available during partial failure (§5.4.2). We also find that, by load balancing traffic on backup paths during peak times, we can satisfy high peak demands with the same infrastructure. SCULPTOR can handle flash crowds (*e.g.,* DDoS attacks) at more than 3× expected traffic volume, reducing the amount of overprovisioning that Service Providers need, thus reducing costs.

For the second objective, SCULPTOR routes bulk low-priority traffic in ways that avoid congesting high-priority traffic, achieving near-optimal latency and reducing congestion by up to 2× (§5.4.3).

After decades of intradomain traffic optimization using programmable networking primitives such as virtual output queueing for differentiated service [223] (and others, §5.7), SCULPTOR brings such control closer to realization in the interdomain setting with a unilaterally deployable framework. Service Providers can use flexible frameworks such as SCULPTOR to help bring us the resilient, performant service that our diverse applications increasingly need.

5.1 Motivation

5.1.1 Variable, Evolving Goals

Evolving Internet use cases are pushing Service Providers to meet diverse requirements for their applications. For example, Service Providers increasingly host mission-critical services such as enterprise solutions (chapter 4), which require high reliability and are predicted to be a $60B industry by 2027 [161]. Gaming is another important industry generating more revenue than the music and movie industries combined [224], but instead requires latency within 50 ms [1]. Service Providers are also expanding the set of services they provide—for example, CDNs that traditionally hosted static content are pivoting to offering services like compute [225, 226, 227].

Moreover, as use cases evolve, Service Providers increasingly need to meet performance requirements for *ingress* traffic since that traffic includes, for example, player movements in real-time games, voice and video in enterprise conferences, and video/image uploads for AI processing in the cloud [228, 229]. This reality is a departure from traditional CDN traffic patterns where ingress traffic was primarily small requests and TCP acknowledgments.

Despite Service Providers ' efforts to meet variable service requirements [42, 84, 108, 62, 69, 68], meeting requirements is still challenging due to issues *outside* Service Provider control.

First, Service Providers lack control over which ingress path traffic takes since BGP, the Internet's interdomain routing protocol, computes paths in a distributed fashion, giving each intermediate network a say in the ingress path. BGP chooses a single path which may not be the best one for meeting a given requirement. Service Providers cannot even unilaterally control the part of the path closest to them—upgrading peering capacity requires coordination among multiple parties [8].

Second, dynamic factors outside Service Provider control make satisfying requirements even harder. Peering disputes can lead to congestion on interdomain links and inflated paths [41, 230, 231], and DDoS attacks still bring down sites/services [45, 54, 55, 56], despite the considerable effort in mitigating DDoS attack effects [43, 44]. Moreover, recent work [48, 53, 5] and blog posts

[46, 47, 49, 50, 51, 52] show that user traffic demands are highly variable due to flash crowds and path changes and so can be hard to plan for.

5.1.2 Routing Traffic to Service Providers

One way of meeting service requirements, despite this lack of control, is to (a) configure a good set of interdomain paths and (b) balance traffic on these paths to satisfy requirements. Existing systems balance traffic over egress interdomain paths [8, 9, 3, 10], but it remains unclear how to configure and balance traffic across a good set of *ingress* interdomain paths.

Today, most Service Providers either use `anycast` prefix advertisements to provide low latency and high availability at the expense of some control [29, 12, 24, 36], or `unicast` prefix advertisements to direct users to specific sites [232, 31, 8, 32]. `Anycast`, where Service Providers advertise a single prefix to all peers/providers at all sites, offers some natural availability following failures since BGP automatically reroutes most traffic to avoid the failure after tens of seconds [24]. Prior work shows that this availability comes with higher latency in some cases [27, 12]. `Unicast` gives clients lower latency than `anycast` by advertising a unique prefix at each site [232, 32], but can suffer from reliability concerns, taking as much as an hour to shift traffic away from bad routes (chapter 3).

5.1.3 Limitations of Current Approaches

Too Specific: Recent work has similarly noted that, by offering better interdomain routes, Service Providers can achieve better performance [41, 29, 35, 36] (chapter 4). These solutions advertise different prefixes to subsets of all peers/providers to offer users more paths. However, these solutions optimize specific objectives, mostly steady-state latency, and it is unclear how to extend their approaches to other objectives (§5.4.4), especially those with capacity constraints.

Figure 5.1 shows how this specific focus on a single objective could lead to reliability issues, *e.g.,* during a site failure. In normal operation, user traffic is split evenly across two prefixes and achieves low latency (Fig. 5.1a). However when site B fails, BGP chooses the route through

Provider 1 for all traffic, causing link overutilization and subsequent poor performance for all users (Fig. 5.1b). In Section 5.4 we show that this overload happens in practice for state-of-the-art systems that provide low (steady-state) latency from users to Service Provider networks such as `AnyOpt` [35] and `PAINTER` (chapter 4).

Too Reactive: To enhance reliability, some Service Providers propose systems that react to changing conditions—for example, `TIPSY` drains traffic from overloaded links/sites [42]. However, reactivity still requires time to change to a new state, which can lead to short-term service degradation and additional uncertainty at a time where unexpected things are already occurring. A more desirable approach would be to never change announcements carrying live traffic, which is known to sometimes lead to outages [233, 234, 235, 68].

Too Costly: Another approach is to overprovision resources to handle transient peak loads [222, 58, 68]. Figure 5.2 demonstrates however, using longitudinal link utilization data from OVH cloud [236], that doing so can incur excessive costs. Meta also stated that peering capacity is hard to change since it cannot be unilaterally upgraded [8], and Microsoft said the same because of lead time [41].

Upgrading capacity has fixed costs (*e.g.,* fiber, router backplane bandwidth, line card upgrades, CDN servers near peering routers) and variable costs (*e.g.,* power, transit), both of which scale with demand and Service Providers strive to keep low [237, 58, 5, 15]. We model deployment cost as correlated with the peering capacity over all links/sites, even though the actual relationship may be complex and additional peering capacity itself is not prohibitively expensive.

To generate Figure 5.2 we first split the dataset into successive, non-overlapping 120-day planning periods and compute the 95^{th} percentile ingress link utilization ("near-peak load") for each link/period. The dataset reports utilizations over 5-minute intervals. We then assign future capacities for each period by setting each link's capacity for the next period as the near-peak load in the current period multiplied by some overprovisioning factor (X-axis variable). We then compute the average link utilization in the next period and the total number of links on which we see $\geq 100\%$ utilization in at least one 5-minute interval.

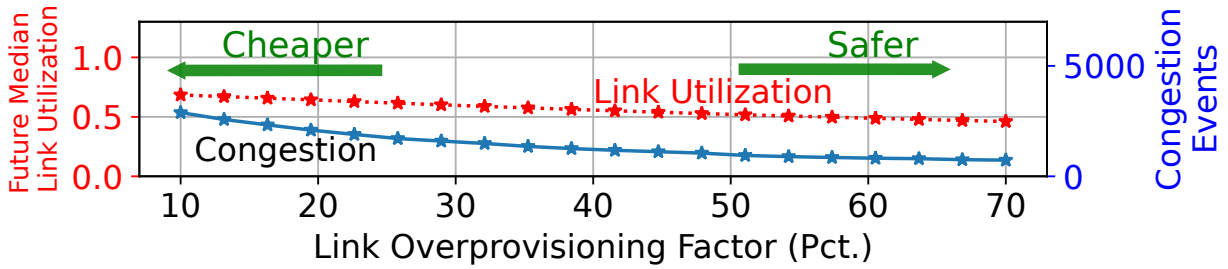


Figure 5.2: Planning for peak loads to avoid overloading requires inefficient overprovisioning.

Figure 5.2 plots the median utilization across links and periods and the number of congestive events as we vary the overprovisioning factor. Figure 5.2 shows that overprovisioning to accommodate peak loads introduces a tradeoff between inefficient utilization and overloading. Low overprovisioning factors between 10% and 30% lead to more efficient utilization (60%-70%) but lead to thousands of congestive events. High overprovisioning factors lead to far less overloading, but only 50% utilization.

Efficient, Proactive Planning is Possible: Despite these limitations, Figure 5.1c shows that by advertising prefix 2 to provider 2 at site A a priori, the Service Provider can split traffic between the two links during failure, avoiding overloading without (a) reacting to the failure and (b) upgrading deployment capacity.

5.1.4 Key Challenges

Since a Service Provider has lots of global capacity, dynamically placing traffic on paths to optimize performance objectives subject to capacity constraints would therefore be simple if all the paths to the Service Provider were always available to all users as in Figure 5.1d. However, making paths available uses IPv4 prefixes, which are expensive and pollute BGP routing tables. The solution in Figure 5.1d uses 50% more prefixes than the solution in Figure 5.1c and, at $20k per prefix for 10,000 peerings would cost $200M [196]. IPv6 is not a good alternative, (a) as IPv6 is not supported in all access networks [238] and (b) IPv6 addresses take 8× the amount of memory to store in a router so would pollute global tables even more. Prior work [35] and this dissertation chapter 4) noted the same but found that advertising around 50 prefixes was acceptable, and most Service Providers advertise fewer than 50 today according to RIPE RIS BGP data [219]. We also

verified with engineers at six Service Providers that this challenge important to address.

Since we cannot expose all the paths by advertising a unique prefix to each connected network, we must find some subset of paths to expose.

Finding that right subset of paths to expose that satisfies performance objectives, however, is hard since there are exponentially many subsets to consider, and each subset currently needs to be tested (*i.e.,* advertised via BGP) to see how it performs. The number of subsets is on the order of 2 to the number of ingresses Service Providers have, which, for some Service Providers (including Vultr, which we measure) [14], is $> 2^{10,000}$. As measuring this many advertisements is intractable, we have to predict how different subsets of paths perform, which is challenging since interdomain routing is difficult to model [239, 42].

5.2 Methodology

5.2.1 SCULPTOR Overview

SCULPTOR's goal is to find an advertisement strategy that gives users good interdomain paths (relative to some objective) and a traffic allocation to those paths. SCULPTOR computes BGP advertisements proactively, only placing live traffic on them after convergence. For objectives we consider, resulting optimization problems are often large with more than 100M constraints and 2M decision variables (§5.2.2). Our framework breaks this challenging problem into manageable components — Figure 5.3 shows the interactions among these components.

Minimizing an objective function requires evaluating it with many different inputs, but performing such measurements (*i.e.,* advertising prefixes) could take years at our problem size (§5.2.3) and so is not scalable. Instead, we estimate the objective using a "Probabilistic Routing Model" (§5.2.3) and update this model over time by "Advertising & Measuring" a small number of advertisements in the Internet using entropy-based "Exploration" (§5.2.3).

We then minimize the objective function using gradient descent ("∇G") which, at each iteration, requires solving millions of sub-problems for traffic allocations ("Inner Loop Workers", §5.2.4). These sub-problems can either be solved exactly with a "General Purpose Solver" (§5.2.4)

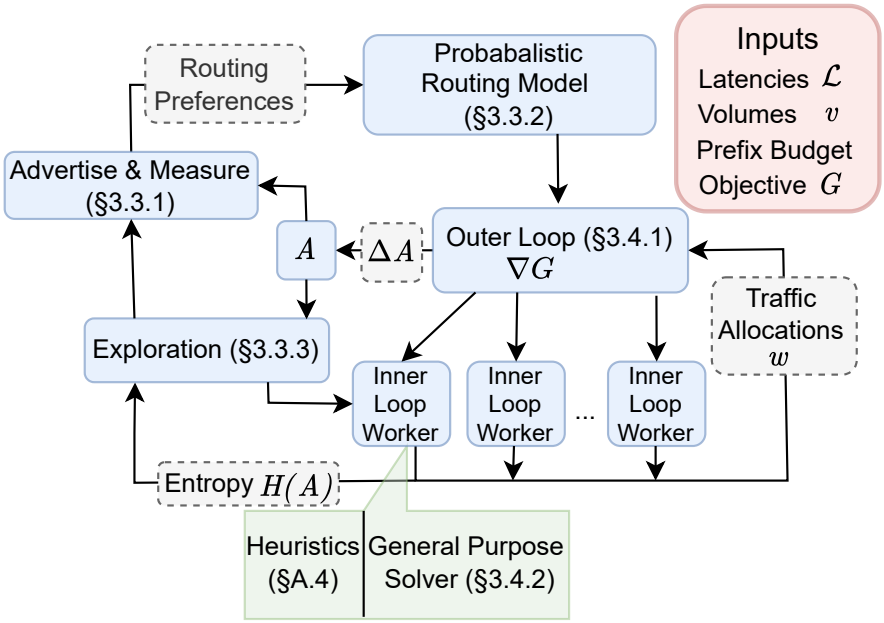


Figure 5.3: SCULPTOR advertisement computation overview.

or approximately with efficient heuristics (§5.6.5).

5.2.2 Problem Setup and Definitions

Setting

Service Providers offer their services from tens to hundreds of geo-distributed sites [2]. The sites for a particular service can serve any user, but users benefit from reaching a low-latency site for performance. Sites consist of sets of servers that have an aggregate capacity. Service Providers also connect to other networks at sites via dedicated links or shared IXP fabrics which we call peering links. Each such link also has a capacity. When utilization of a site or link nears/exceeds the capacity, performance suffers, so Service Providers strive to avoid high utilization [41, 5]. Resources can also fail due to, for example, physical failure and misconfiguration.

Users route to the deployment through the public Internet to a prefix over one of the peering links via which that prefix is advertised. The path to a prefix (and therefore peering link) is chosen via BGP. We fix the maximum number of allowable prefixes according to the Service Provider's budget, which is generally much less than the number of peerings.

We model interdomain paths from users to the Service Provider as non-overlapping, except at the peering link through which each path ingresses to the Service Providers ' deployment. We assume the peering link is the bottleneck of each path for two reasons. First, this link increasingly represents the most important part of the path for Service Providers due to Internet flattening [7]. Second, other parts of the path are less interesting from an optimization perspective—last-mile bottlenecks are common to all paths for a user, and existing systems optimize intradomain paths in other networks. Handling bottlenecks can also be viewed as an "Unseen Scenario" (§5.2.2) so SCULPTOR handles violations of this assumption. A user may have many paths to a site through different links. A path to a prefix will be through one of the corresponding ingress links over which that prefix is advertised.

We consider users at the granularity of user groups (UG*s*), where a UG refers to user networks that route to the Service Provider similarly, but could mean different things in different instantiations of our system (*e.g.,*/24 IPv4 prefixes, metros). UG*s* generate steady-state traffic volumes, v_{UG}, and the Service Provider provisions capacity at links/sites to accommodate this load. A system run by the Service Provider measures latency from UG*s* to Service Provider peering links *l*, which is a reasonable assumption [74, 32, 206, 240, 35] (chapter 4).

We assume that the Service Provider exposes some technology to SCULPTOR for directing traffic towards prefixes. Examples include DNS [232, 32, 29, 31], multipath transport [241, 190, 242], or control-points at/near user networks [173] (chapter 4). DNS offers slow redirection due to caching (chapter 3) but is the most deployable, whereas Service Provider-controlled appliances offer precise control but may be hard to deploy. Multipath transport will eventually see wide enough deployment to be used by all Service Providers. Today, MPTCP is enabled by default in iOS [191] and Ubuntu 22 [243], all applications can use MPQUIC [105].

General Formulation

The problem of finding advertisement configurations that admit good assignments of user traffic to paths is a multivariate optimization over both configurations and traffic assignments.

We represent an advertisement configuration A as a binary vector. Each entry indicates whether we advertise/do not advertise a particular prefix to a particular peer/provider (similar to prior work [35] and PAINTER (chapter 4)). Implementing this configuration (*i.e.,* advertising prefixes via BGP sessions) results in routes from UG*s* to the Service Provider. The assignment of traffic to resulting routes from this configuration is given by the nonnegative real-valued vector w, whose entries specify traffic allocation for each UG along each route.

Announcing a configuration will result in some set of routes (although knowing exactly which routes a priori is challenging), and these routes define which ingress link a UG uses for a prefix. We can think of this process as a routing function R that takes an advertisement configuration A and outputs a map from $\langle \texttt{prefix, UG} \rangle$ pairs to ingress links. For example, say $R(A) = f_A$, and $f_A(p, \texttt{UG}) = l$ — this notation means that the output of an advertisement configuration A under routing R is a function f_A that tells us that users UG reach prefix p via link l. It could be that a configuration leads to no route for some UG to some prefix. We define a function e such that $e(R(A)(p, \texttt{UG})) = 1$ when there is some route for UG to prefix p under configuration A, and 0 otherwise.

Now suppose the overall metric we want to minimize is G which is a function of both configurations and traffic assignments. Examples include traffic cost, average latency, maximum latency, and their combinations (see §5.6.3 for a discussion of which metrics may work better than others). The joint minimization over configurations and traffic assignments can then be expressed as the following.

$$\begin{aligned} \min_{A,w} \quad & G(R(A), w) \\ \text{s.t.} \quad & w(p, \texttt{UG}) \geq 0; \quad A(p, l) \in \{0, 1\} \quad \forall\, \texttt{UG}, p, l \\ & \sum_{p} w(p, \texttt{UG}) e(R(A)(p, \texttt{UG})) = v(\texttt{UG}) \quad \forall\, \texttt{UG} \end{aligned} \tag{5.1}$$

The first constraint requires that traffic assignments be nonnegative and that configurations are binary. The second constraint requires that all user traffic $v(\texttt{UG})$ is assigned, and none is assigned

to nonexistent paths. Capacity constraints depend on the choice of G.

Specific Objectives

In our evaluations we focus on two specific objectives: (1) minimizing latency and maximum link utilization in unseen scenarios, and (2) optimally routing different traffic classes. We believe the framework will accommodate objectives that have been met in other networking settings [15, 8, 63, 35, 32, 5, 4, 3, 64], although we provide no formal characterization of which objectives definitely work (because the gradient descent approach it uses does not provide formal guarantees (§5.6)). With our interdomain path model (§5.2.2), steady-state path latency for a UG to a prefix is uniquely determined by UG and the corresponding peering link over which the UG ingresses (but latency may change, which we evaluate in Section 5.4.2. Let the latency for a user UG via a link l be $\mathcal{L}(\text{UG}, l)$. G is then given by the following.

$$\begin{aligned} G(R(A), w) &= \frac{1}{\sum_{\text{UG}} v(\text{UG})} \sum_{p,\text{UG}} \mathcal{L}(\text{UG}, R(A)(p, \text{UG}))w(p, \text{UG}) \;+\; \beta M \\ \text{s.t.} \;& \frac{\sum_{R(A)(p,\text{UG})=l} w(p, \text{UG})}{c(l)} \leq M \quad \forall\, l \end{aligned} \tag{5.2}$$

The capacity for link l is $c(l)$. Constraining maximum link utilization, M, to be at least as much as the utilization of each link and then minimizing a sum including it forces M to be the maximum link utilization. The sum of average latency and maximum link utilization is weighted by a parameter, β, which represents a tradeoff between using uncongested links/sites and low propagation delay and is set by the Service Provider based on their goals. We first solve Equation (5.2) with M = 1 to see if we can allocate traffic to paths with zero overloading, which may not be possible for all A.

Unseen Scenarios To encourage good solutions to Equation (5.1) not only in steady-state but also in unseen conditions (failures, shifting traffic distributions), we add a term to G given by $\sum_l \alpha_l G(R(A * F_l), w)$. The vectors F_l are binary vectors defined so that multiplying advertisements, A, by these vectors simulates withdrawal/failure on a link/site (*i.e.,* zeroing out the cor-

responding components). This term (*i.e.,* regularizer) intuitively encourages advertisement solutions to provide UG*s* good primary and backup paths, thus preparing for dynamic conditions. In Section 5.6.2 we discuss why this regularizer helps us find a good global minimum by drawing analogies to the deep learning literature.

Different Traffic Classes To demonstrate that SCULPTOR works with other useful objectives, we consider a separate problem where we try to route traffic with different performance requirements. Service Providers already perform multi-class traffic engineering on their private WANs [3, 4] and would increasingly benefit from such capabilities in an interdomain setting due to varied service offerings (§5.1.1), but they currently have no capability to do so. We use a similar scenario to the private WAN setting where one traffic class (high-priority) should be routed with low latency, and the other (low-priority) should not congest the high-priority traffic. A key additional challenge in the interdomain setting is that we cannot use priority queueing to ensure that high-priority traffic is not congested since we do not control queue behavior on the path. The objective function is a weighted combination of the average latency of high-priority traffic and the amount of high-priority traffic that is congested. We also constrain the maximum low-priority overprovisioning on any link so that not all low-priority traffic lands on one link, as this solution would lead to poor low-priority goodput.

5.2.3 Predicting Interdomain Routes

Solving Equation (5.1) is challenging because we need to compute $G(R(A), w)$, but measuring $R(A)$ exactly requires advertising prefixes in the Internet which can only be done infrequently to avoid route-flap-dampening. Our optimization (detailed below) requires evaluating $G(R(A), w)$ for millions of different A which could take a hundred years at a rate of advertising one strategy every 20 minutes. Hence we model, instead of measure, UG paths and improve this model over time through relatively few measurements.

Initialization and Measuring $R(A)$

We initialize our strategy to be `anycast` on one prefix, and `unicast` on remaining prefixes (*i.e.,* one prefix per site). If we have more prefixes in our budget, we randomly do/do not advertise those prefixes via random ingresses. We do not use a completely random initialization since `anycast` and `unicast` have their benefits (§5.1.2).

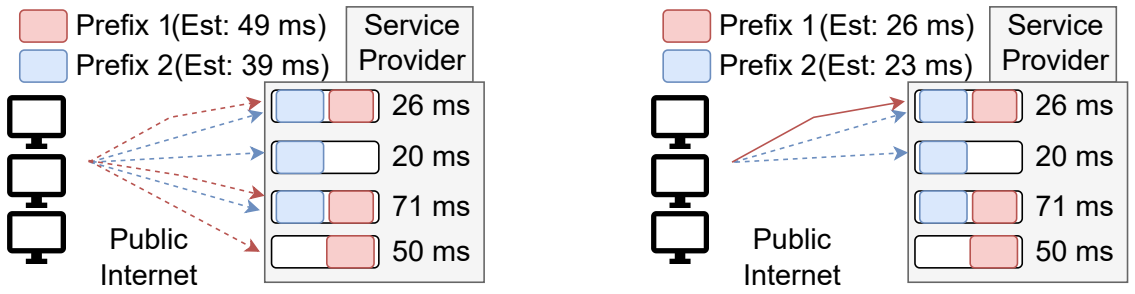
During optimization we measure $R(A)$ by occasionally advertising the corresponding prefixes via peerings as specified by A and measuring routes taken by `UG`*s* to each prefix. We alternate between measuring advertisements we think are good (§5.2.4) and ones we think offer useful information (§5.2.3).

Probabilistic `UG` Paths

We model the routing function, and therefore our objective functions, probabilistically and update our probabilistic model over time as we measure how `UG`*s* route to the deployment. When, during optimization, we require a value for $G(R(A), w)$, we compute its expected value given our current probabilistic model. We compute this expected value either approximately via Monte-Carlo methods, or exactly if G's structure admits efficient computation (§5.6.5).

Our probabilistic model assumes a priori, for a given `UG` towards a given prefix, that all ingress link options that prefix is advertised to that are reachable from a `UG` are equally likely. (Providers can discern the set with high reliability based on which ingresses the `UG` is reachable from.) Upon learning that one ingress is preferred over the other, we exclude that less-preferred ingress as an option for that `UG` in all future calculations for all prefixes for which both ingresses are an option. Prior work [35, 239] and this dissertation (chapter 4) used a similar routing model but did not extend it to deal with general objectives nor did it treat routing probabilistically.

A useful property of many choices of objective function G is that we do not need to approximate the routing function perfectly to solve Equation (5.1). For example, most `UG`*s* have similar latency via their many paths to a single site [217], so when optimizing latency as in Equation (5.2) we effectively need to predict the site the user ingresses at. As we exclude more options of where



(a) (b)

Figure 5.4: We initially estimate latency from this UG to both the red and blue prefixes with the average over possible ingress latencies (5.4a). A priori, no prefix is advertised. We then advertise the red prefix and learn that the first ingress has higher preference than the third and fourth (5.4b). We use this information to refine our latency estimate towards the blue prefix (since the third ingress is no longer a possible option) without advertising the blue prefix, saving time.

UG*s* could possibly ingress for each prefix, our objective function's distribution on unmeasured scenarios converges to the true value.

An example of this process is shown in Figure 5.4, where we refine a latency estimate towards an unmeasured prefix (blue) using measurements towards other prefixes (red). Even though we have 50% confidence in which path the user takes (*i.e.,* in $R(A)$), we have high confidence that the user's latency towards the blue prefix will be about 23 ms.

Exploration to Refine $R(A)$

Probabilistic estimates of G could be noisy until we learn UG preferences. To refine our model, we periodically measure $R(A)$ on adjacent configurations to the configuration at the current optimization step (§5.2.4). By adjacent configurations, we mean ones that differ from the current configuration, A, by one entry, or configurations representing failures ($A \times F_l$). Of the adjacent options, we select the one for which the model is most uncertain about whether it will be better or worse (*i.e.,* the probabilistic options include both). One could use various measures of uncertainty—we choose entropy. We measure adjacent configurations since we use a descent-based optimization (§5.2.4) which benefits from knowing whether nearby strategies are better.

5.2.4 A Two Pronged Approach

Even with our probabilistic model, we cannot solve Equation (5.1) directly since it is a mixed-integer program with millions of constraints which is too large for off-the-shelf solvers. Greedy approaches like in `PAINTER` (chapter 4) and random approaches like in `AnyOpt` [35] find good solutions in this setting for simple objectives such as steady-state latency, but, without more intelligent search through the large space, those approaches can converge to poor solutions on other objectives (§5.4.4).

Instead, we split Equation (5.1) into an outer and inner optimization, solving for configurations, A, in the outer loop using gradient descent and traffic assignments, w, in the inner loop using a general-purpose solver. Intuitively, this approach works because it breaks up a challenging optimization into many sub-problems that can be solved in parallel, and is guaranteed to push us towards *a* (perhaps locally optimal) solution.

Outer Loop: Gradient Descent

To apply gradient descent to our problem, we extend A to have real (instead of binary) entries between 0 and 1 and threshold its entries at 0.5 to determine if a prefix is advertised to a certain peer/provider. We approximate gradients between adjacent advertisements by computing the expected value of $G(R(A), w)$ at each advertisement and continuously interpolating G at intermediate configurations using sigmoids (like prior work in a different domain [244]). Since there are too many gradients to compute, we subsample gradient entries and track the largest ones (like prior work in a different domain [245]).

Scaling: Minimization in the outer loop scales with the product of the number of ingresses and number of prefixes. Despite this high complexity, in practice the implementation runs quickly relative to our announcement rate (20-50 announcements, 20 minutes - 2 hours per announcement). Gradient computations are parallelizable so this loop scales horizontally to where announcements are the bottleneck.

Convergence: Gradient descent converges to a local minimum for bounded objectives [246].

Our evaluations show that SCULPTOR finds good solutions over a wide range of simulated topologies and converges quickly with thousands of UG*s* and ⟨peering, prefix⟩ pairs (§5.4). We comment on which problem properties will likely lead to faster convergence to more a optimal minimum in Section 5.6.2 by drawing analogies to the deep learning literature. Allocating higher prefix budgets and adding richer advertisement capabilities (*e.g.,* BGP community tagging) can lead to convergence to a better minimum, which is an area for future work.

Inner Loop: General Purpose

Each iteration of the outer loop requires solving for traffic allocations on advertisements corresponding to the gradient entries that we wish to compute. We refer to the process of solving for these traffic allocations as an inner loop.

Scaling: In the case where we use Monte Carlo methods to approximate $G(R(A), w)$, we solve for allocations given several randomly generated $R(A)$. Hence the number of iterations in the inner loop scales with the product of the number of Monte Carlo simulations and the scaling behavior of each traffic allocation, which depends on the objective.

Convergence depends on the objective, G. Our objectives (§5.2.2) and many others can be expressed as linear programs, so solvers can find globally optimal solutions. Non-convex objectives such as Cascara's sometimes have efficiently computable solutions [15].

5.3 Implementation

We prototype SCULPTOR on the PEERING testbed [38], which is now available at 32 Vultr cloud sites. We describe how we built SCULPTOR on the real Internet and how we *emulate* a Service Provider including their clients, traffic volumes, and resource capacities. (We are not a Service Provider and so could not obtain actual volumes/capacities, but our extensive evaluations (§5.4) demonstrate SCULPTOR's potential in an actual Service Provider and our open/reproducible methodology provides value to the community.)

5.3.1 Simulating Clients and Traffic Volumes

To simulate client performances, we measured actual latency from IP addresses to our PEERING prototype as we did with PAINTER (chapter 4), and selected targets according to assumptions about Vultr cloud's client base.

We first tabulate a list of 5M IPv4 targets that respond to ping via probing each /24. Vultr informs cloud customers of which prefixes are reachable via which peers, and we use this information to tabulate a list of peers and clients reachable through those peers. We then measure latency from all clients to each peer individually by advertising a prefix solely to that peer using Vultr's BGP action communities and pinging clients from Vultr. We also measure performance from all clients to all providers individually, as providers provide global reachability.

In our evaluations, we limit our focus to clients who had a route through at least one of Vultr's direct peers (we exclude route server peers [247]). Vultr likely peers with networks with which it exchanges a significant amount of traffic [8], so clients with routes through those peers are more likely to be "important" to Vultr. We found 700k /24s with routes through 1086 of Vultr's direct ingresses. In an effort to focus on interesting optimization cases, we removed clients whose lowest latency to Vultr was 1 ms or less, as these were assumed to be addresses related to infrastructure, leaving us with measurements from 666k /24s to 825 Vultr ingresses.

As we do not have client traffic volume data, we simulate traffic volumes in an attempt to both balance load across the deployment but also encourage some diversity in which clients have the most traffic. To simulate client traffic volumes, we first randomly choose the total traffic volume of a site as a number between 1 and 10 and then divide that volume up randomly among clients that anycast routes to that site. Client volumes in a site are chosen to be within one order of magnitude of each other. Although these traffic volumes are possibly not realistic, in demonstrating the efficacy of SCULPTOR over a wide range of subsets of sites and simulated client traffic volumes, we demonstrate that SCULPTOR's benefits are not tied to any specific choice of sites or traffic pattern within those sites.

5.3.2 Deployments

We use a combination of real experiments and simulations to evaluate SCULPTOR. Both cases use simulated client traffic volumes, but our real experiments measure real paths using RIPE Atlas probes, while our simulations use simulated paths.

We implement SCULPTOR with Nesterov's Accelerated Gradient Descent (§5.2.4) in Python [248]. We set $\alpha_l = 4.0$ and set the learning rate to 0.01 with decay over iterations. We solve traffic allocations (§5.2.4) with Gurobi [249].

Experiments in the Internet We assess how SCULPTOR performs on the Internet using RIPE Atlas probes [72], which represent a subset of all clients. RIPE Atlas allows us to measure paths (and thus ingress links) to prefixes we announce from PEERING, which SCULPTOR needs to refine its model (§5.2). We limit the scale of our deployment to 10 sites to avoid reaching RIPE Atlas daily probing limits (15k traceroutes/day). We choose a deployment with high geographic density rather than greater geographic coverage, as we believe the proximity creates a more interesting routing surface to optimize over (differences from unicast could be smaller, for example). These 10 sites were Miami, New York, Chicago, Dallas, Atlanta, Paris, London, Stockholm, Sao Paulo, and Madrid. Choosing RIPE Atlas probes to maximize network coverage and geographic diversity, we select probes from 972 networks in 38 countries which have paths to 484 unique ingresses. Each probe has paths via approximately 60 ingresses. We use 12 prefixes.

Simulations We also evaluate SCULPTOR by simulating user paths which allows us to conduct more extensive evaluations, as experiments take less time and use clients in more networks. We compute solutions over many random routing preferences, demands, and subsets of sites to demonstrate that SCULPTOR's benefits are not limited to a specific deployment. We evaluate SCULPTOR over deployments of size 3, 5, 10, 15, 20, 25, and 32 sites. Each size deployment is run at least 10 times with random subsets of UG*s*, UG demands, and routing preferences. The distribution of the number of /24s per peer is Pareto-like, so we consider random subsets of /24s through each ingress in a way that balances the number of unique /24s per ingress. Over all scenarios, we consider paths from clients in 52k prefixes (representing 31% of APNIC population [127]) to 873

ingresses. We use one tenth of the number of ingresses as the number of prefixes in our budget (10 prefixes for 3 site deployments, 60 prefixes for 32 site deployments). Prior work [35] and this dissertation (chapter 4) found that using tens of prefixes to improve performance was a reasonable cost.

5.3.3 Setting Resource Capacities

We assume that resource capacities are overprovisioned proportional to their usual load. However, we do not know the usual load of Vultr links and cannot even determine which peering link that traffic to one of our prefixes arrives on, as Vultr does not give us this information. (This limitation exists since we are not a Service Provider, but a Service Provider could measure this using `IPFIX`, for example.) We overcome this limitation using two methods corresponding to our two deployments in Section 5.3.2.

Experiments in the Internet For our first method of inferring client ingress links, we advertise prefixes into the Internet using the PEERING testbed [38], and measure actual ingress links to those prefixes using traceroutes from RIPE Atlas probes [72]. Specifically, we perform IP to AS mappings and identify the previous AS in the path to Vultr. This approach has limited evaluation coverage, as RIPE Atlas probes are only in a few thousand networks. In cases where we cannot infer the ingress link even from a traceroute, we use the closest-matching latency from the traceroute to the clients' (known) possible ingresses. For example, if an uninformative traceroute's latency was 40 ms to Vultr's Atlanta site and a client was known to have a 40 ms path through `AS1299` at that site, we would say the ingress link was `AS1299` at Atlanta.

Simulations The second method we use to infer ingress links is simulating user paths by assuming we know all user routing preference models (§5.2.2). We use a preference model where clients prefer peers over providers, and clients have a preferred provider. When choosing among multiple ingresses for the same peer/provider, clients prefer the lowest-latency option. We also add in random violations of the model. This second approach allows us to evaluate our model on all client networks but may not represent actual routing conditions, though prior work found it held

in 90% of the cases they studied. However, we found that our key evaluation results (§5.4) hold regardless of how we simulated routing conditions (we also tried random preference assignments).

Given either method of inferring client ingress links (RIPE Atlas/simulations), we then measure paths to an `anycast` prefix and assign resource capacities as some overprovisioned percentage of this catchment. (Discussions with operators from Service Providers suggested that they overprovision using this principle.) We report results for an overprovisioning rate of 30%, but find similar takeaways for 10% through 50%.

5.4 Evaluation

5.4.1 General Evaluation Setting

We compare `SCULPTOR` to other solutions.

`Anycast`: A single prefix announcement to all peers/providers at all sites.

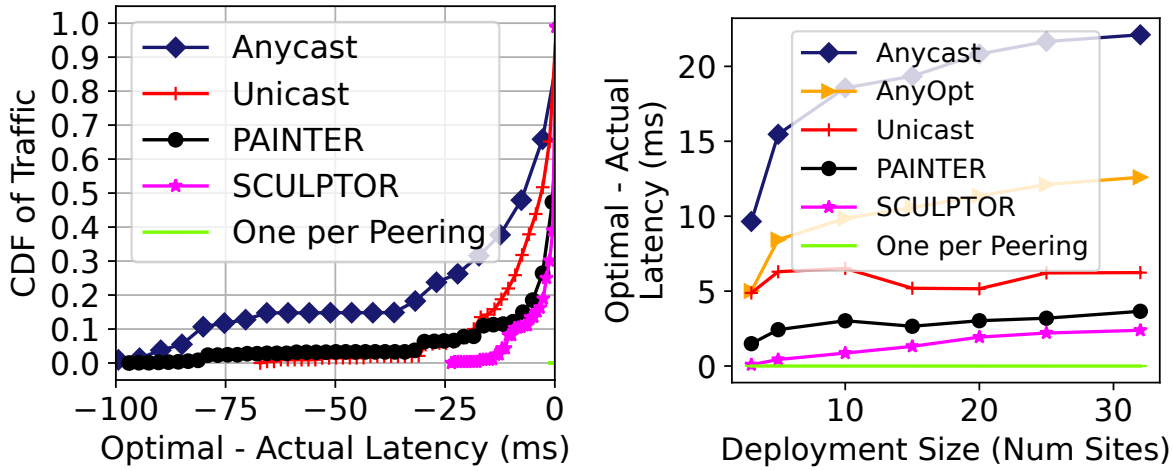
`Unicast`: A single prefix announcement to all peers/providers at each site (one per site).

`AnyOpt`: A proposed strategy for reducing steady-state latency compared to `anycast` [35]. We do not compute this solution for our evaluations on the Internet since it did not perform well compared to any other solution in our simulations, and since it takes a long time to compute.

`PAINTER`: Our proposed strategy for reducing steady-state latency in Chapter 4.

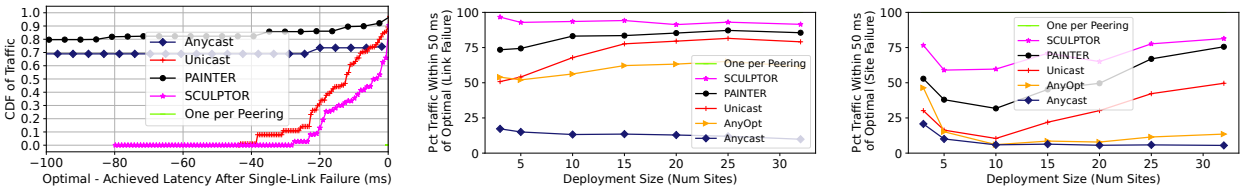
`One-per-Peering`: A unique prefix advertisement to each peer/provider, so many possible paths are always available from users. This solution serves as our performance upper-bound, even though it is prohibitively expensive. (We do not know an optimal solution with fewer prefixes.)

We compute both average overall latency and the fraction of traffic within 10 ms (very little routing inefficiency), 50 ms (some routing inefficiency), and 100 ms (lots of routing inefficiency) of the `One-per-Peering` solution for each advertisement strategy, as these statistics provide a more informative measure of latency improvement than averages.



(a) (b)

Figure 5.5: SCULPTOR provides the lowest steady-state latency both on the Internet (5.5a) and in simulation (5.5b).



(a) (b) (c)

Figure 5.6: SCULPTOR improves resilience to failure both on the Internet (5.6a) and in simulation (5.6b, 5.6c).

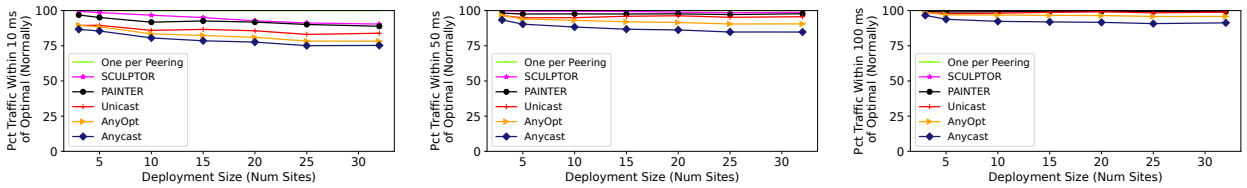
5.4.2 Handling Unseen Conditions

In optimizing for user latency (§5.2.2), SCULPTOR achieves that objective both during steady-state and also during failures and conditions unseen at the time of optimization.

For context, at the scale of Service Providers that serve trillions of requests per day, improving a few percent of traffic by tens of milliseconds represents a significant improvement. Service Providers recently emphasized that small percentage gains are important [250, 251].

Lower Latency in Steady-State

Internet Deployment Figure 5.5a shows a CDF of the difference in latency between each solution and One-per-Peering for all UG*s*, weighted by traffic. SCULPTOR outperforms other solutions and is only 2.0 ms worse than (the unreasonably expensive) One-per-Peering so-



(a) (b) (c)

Figure 5.7: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of their optimal latency during normal operation.

lution on average, whereas the next best solution, PAINTER, is 5.5 ms worse. SCULPTOR also serves 91.8% of traffic within 10 ms of the latency with which One-per-Peering serves it, whereas the same is true for only 88% of traffic for PAINTER.

Simulations Figure 5.5b shows the average latency compared to the One-per-Peering solution over all simulated deployments at each deployment size. Average latency for SCULPTOR ranges from 0.1 to 2.4 ms worse than One-per-Peering. The next-best solution (PAINTER) is on average 1.1 ms and 2.2 ms worse than SCULPTOR. Interestingly, unicast (5.7 ms worse than One-per-Peering) performs better than AnyOpt (10.0 ms worse than One-per-Peering), which could be due to the different setting AnyOpt was designed for. AnyOpt was designed to optimize latency without capacity constraints over provider connections, which does not capture that many Service Providers have many peers and limited capacity [8].

These average latency improvements translate into quantifiable routing inefficiency for different fractions of traffic. Figure 5.7 shows how SCULPTOR compares more favorably to the One-per-Peering solution than other advertisement strategies. SCULPTOR has on average 94.9% traffic within 10 ms of the One-per-Peering solution, 99.2% within 50 ms, and 99.9% within 100 ms. These percentages compare favorably to the next-best solution, PAINTER, which has on average 92.3% traffic within 10 ms of the One-per-Peering solution, 97.7% within 50 ms, and 99.2% within 100 ms.

Better Resilience to Failure

We next assess SCULPTOR's ability to gracefully handle a type of *unseen* condition—ingress failures and site failures. Examples include excessive DDoS traffic on the link/site (thus using the link/site as a sink for the bogus traffic), physical failure, resource draining, changes in latency on a path, and planned maintenance. Figure 5.6 demonstrates that SCULPTOR shifts traffic without overloading alternate links/sites more effectively than any other solution, *without reactive BGP changes* which could cause further failure (§5.1.3).

Here, we fail each ingress/site once and compute traffic allocations. For each advertisement strategy and failed component, we compute the difference between One-per-Peering and achieved latency for UG*s* that use that component in steady-state. For example, if the Tokyo site fails, we report on the post-failure latency of UG*s* that were served from Tokyo before the failure and not of other UG*s*.

In solving for traffic allocations during failure scenarios, links may be overloaded. We say all traffic arriving on a congested link is congested and do not include this traffic in latency comparisons (congested traffic latency would be a complicated function of congestion control protocols and queueing behavior). We separately note the fraction of traffic that lands on congested links and do not include it in average latency computations, but say such traffic does *not* satisfy 10 ms, 50 ms, or 100 ms objectives.

Internet Deployment Figure 5.6a demonstrates that SCULPTOR offers lower latency paths for more UG*s* during single link failure in realistic routing conditions. On average, SCULPTOR is 7.9 ms higher latency than One-per-Peering, compared to unicast which is 14.2 ms higher than One-per-Peering. PAINTER struggles to find sufficient capacity for UG*s*, overloading 69.6% of traffic.

SCULPTOR provides better resilience than other solutions to site failures on the Internet, yielding no congested traffic and on average 18.1 ms worse than One-per-Peering. The next best solution, unicast, yields 16.1% overloading and, for uncongested traffic, 28.1 ms worse latency than One-per-Peering on average. On average, SCULPTOR is 13.1 ms higher latency than

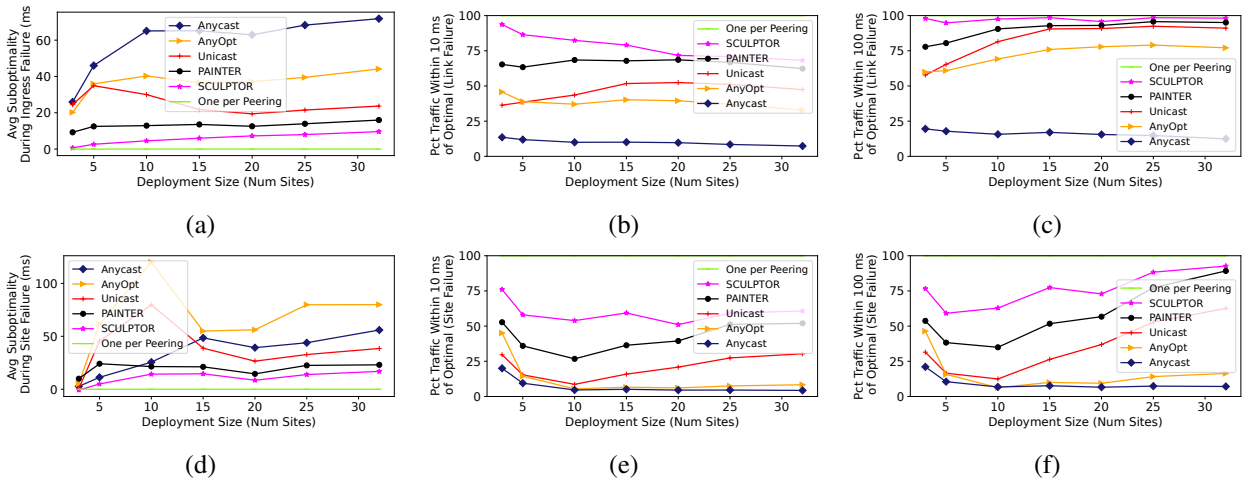


Figure 5.8: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of One-per-Peering latency during both link and site failures over many simulated deployments.

One-per-Peering, while the next best solution, unicast, is 25.1 ms higher. PAINTER again performs poorly, with 95% of traffic overloaded during site failure.

Simulations We show the fraction of traffic within 50 ms of the One-per-Peering solution for link and site failures in Figure 5.6b and Figure 5.6c.

For single-link failures, compared to One-per-Peering, SCULPTOR ranges from 0.7 ms and 9.6 ms worse. SCULPTOR also avoids more overloading, with only 1.3% of traffic being congested on average, while PAINTER (the next-best solution) leads to 3.7% of traffic being congested on average. Single-site failure exhibits similar trends where SCULPTOR is 8.7 ms worse than One-per-Peering's latency and has 18.5% traffic overloaded, on average, while PAINTER leads to 14.7 ms worse latency than One-per-Peering's latency and 34.8% overloading on average.

SCULPTOR also has 78.9% of traffic within 10 ms of One-per-Peering 's latency on average during link failure, 93.3% within 50 ms, and 97.3% within 100 ms. The next-best solution, PAINTER, only has 66.1% within 10 ms, 81.8% within 50 ms, and 89.3% within 100 ms. Site failures show similar trends.

We show additional evaluations of SCULPTOR during link and site failures. Figure 5.14 shows that SCULPTOR provides good resilience to site failures on the Internet, and Figure 5.8 shows similar results for link and site failures in our simulations.

Efficient Infrastructure Utilization

Figure 5.10 shows that installing more capacity to handle peak loads during unseen scenarios is not always necessary with better routing — SCULPTOR finds ways to distribute load over existing infrastructure to accommodate increased demand. We quantify this improved infrastructure utilization under two realistic traffic patterns that SCULPTOR *did not explicitly optimize for* — flash crowds and diurnal effects.

Methodology We define a flash crowd as a transient traffic increase for users in a region. Examples include content releases that spur downloads in a particular region, and localized DDoS attacks. Since increased demand is localized, we can spread excess demand to other sites, which is a cheaper option than installing more capacity (see Section 5.2.2 for our cost model). Links are still provisioned 30% higher than anycast traffic volume as in the rest of Section 5.4.2.

To generate Figure 5.10, we identify each client with a single "region" corresponding to the site at which they have the lowest possible latency ingress link. For each region individually, we then scale each client's traffic in that region by $M\%$ and compute traffic to path allocations. If there are S sites in the deployment, we thus compute S separate allocations per M value where each allocation assumes inflated traffic in exactly one region, but all the allocations are across a single set of routes calculated based on the original (non-flash) traffic. For a target region, we increase M until any link experiences overloading, then find the lowest such M value across regions. For example, if a 60% increase in traffic for Atlanta clients creates overload (and no values $< 60\%$ did for any region), we call $M = 160\%$ the critical value of M.

Our diurnal analysis in Figure 5.10b uses a similar methodology. We define a diurnal effect as a traffic pattern that changes volume according to the time of day. Diurnal effects might be different for different Service Providers, but a prior study from a Service Provider demonstrates that these effects can cause large differences between peak and mean site volume [5]. We sample diurnal patterns from that publication and apply them to our own traffic. We group sites in the same time zone and assign traffic in different time zones different multipliers — in "peak" time zones we assign a multiplier of M and in "trough" time zones a multiplier of $0.1M$. In Figure 5.9 we show

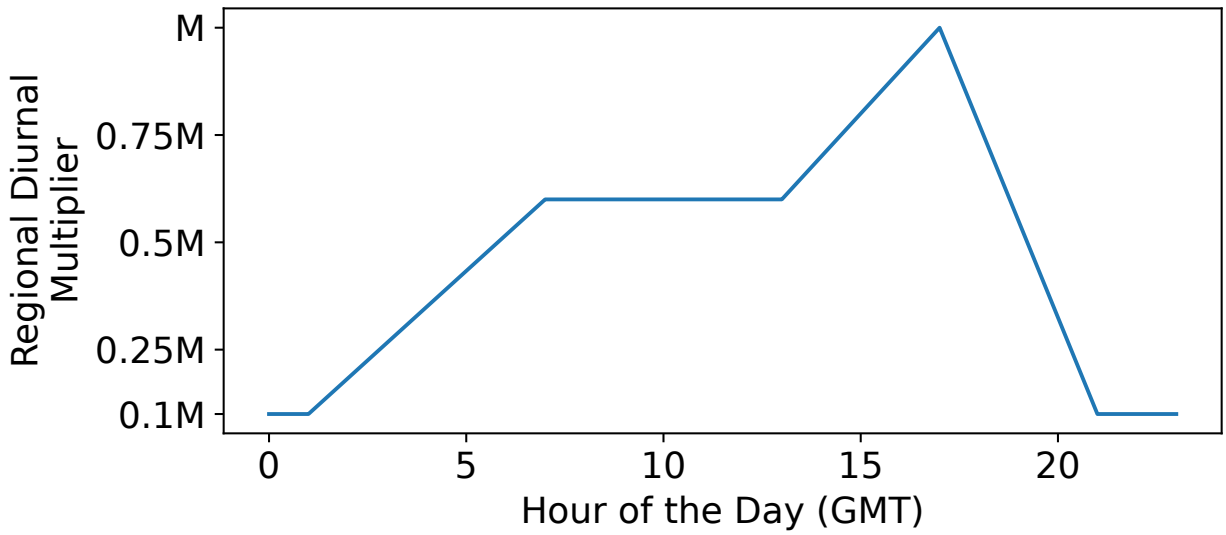


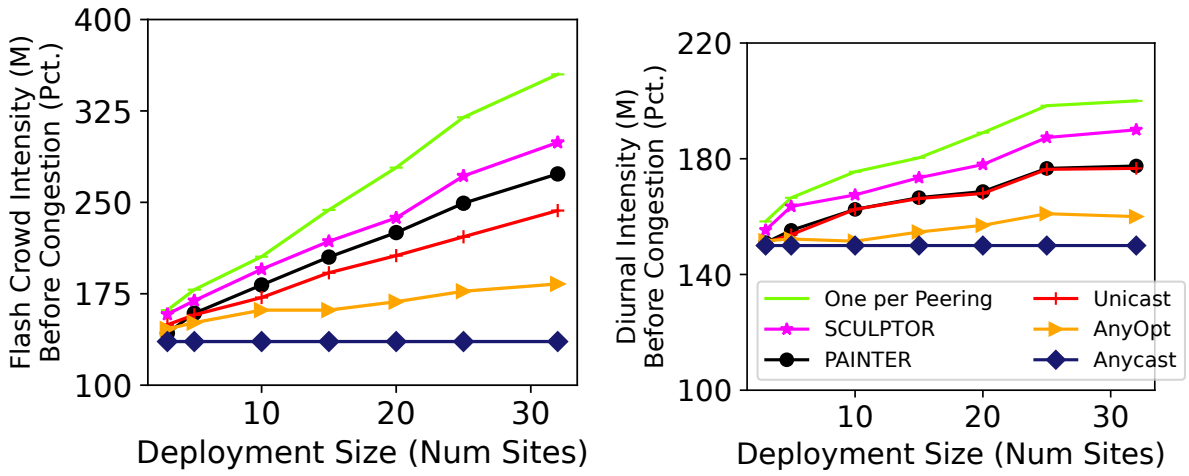
Figure 5.9: The diurnal traffic shape we use to evaluate SCULPTOR.

the shape of our diurnal curve taken from observed diurnal traffic patterns at a Service Provider [5]. Specifically, we linearly interpolate 6 points on the purple curve (Edge Node 1) in Figure 1 of that paper, capturing the essential peaks and troughs. Similar to our flash crowd analysis, we increase M until at least one link experiences overloading at least one hour of the day.

Results Figure 5.10 plots the average over simulations of critical M values that cause overloading for each deployment size under flash crowds and diurnal effects computed using simulated deployments.

Figure 5.10a shows that SCULPTOR finds ways to route more traffic during a flash crowd without overloading than other solutions. For deployments with 32 sites, SCULPTOR can handle flash crowds at 2× more than the expected volume, creating a 3× savings in provisioning costs compared to anycast, and 26% more savings than PAINTER. Figure 5.10b shows that SCULPTOR also handles more intense diurnal traffic swings, allocating traffic to paths without overloading for 24% more intense swings than both PAINTER and unicast with 32 sites. Hence, instead of scaling deployment capacity to accommodate peak time-of-day traffic, Service Providers can re-distribute traffic to sites in off-peak time zones.

Using backup paths does not imply reduced performance as Service Providers can move less latency-sensitive traffic onto these backup paths so as to not impact high-level applications. We



(a) (b)

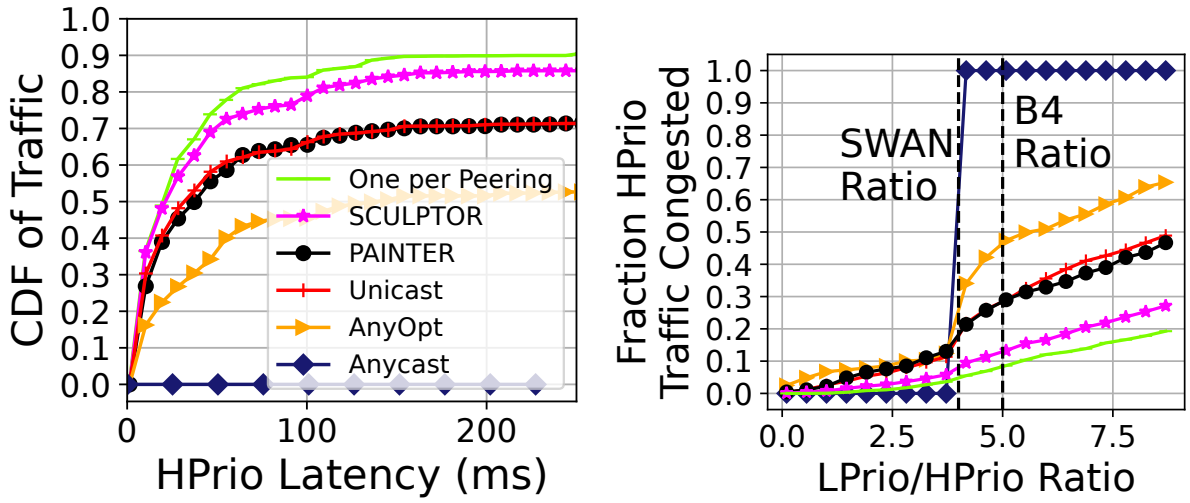
Figure 5.10: SCULPTOR improves infrastructure utilization under flash crowds and diurnal traffic patterns so that Service Providers can underprovision compared to peak loads.

explore this idea in Section 5.4.3.

5.4.3 Handling Multiple Traffic Classes

We also evaluate SCULPTOR's ability to satisfy multiple traffic classes. We split traffic into high and low-priority. The objective is to route high-priority traffic to low-latency routes while limiting congestion on those routes from low-priority traffic. We do not penalize cases where low-priority is congested, but do limit the maximum amount of any traffic on a link to 10× the capacity of the link to avoid solutions where all low-priority is placed on one link, as this solution would lead to low goodput for low-priority traffic (in practice, UG*s* would lower sending rates in response to congestion). We solve SCULPTOR on a single simulated 32 site deployment.

In Figure 5.11b we vary the amount of low-priority traffic as a multiple of high-priority traffic volume and compute the fraction of high-priority traffic congested. Links are provisioned to 5× the high-priority traffic volume (different from Section 5.4.2), as that is roughly the LPrio/HPrio ratio reported in prior work [3, 4]. There is insufficient global capacity to route all traffic for all LPrio/HPrio over 4.0, thus the jump in anycast congestion in Figure 5.11b. SCULPTOR finds strategies that allow us to route more low-priority traffic with less congestion than all other approaches. For example, when LPrio/HPrio = 5, SCULPTOR achieves half as much conges-



(a) (b)

Figure 5.11: SCULPTOR routes high-priority traffic with low latency (5.11a) and minimal congestion from low-priority traffic (5.11b). We inferred LPrio/HPrio ratios for SWAN [3] and B4 [4] from those papers.

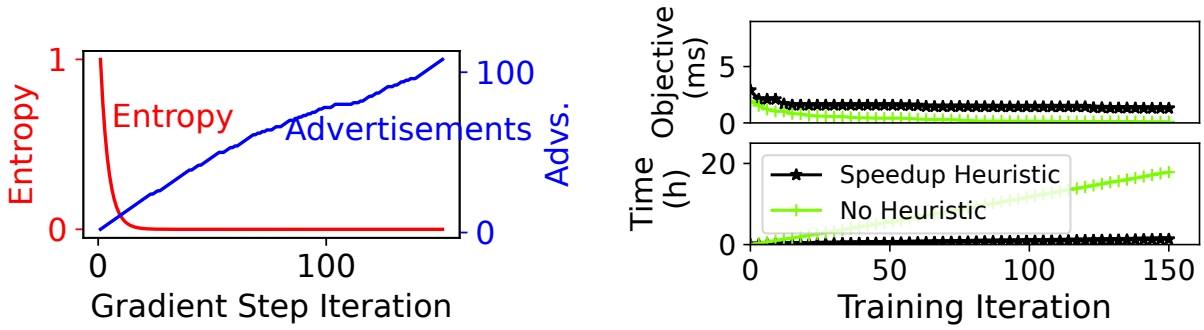
tion as PAINTER and unicast.

Figure 5.11a also shows that SCULPTOR routes traffic with lower latency than other solutions (intercepts at the right of the graph show fractions of high-priority traffic not congested). Figure 5.11a uses LPrio/HPrio = 4, a midpoint between the ratios seen in SWAN [3] and B4 [4].

5.4.4 Why SCULPTOR Works

Comparing More Options First, SCULPTOR compares far more advertisement strategies than the other solutions, so it has potentially better options to choose from. In our 32 site deployments, over 200 gradient steps SCULPTOR estimates latencies in approximately 20M scenarios across every UG. PAINTER only considers thousands (2k), and AnyOpt considers 1k (a configurable number, but the approach would not scale close to the numbers that SCULPTOR tries).

Conducting Fewer Advertisements SCULPTOR only conducts advertisements for exploration (§5.2.3) or for strategies that SCULPTOR thinks will yield good performance (§5.2.4). PAINTER measures advertisements that it thinks might be good, but its greedy approach means that it only considers a single scenario per advertisement iteration. AnyOpt spends time measuring potentially uninformative advertisements. Hence, both AnyOpt and PAINTER conduct a large number of



(a) (b)

Figure 5.12: SCULPTOR's model entropy decreases exponentially with few path measurements (5.12a). Heuristic speedups can also speed computation, but may sacrifice convergence (5.12b).

advertisements on the Internet relative to the configurations they estimate, limiting the number of configurations they can explore. Figure 5.12a shows how the maximum entropy of the distribution of G on unmeasured advertisements (§5.2.3) exponentially decreases as SCULPTOR makes these advertisements on the Internet, and so it quickly grows confident that it does not need to issue more advertisements to find good strategies. This quick convergence manifests since we only have to predict the objective, G, not the paths (§5.2.3).

Horizontal Scaling SCULPTOR converges faster given more compute since gradient computations are parallelizable. PAINTER does not scale horizontally since it uses greedy search (chapter 4). Some objectives additionally admit heuristics for finding approximately optimal traffic allocation solutions (*e.g.,* Section 5.6.5). Other work similarly uses heuristics to quickly solve challenging optimization problems [15, 4, 53]. Figure 5.12b shows that our heuristic (Section 5.6.5) marginally hurts SCULPTOR's convergence (2 ms average latency difference) but requires drastically less compute (50× speedup).

5.5 Related Work

Egress Traffic Engineering Prior work noted that traffic from Service Providers to users sometimes experienced suboptimal performance due to BGP's limitations [9, 8, 10]. SCULPTOR works in tandem with these systems, similarly working with BGP, but to optimize ingress traffic. Other work also shifts traffic to other links/sites to lower cost [61, 15, 5]. SCULPTOR adapts this idea in

a new way but instead uses the public Internet to carry traffic to lower costs.

Ingress Traffic Engineering We compare SCULPTOR to other work on ingress traffic engineering [29, 34, 35, 24, 36, 42] and to PAINTER (chapter 4) both in evaluation (§5.4) and discussion (§5.1.3). FastRoute coarsely balances users across anycast rings to respond to changing conditions [41], giving Service Providers much less control over the specific path users take. PECAN [96] and Tango [6] exhaustively expose paths between endpoints and so may provide resilience to changing conditions if the right paths were exposed. However, exposing all the paths does not scale to our setting since there are too many paths to measure.

Other prior work built a BGP playbook to mitigate DDoS attacks [103], but it is unclear if those strategies would scale to larger Service Providers (they tested on a few sites/providers). SCULPTOR deals with arbitrary unseen conditions over thousands of peering links. Other work and companies create overlay networks and balance load through paths in these overlay networks to satisfy latency requirements [99, 100, 101, 102, 48]. SCULPTOR can work alongside these overlay networks by advertising the external reachability of these nodes in different ways to create better paths through the overlay structure.

Intradomain Failure Planning Service Providers have shown interest in reducing the frequency/impact of failures in their global networks [67, 108, 62, 68]. SCULPTOR works alongside these systems by, for example, enabling Service Providers to shift traffic away from failed components/regions while still retaining good performance. Prior work also shifted traffic during peak times to limit cost/congestion [5], but did so using a private backbone. SCULPTOR's benefits are orthogonal to this prior work and useful for Service Providers without private backbones, as they use the public Internet to realize the same result. Other prior work tried to plan intradomain routes to minimize the impact of k-component failures [63, 64, 65, 66, 5]. SCULPTOR solves different challenges that arise due to the lack of visibility/control into potential paths and their capacities in the interdomain setting.

5.6 Optimization Extensions

In our evaluations we focused on two specific objectives (§5.2.2) It could be, however, that Service Providers wish to optimize other/additional metrics. Here we try to determine which metrics our framework will more likely work for and, in doing so, identify what about SCULPTOR gives good performance. We do not provide any theoretical guarantees but give valuable intuitions.

Our high-level finding is that SCULPTOR works with most metrics we could think of, and will work better both with more prefixes and with richer BGP configurations. However, we also find that considering metrics other than latency could introduce considerable computational costs. SCULPTOR's ability to handle many different traffic conditions likely stems from its consideration of single component failures directly in the loss function.

5.6.1 Gradient Descent Challenges

We propose using gradient descent to solve Equation (5.1) because of its parallelizability which is important given the large problem sizes we consider and the complexity of predicting routes. Recent work suggests that gradient descent may not be important for finding good minimizers [252], so other minimizers may also work well for SCULPTOR.

To use gradient descent to solve Equation (5.1), we require that our objective function is a differentiable function of real variables. Ours, as we have presented it, does not satisfy that condition. Ignoring possible problems with the objective function G, there are two problems: first, our configuration A is a binary variable, and, second, the function R is not well-behaved.

To see what we mean by this second point, consider computing a gradient near a configuration that only advertises one prefix to one provider. A "small" change in the configuration (withdrawing that prefix) would lead to zero reachability and thus no solution, which suggests that R is not "differentiable", or at least not differentiable everywhere. Gradient descent converges to a minimum for functions whose second derivatives are locally bounded [246]. Intuitively, this condition means that a gradient should not push the descent in significantly different directions in small regions

since descent could, for example, oscillate and never decrease the loss.

To solve the first problem, we apply gradient descent to continuous extensions of configurations and the routing function. There are many ways of performing this extension. One way of continuously extending these ideas that we found works well is to treat configuration components as real-valued, rather than binary, variables between 0 and 1 and interpolate objective function values between "adjacent" components of configurations using sigmoids (see Section 5.2 for details). Sigmoids likely work well since their configurable parameter controls steepness, which allows us to control the magnitude of gradients (*i.e.,* not making them too large which could harm convergence).

To solve the second problem, we restrict gradient descent to a region of configurations for which it is unlikely that catastrophic behavior occurs. For example, `anycast` configurations generally still provide (possibly congested) routes for all users when any one link fails, and so we would not expect divergence for any configuration "near" an `anycast` one. In practice, this restriction generally means we pick an initialization where users have paths to many links/sites, and we heavily discourage routeless configurations during gradient descent by assigning these divergent cases a large value.

This solution is reasonable as Service Providers often connect to several transit providers at each site to provide global reachability to all users even under partial congestion or failure [7].

5.6.2 The Key Challenge: Convergence

With bounded second derivatives of the objective function gradient descent converges to *a* minimum [246], and can converge to that minimum quickly with good initializations and choices of the learning rate. However, unless the objective function is convex (ours is not because the routing function is not) then gradient descent may converge to a local minimum. We find that `SCULPTOR` likely converges to a good minimum because of our choice of loss function and that this convergence could improve with more prefixes and richer BGP configuration options.

Nonconvexity in Deep Learning We first draw an analogy to the Deep Learning setting which

has used gradient descent with lots of success, despite its exclusive focus on nonconvex objectives. Deep Learning practitioners identified that we can think of the convergence problem in terms of both the representational power of our models and in terms of the size of our training data set.

Deep Learning tries to find functions f_α parameterized by parameters α that minimize the expectation of a loss function, where the expectation is taken over the distribution of "data". Functions f_α accept this data as input. The loss function is usually chosen so that minimizing the expected value of the loss function over parameters α results in a "useful" thing — evaluating the function f_α with learned parameters α accomplishes a useful task, such as image classification.

In practical settings we only have a corpus of data, rather than the distribution, which serve as samples of that distribution. During gradient descent, deep learning algorithms randomly sample from this corpus and move the parameter gradient roughly along the average of the gradients evaluated at each corpus point [253]. Hence, we often call gradient descent in this setting *stochastic* gradient descent. It is up for debate whether this stochasticity is an important contributor to deep learning's success [254] (or gradient descent at all [252]).

Functions f_α contain potentially billions of parameters and are generally nonconvex. The functions can be compositions of many (tens to hundreds) of functions and so may be "deep". Since these functions contain so many parameters, they can both theoretically and experimentally find a global minimizer of the loss [255], despite nonconvexity. One risk of using too many parameters, however, is "overfitting" — even though a learned function f_α minimizes loss on the corpus of data, it may have high loss on unseen data. This problem is of practical importance since it is often impossible to represent every single function input with a finite data corpus. While it is generally believed that using too many parameters leads to overfitting, recent work suggests that it may relate closer to the geometry of the loss function [252].

Given a task to complete, training a deep learning model to accomplish this task consists of choosing a model, loss function, and optimization method (*e.g.,* gradient descent). To test the efficacy of a training methodology, practitioners mimic applying their learned model to a new setting by splitting their corpus into a *train* and *test* set. Practitioners then optimize using the

training set, and evaluate the performance of their learned model on the test set. If practitioners observe good test performance, they can be more confident that in other settings their algorithms did not overfit and hence "generalize".

Nonconvexity in SCULPTOR In our setting, the parameters α are the configuration and traffic assignment variables, since those are the variables we update with gradient descent. Our data corpus consists of path/deployment metrics (*e.g.,* latency), link capacities, and traffic volumes.

Overfitting & Generalization A potential problem with converging to a local minimum is overfitting. Our deep learning analogy identifies our training data set as both steady-state conditions and conditions under single link and site failures. We explicitly "test" SCULPTOR on flash crowds and diurnal traffic patterns (§5.4.2) and find good generalization. The question is: why does SCULPTOR provide good generalization?

Although the question of why neural networks offer generalization is open, one compelling theory relates to the geometry of the loss landscape. Prior work observes that "wide" loss regions generalize much better than thin/sharp loss regions [256]. One way of interpreting wideness is that perturbations in the parameters, α, do not significantly increase the value of the loss.

One reason why SCULPTOR might provide good generalization is that our choice of loss function encourages finding configurations that are resilient to minor changes. For example, consider a small change in the configuration that withdraws an advertisement of a specific prefix via a specific ingress. Under this change, users will have strictly more routes to the deployment than if all prefixes via that ingress were withdrawn (*i.e.,* a link failure). Hence, the feasible set of all traffic assignments, w, under single-prefix withdrawal is a superset of those assignments under the corresponding link failure, and so the optimal loss value under this small change is less than (*i.e.,* more optimal) than in the link failure setting.

Similarly, small changes in the configuration that *advertise* a prefix via specific ingress can lead to some routes to that prefix no longer being available for some UG*s* if that newly advertised ingress is preferred over others. Hence, such changes are equivalent to withdrawing the advertisement via that ingress for those UG*s*, so our above discussion still applies.

By explicitly minimizing loss under all link failures, SCULPTOR upper bounds the impact of these minor configuration variations, and so (at least intuitively) settles in a "wide" minimum.

Some objectives may also naturally provide wide a minimum in the interdomain path optimization setting we consider. For example, we observed that many UG*s* have similar latencies via many different ingresses to the same site. Hence, small changes in the traffic allocation (which is another one of our "parameters"), α may not affect the overall objective (average latency) that much if those small changes simply shift UG*s* among those different paths to the same site.

Underfitting Another concern with convergence to a local minimum is that the objective function value at that point will be much higher than it could otherwise be at the global minimum. For example, when minimizing average latency it could be that a solution to achieve 35 ms average latency exists, but gradient descent arrives at a local minimum giving an average latency of, for example, 50 ms. In Deep Learning, this problem is called underfitting and can happen when the chosen function does not have sufficient representational complexity [257].

SCULPTOR underfits the training data as shown by the suboptimality compared to the One-per-Peering solution (§5.4.2).

One way of fitting the data better is to consider richer functions with more parameters. In our setting, this option means increasing the number of "parameters" in both advertisement configurations and path options. In other words, this option means increasing the number of prefixes, or types of configurations to offer UG*s* more paths. By this last point we mean, for example, using AS path prepending to potentially offer richer path options. Our investigation in Figure 5.13 confirms this intuition, as using more prefixes allowed us to converge to better solutions whereas using too few prefixes led to similar/worse performing configurations than PAINTER. Other work has used richer configurations to solve problems [96, 6], but adopting such methods to SCULPTOR would require solving scalability challenges.

We Do/Should Not Learn Routing with GD Another type of "learning" we discuss in Section 5.2 is refining our estimate of the routing function *R*. However, this is not descent based learning and is independent of this discussion. In Section 5.2 we model the function *R* as a random

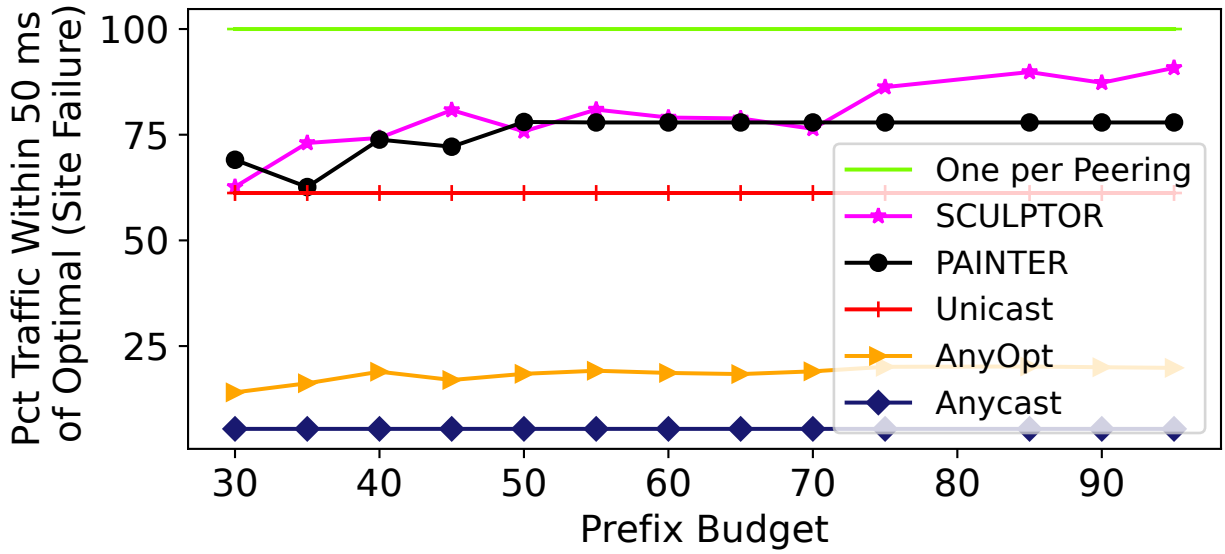


Figure 5.13: Using more prefixes allows SCULPTOR to converge to better solutions.

function, but one that follows a specific structure (preference routing model), and opportunistically removes randomness using Internet measurements. As we demonstrate in Figure 5.12a, the randomness is essentially zero for most gradient descent iterations.

We choose this approach over modeling R as in the deep learning setting since learning R requires advertising configurations in the Internet, which takes a long time. Using the known structure of R allowed for easier debugging than using blackbox deep learning models which is important given the slow iteration process.

5.6.3 So, Which Metrics Work?

The above challenges, solutions, and intuitions are valid independent of the specific metric we care to optimize. Certain metrics could introduce either convergence or computational challenges. Before discussing those challenges, we first present choices of G other than the ones we evaluated (§5.2.2).

Any Function of Latency: Configuring routes that minimize any function of latency are trivial extensions of objectives we considered. The objective function would take the form $G(R(A), w) = \sum_{p,\mathtt{UG}} L(R(A)(p,\mathtt{UG}))w(p,\mathtt{UG})$ where the function L is some arbitrary mapping on latency. The resulting problem is a linear program in w and hence convex in w. A simple variation is

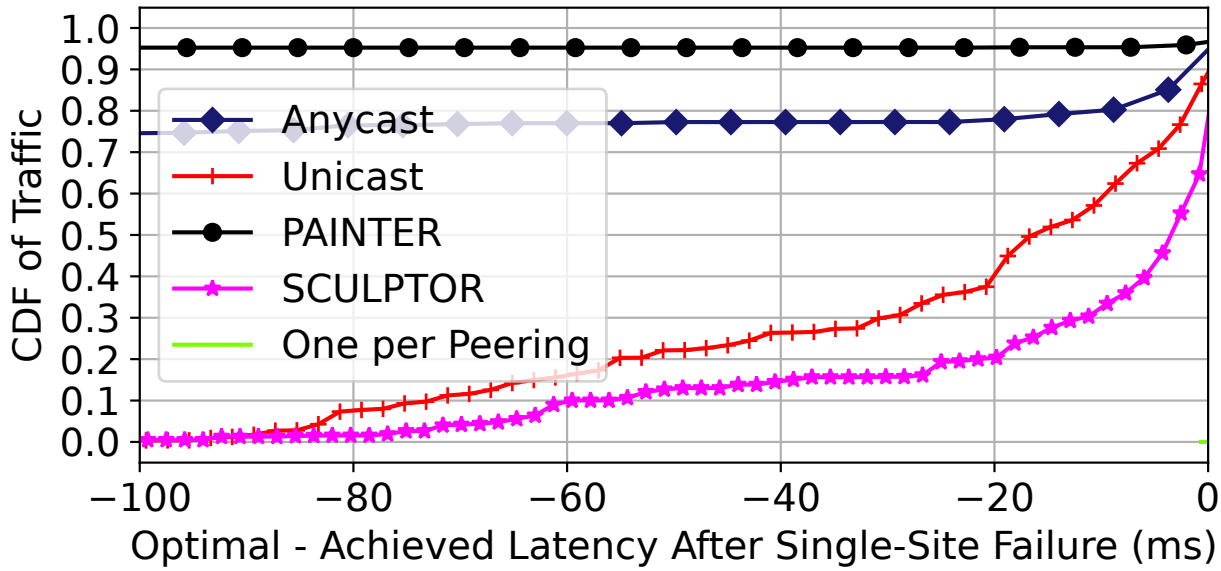


Figure 5.14: Further results on SCULPTOR providing resilience to failure.

maximizing the amount of traffic below a latency threshold, which may be useful for enabling specific applications [1].

Transit Cost: Configuring routes that reduce expected transit cost would roughly amount to minimizing the *maximum* traffic allocation on any provider (possibly weighted by that provider's rate, possibly only considering certain traffic). Hence the objective function would take the form $G(R(A), w) = \max_l \sum_{p,\text{UG}:G(R(A)(\text{UG},p)=l)} w(p, \text{UG})$ which can be expressed as a linear program in w. One could alternatively use methods from prior work in SCULPTOR [15].

Compute/WAN Costs: It could be that sending traffic to a certain site causes increased utilization of WAN bandwidth, compute, or power in a certain region, and that this cost scales non-linearly. Operators can specify custom cost as a function of site traffic. The objective function would take the form $G(R(A), w) = \sum_{p,\text{UG}} C(w(p, \text{UG}))$ where C is some cost function on traffic. Examples include polynomials and exponents which may not be convex in w but can still often be solved with descent-based methods (see the following subsection).

End-to-End Considerations: It could be that, for example, a particular site has low ingress latency for a UG but large egress latency and so may not be a good choice for that UG. As Service Providers have knowledge of these limitations (for example, they can measure egress link utilization or egress latency [8]), they can serve these variables as inputs into SCULPTOR and solve for

path allocations, w subject to those variables.

5.6.4 Convergence Concerns

We expect functions G whose gradients do not explode in the region of interest to converge well. For configurations, A, we encourage gradients to be well-behaved with a good initialization (§5.2.3, §5.6.1). For traffic assignments w, it depends on how the metric, G, incorporates w.

Examples of metrics G of w that offer good convergence include low-order polynomials and exponents (within a reasonable range such that gradients are not too large) whereas bad choices are ones with ill-behaved gradients. As an example, we compared convergence between $G(R(A), w) = \sum_{p,\texttt{UG}} w(p, \texttt{UG})^2$ and $G(R(A), w) = \sum_{p,\texttt{UG}} \sqrt{w(p, \texttt{UG}))}$. The latter has exploding gradients near 0 (which many values of w take on) and so SCULPTOR struggled to find a good solution, while it found a good solution for the first one. (We could not think of why a Service Provider would choose either metric, but they are possible choices.)

5.6.5 Heuristics for Fast Computation

With sufficient resources, solving Equation (5.1) using the two pronged approach outlined in Section 5.2.4 is feasible. Given our advertisement rate limitations (§5.2.3) we wish to compute gradients in tens of minutes. With 100 Monte Carlo simulations per gradient and tens of thousands of gradients to compute, we thus need to solve millions of linear programs at each gradient step. Meta recently demonstrated that they could solve millions of linear programs in minutes using their infrastructure [67].

As we do not have these resources, we instead use efficient heuristics to approximately solve each linear program. Our heuristics depend on the structure of our objective function, and so are specific to our choice of average latency (Eq. (5.2)). During optimization we observed that these heuristics tended to yield accurate estimates for G despite inaccuracies in predicting any individual UG's latency or link's utilization.

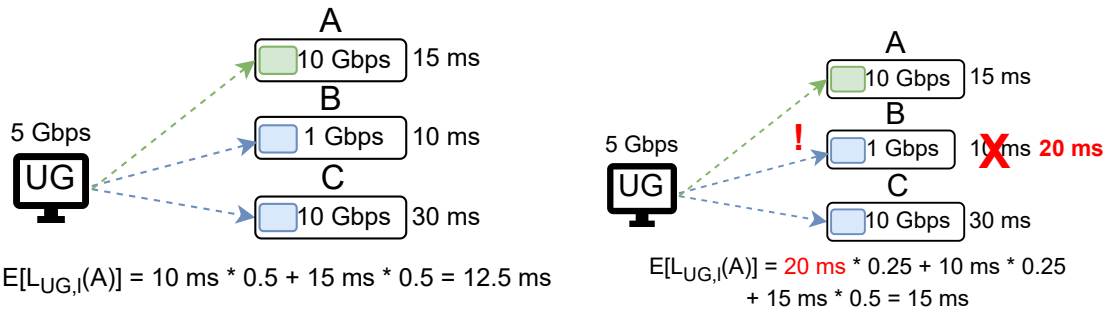
Capacity-Free UG Latency Calculation Minimizers of Equation (5.2) balance traffic across

links to satisfy capacity constraints. We instead temporarily ignore capacity constraints and assign each UG to their lowest-latency path. This approximation solves two problems at once. First, we do not need to solve a linear program to compute traffic assignments. Second, we can now analytically compute the distribution of our objective function, G, over all possible realizations of $R(A)$. Analytically computing this distribution is useful, since it lets us compute entropy (§5.2.3) and expected value (§5.2.3) without Monte Carlo methods.

To see why we can compute the distribution exactly, notice that the objective function is a composition of functions of the form $min(X, Y)$ and $X + Y$ since we choose the minimum latency path across prefixes for each UG and average these minimums. We compute UG latency distributions for each prefix exactly using our routing preference model, and use analytical methods to efficiently compute the distributions of the corresponding minimums and averages.

Imposing Capacity Violation Penalties It could be that such minimum latency assignments lead to congested links. We would like to penalize such advertisement scenarios, and favor those that distribute load more evenly without solving linear programs. Hence, before computing the expected latency, we first compute the probability that links are congested by computing the distribution of link utilizations from the distribution of UG assignments to paths. We then inflate latency for UG*s* on likely overutilized links.

That is, for each possible outcome of UG assignments to links, we compute link utilizations and note the probability those UG*s* reach each link. We then accumulate the probability a link is congested as the total probability over all possible scenarios that lead to overutilization. We discourage UG*s* from choosing paths that are likely congested using a heuristic — we emulate the impact of overloading by inflating latency in this calculation for UG*s* on those paths proportional to the overutilization factor. After emulating the effect of this overloading, we recompute the expected average UG latency *without changing UG decisions*. We do not change UG decisions as doing so could induce an infinite calculation loop if new decisions also lead to overloading. This heuristic penalizes advertisement strategies that would lead to many overutilized links if every user were to choose their lowest latency option. We show an example in Figure 5.15.



(a) (b)

Figure 5.15: A UG has a path to two prefixes, green and blue. The green prefix is only advertised via ingress A and the blue prefix is advertised via two ingresses B and C, each of which are equally likely for this UG. If the UG prefers B over C, we assign the client to the blue prefix (10 ms) while if the UG prefers C over B we will assign the client to the green prefix (15 ms). Hence the initial expected latency is the average of 10 ms and 15 ms corresponding to whether this UG prefers ingress B or C (5.15a). However, ingress B does not have sufficient capacity to handle this UG's traffic and so is congested with 50% probability corresponding to the 50% probability that this user prefers ingress B. We artificially inflate this UG's expected latency (and all other UG*s* using ingress B) to reflect this possible overutilization (5.15b).

5.7 Discussion and Summary

Unilateral control over intradomain traffic has enabled technology that improves performance, reliability, and flexibility, including fast reroute for quick recovery from failure [258], virtual routing and forwarding for flexible traffic engineering [259], and virtual output queueing for differentiated service [223]. Researchers have proposed making similar functionality available in interdomain settings, but, after decades of little adoption, it seems unlikely that such technologies will be widespread enough for Service Provider use. For example, intserv/diffserv [260] and L4S [261] are proposed frameworks for achieving differentiated service but show no signs of deployment. MIRO is a flexible interdomain multipath routing protocol that similarly shows no signs of deployment [17].

Rather than require any changes to the Internet, SCULPTOR uses a simple, black-box model of interdomain routing, BGP's flexibility, and the unused capacity of a Service Provider's global resources to give operators the benefits of those aforementioned technologies in the interdomain setting. For example, SCULPTOR can configure multiple paths per user to enable traffic failover and can set up different routes for different traffic classes to enable virtual output queueing despite

lack of control on the interdomain path. SCULPTOR is a step towards providing Service Providers with programmable interdomain networking primitives so that they can give us the performance that our services increasingly need.

Chapter 6: Conclusion

6.1 Establishing the Thesis

This dissertation supports the following thesis statement: *Existing approaches to interdomain routing suffice for traditional Internet services. To meet the more demanding* `SLO`*s of emerging applications, Service Providers can build systems that use limited Internet protocols in new ways to better meet* `SLO`*s by offering performant, resilient interdomain routes to users, and directing traffic on those routes, at the cost of additional deployment complexity and deployment infrastructure.*

We first thoroughly demonstrate in Chapter 3 that existing approaches to interdomain routing provide low latency at reasonable reliability, which suffices for traditional Internet use cases. While applications such as browsing the web, email, and DASH are somewhat latency-sensitive (< 100 ms), existing approaches such as `anycast` and `unicast` provide these low-latency guarantees for all but a small fraction of users. Critical to providing this latency for Service Providers to have sites physically near users, and to expend engineering effort to make sure users visit close sites. Also critical is caching — much of this content (*e.g.,* video) is static and so placing content close to the user can improve performance even more. If performance changes or sites fail, Service Providers can use DNS to move users to healthier sites even though such changes may take minutes to take effect.

These existing approaches to ingress routing, while sufficient for these non-immersive, cacheable services, do not suffice for applications that we rely on more heavily (*e.g.,* enterprise cloud) or which require more stringent `SLO`*s* like virtual reality. In Chapter 4 we make the argument that Service Providers increasingly need alternate solutions to `anycast` and `unicast` to help meet customer needs. We then present `PAINTER`, a system which offers both lower latency paths than `anycast` and faster redirection among those paths than `unicast`, enhancing resilience.

PAINTER adds significant complexity and investment compared to both anycast and unicast, since it has to manage a system that both manipulates BGP advertisements and places traffic onto those paths.

We noted, however, that PAINTER did not generalize to helping Service Providers meet generic SLO*s* such as satisfying multiple traffic classes or reducing transit costs. We provided an even richer mechanism for Service Providers to meet these guarantees with SCULPTOR, demonstrating that this mechanism could satisfy a much wider class of SLO*s* than alternatives (including PAINTER). This wider class of SLO*s* includes those studied in existing egress traffic engineering systems, and in existing intradomain traffic engineering systems, even though we did not exhaustively characterize the set of SLO*s* that SCULPTOR can meet. SCULPTOR does, however, significantly increase deployment complexity over anycast and unicast, and even over PAINTER.

Both SCULPTOR and PAINTER use existing Internet protocols to achieve these goals by simply offering user networks more routes to choose from. Hence, this dissertation fully supports the thesis.

6.2 Lessons Learned

As our reliance on the Internet has grown and as applications place increasingly strict performance requirements on the network, Service Providers have been tasked with providing increasingly reliable, performant service to more users for more applications. Service Providers need to work with decades-old Internet protocols to do so, and we have shown in this dissertation how to use those protocols to provide the new performance we require.

PAINTER and SCULPTOR both use BGP in nontraditional ways to overcome BGP's shortcomings. Other work has similarly used BGP in nontraditional ways to overcome limitations in different settings. For example Edge Fabric uses BGP in new ways to overcome BGP's key limitations [8]. This flexibility suggests that perhaps it is not solely the protocols that are limited but also our understanding of how best to use them.

This dissertation was not, however, solely focused on improving how Service Providers deliver

services with complex systems. A prevailing theme in this dissertation was that different applications and settings place different requirements on the network (and therefore Service Provider), and the lightest touch to meeting these requirements was generally the best strategy. Our findings on `anycast` and `unicast` performance demonstrate that for many applications `anycast` (or possibly a combination of `anycast` and `unicast`) is an imperfect but sufficient solution to interdomain routing. Both of these strategies are simple to understand and implement and offer good performance. Moreover, their performance can be improved through strategic maneuvering, such as by peering with regional providers to shorten AS paths.

However, `anycast` and `unicast` cannot offer the performance and reliability for certain applications/use cases, and it is in these cases where systems such as `PAINTER` and `SCULPTOR` are useful. These systems both stress the second prevailing theme in this dissertation — the key limitations of current approaches to interdomain routing are (a) Service Providers cannot directly control which path is selected by user networks and (b) Service Providers cannot precisely place traffic on those paths.

But even within these two systems, there are varying degrees of investment a Service Provider can make into optimizing performance and that level of investment will likely depend on how critical meeting those performance requirements is. For example, Service Providers can invest in placing `TM-Edge` nodes in end-user networks to enhance control or just use DNS to direct traffic at a cost of some of the flexibility `TM-Edge` nodes provide. As another example, `SCULPTOR` clearly outperforms `PAINTER` but is significantly more complex to implement and offers less substantial gains compared to what `PAINTER` provides over `anycast`, and so may not make sense to implement for all Service Providers or applications.

We hope that Service Providers keep these key themes in mind to help choose the solution to advertising reachability that works best for them. This dissertation should provide Service Providers with a better understanding of what can go wrong when using each solution, and whether or not something going wrong really matters.

6.3 Future Work

This dissertation provides a step towards removing some of the barriers Service Providers face in meeting the demands of modern applications, but there is still ample room for improvement. We possible future directions that build directly on this dissertation, and then mention other directions only loosely related to this dissertation.

6.3.1 Direct Followup Work

This dissertation characterized and developed methods of advertising reachability from Service Providers in order to improve latency and resilience for networked services, and there are natural extensions to this work.

Our new systems `PAINTER` and `SCULPTOR` advertise reachability in ways that improve latency and resilience for Service Providers. However, it is still unclear how close to an optimal these solutions were. Our closest notion of an optimal way to advertise reachability was an unreasonably expensive solution. Hence, it is unclear whether more work can be done to improve the performance of `SCULPTOR` to satisfy latency or reliability objectives *with fewer prefixes*. Future work could explore theoretical characterizations of the sub-optimality of solutions to advertising reachability in the context of optimizing performance objectives. For example, one could find a set of conditions that guarantees the existence of a solution with fewer prefixes than the `One-per-Peering` solution and with performance near the `One-per-Peering` solution.

This dissertation also only looked at the problem of advertising reachability where advertisements are binary in nature – *i.e.,* whether a prefix was advertised or not to a certain peer/provider. It could be that tagging advertisements with BGP communities, poisoning advertisements, or prepending advertisements could yield more efficient, performant solutions than either `PAINTER` or `SCULPTOR`. For example, advertising a single prefix to a single peer/provider may expose a different path than `anycast`. However, one could uncover even more paths by advertising prefixes that poison ASes along that path [6]. Having more paths generally means having better perfor-

mance, so using these methods could yield more performant solutions.

This dissertation also noted that a challenge in advertising more paths was that IPv4 prefixes incur a significant cost to use. This cost is not only monetary — polluting the BGP routing tables with many prefix advertisements whose sole purpose is traffic engineering could be seen as harmful. As more networks (of all types, not just Service Providers) realize the benefits of advertising more prefixes, this hypothetical problem of polluting BGP routing tables could quickly become tangible for thousands of networks around the world that operate routers at their memory limits. The problem would only be exacerbated by IPv6, where prefixes are much cheaper monetarily but take up 8× the amount of memory in routers. Finding more scalable solutions to either interdomain traffic engineering or to interdomain packet switching would be interesting areas for future research.

Finally, this dissertation used a simple model of congestion of Internet paths. Paths from users to the Service Provider were distinct edges in a graph and had capacities corresponding to the capacity of the Service Provider's ingress link. In practice this model may not be valid — paths from many user networks to a Service Provider may overlap, and the ingress link may not be the bottleneck of the path. Furthermore, capacity along paths may be affected not only by traffic from user networks towards the Service Provider but also by cross-traffic unrelated to the Service Provider. This problem is challenging to investigate since bottlenecks might occur in unobservable areas (other networks), and since it is hard to study bottleneck link effects in a research setting since finding bottlenecks necessarily requires inundating networks, which is expensive and invasive. Investigating the impact of these dynamics and developing models that capture such dynamics is an interesting future direction to explore.

6.3.2 Broad Further Directions

This dissertation tries to find better ways for Service Providers to deliver networked services from an interdomain perspective. There are several research directions towards that goal that this dissertation did not explicitly focus on.

Optimization of Deployment Build-Out

This dissertation assumed the deployment was fixed in terms of both where infrastructure was located and what network options were available at each location. For example, we did not consider peering with other networks or switching from one provider to another to improve performance or reduce costs. Future work could cast the entire process of deployment build-out as an optimization problem: site placement, datacenter placement, private WAN buildout, who to peer with, and where to place caches/edge compute are all dimensions on which we can explore the problem.

This problem is interdisciplinary, as infrastructure placement is clearly an economic, as well as networking, problem. This problem is also challenging since we cannot rely purely on historical data to make such decisions. For example, edge computing is a new capability, and so it is difficult to predict how much such infrastructure costs and how such infrastructure could impact Service Provider objectives. Placement and performance are interrelated, as component placement incurs costs but could help performance for local users. Energy cost and availability is an increasingly relevant metric in data center placement in particular, as modern applications like cryptocurrency mining and machine-learning training demands grow.

A related problem is the inverse, deployment scale-down, which is equally relevant as the applications Service Providers serve evolve. For example, with the increased proliferation of dynamic content (*e.g.,* social media, AI models) it becomes harder to cache responses to user queries near the edge. Hence, it might make sense to scale down caching services and instead scale up a different type of edge service.

A unique challenge to consider in this domain is that all these problems are parts of the same, higher-level problem of optimizing costs and performance to best meet Service Provider objectives. The problems may have strong correlations (for example adding sites and removing caches) and so could benefit from joint solutions, and so finding scalable ways to compute such joint solutions is an important direction to consider.

Service Provider Deployments as a Service

Service Providers have built out their deployments to scale to global user bases which is great for Service Providers and their users but poses a possible threat to competition as new entrants face competitors with global resources and expert talent [262]. Luckily, many Service Providers increasingly offer their infrastructure as a service, including compute, networking functions, physical connectivity, and caching. Some Service Providers, like Vultr, even offer network control-plane functions such as BGP advertisements to their customers. These functionalities offer opportunity for a new deployment model and service offering — Service Providers as a service.

Consider the problem of launching an Internet service (or set of services) with an evolving user base and possibly evolving service requirements, and trying to find the best infrastructure/system to use to meet those objectives. Solving this problem could involve finding the set of Service Providers who offer required services (*e.g.,* BGP controllability, virtual networking) in desired locations at the best cost, subject to current service demands [263]. Challenges in solving this problem involve handling logistic deployment complexity (for example, handling several deployments in several clouds simultaneously and ensuring compatibility) and balancing objectives of achieving low-cost but also retaining reliable, performant service.

A related idea is the possibility of Service Providers selectively offering tenants traffic engineering services, like the ones presented and referenced in this dissertation (*e.g.,* B4 [4], Edge Fabric [8], `PAINTER` (chapter 4), `SCULPTOR`). Each of these systems can treat some traffic with less priority than others. For example, Edge Fabric can place lower priority traffic onto higher latency links to free room on lower latency links for higher priority traffic. Service Providers can offer tenants the choice between acting as a high priority tenant which benefits both parties — Service Providers have a direct signal of which traffic is more important and so can better optimize how they deliver traffic, and tenants who do not care about small performance differences can lower their costs. Some Service Providers already offer similar services, such as Google's premium versus standard networking option [131], but it is unclear if they currently optimize such offerings from a profit/performance perspective.

6.4 Closing Thoughts

One of the Internet's greatest strengths and weaknesses is its distributed nature. Centralization slows innovation and reduces scalability, and the Internet's distributed design philosophy has clearly allowed networked applications to proliferate rapidly. Distributed systems, however, tend to have inefficiencies of their own such as complexity and possibly reduced performance. The degree to which we integrate the Internet with our lives and the level of immersion in networked applications we now desire make the inefficiencies that come with the distributed Internet system more of a problem. Some networked applications require 10 ms latency with high reliability, which pushes the limits of what we can physically accomplish given the speed of light. Processors will not get much faster [264], while application compute demands continue to skyrocket (consider LLMs), suggesting a move towards more computation required by the network. Hence, as much as we feel these problems today, it could be that they will be even *more* important in a few years. This dissertation can be seen as part of the resolution of the goal of increased need for centralized networked applications in our lives with the reality of a distributed Internet. We hope this dissertation spurs more work that will continue this necessary resolution and ultimately give networked applications the underlying service they need to interconnect our world.

References

[1] N. Mohan, L. Corneo, A. Zavodovski, S. Bayhan, W. Wong, and J. Kangasharju, "Pruning Edge Research With Latency Shears," in *HotNets 2020*.

[2] Microsoft, *Microsoft Datacenters*, 2023.

[3] C.-Y. Hong *et al.*, "Achieving High Utilization with Software-Driven WAN," in *SIGCOMM 2013*.

[4] S. Jain *et al.*, "B4: Experience with a Globally-Deployed Software Defined WAN," in *SIGCOMM 2013*.

[5] D. Chou *et al.*, "Taiji: Managing Global User Traffic for Large-Scale Internet Services at the Edge," in *SOSP 2019*.

[6] H. Birge-Lee, M. Apostolaki, and J. Rexford, "It Takes Two to Tango: Cooperative Edge-to-Edge Routing," in *HotNets 2022*.

[7] T. Arnold *et al.*, "Cloud Provider Connectivity in the Flat Internet," in *IMC 2020*.

[8] B. Schlinker *et al.*, "Engineering Egress with Edge Fabric: Steering Oceans of Content to the World," in *SIGCOMM 2017*.

[9] K.-K. Yap *et al.*, "Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering," in *SIGCOMM 2017*.

[10] R. Landa, L. Saino, L. Buytenhek, and J. T. Araújo, "Staying Alive: Connection Path Reselection at the Edge," in *NSDI 2021*.

[11] Y.-C. Chiu, B. Schlinker, A. B. Radhakrishnan, E. Katz-Bassett, and R. Govindan, "Are We One Hop Away from a Better Internet?" In *IMC 2015*.

[12] T. Koch, E. Katz-Bassett, J. Heidemann, M. Calder, C. Ardi, and K. Li, "Anycast in Context: A Tale of Two Systems," in *SIGCOMM 2021*.

[13] Microsoft, *Microsoft Global Network*, 2023.

[14] T. Takami, *Project Myriagon: Cloudflare Passes 10,000 Connected Networks*, 2021.

[15] R. Singh, S. Agarwal, M. Calder, and P. Bahl, "Cost-Effective Cloud Edge Traffic Engineering with Cascara," in *NSDI 2021*.

[16] C. Krähenbühl *et al.*, "Deployment and Scalability of an Inter-Domain Multi-Path Routing Infrastructure," in *CoNEXT 2021*.

[17] W. Xu and J. Rexford, "MIRO: Multi-Path Interdomain Routing," in *SIGCOMM 2006*.

[18] E. Pujol, I. Poese, J. Zerwas, G. Smaragdakis, and A. Feldmann, "Steering Hyper-Giants' Traffic at Scale," in *CoNEXT 2019*.

[19] D. D. Clark, "The Design Philosophy of the DARPA Internet Protocols," *SIGCOMM Computer Communication Review*, 1995.

[20] N. Spring, R. Mahajan, and T. Anderson, "The Causes of Path Inflation," in *Applications, technologies, architectures, and protocols for computer communications 2003*.

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

[22] Í. Cunha *et al.*, "Sibyl: A Practical Internet Route Oracle," in *NSDI 2016*.

[23] R. Beverly and M. Allman, "An Internet Heartbeat," *arXiv preprint arXiv:1901.10441*, 2019.

[24] J. Zhu, K. Vermeulen, I. Cunha, E. Katz-Bassett, and M. Calder, "The Best of Both Worlds: High Availability CDN Routing Without Compromising Control," in *IMC 2022*.

[25] T. Koch, S. Yu, E. Katz-Bassett, R. Beckett, and S. Agarwal, "PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks," in *SIGCOMM 2023*.

[26] S. Sarat, V. Pappas, and A. Terzis, "On the Use of Anycast in DNS," in *SIGMETRICS 2006*.

[27] Z. Li, D. Levin, N. Spring, and B. Bhattacharjee, "Internet Anycast: Performance, Problems, & Potential," in *SIGCOMM 2018*.

[28] S. McQuistin, S. P. Uppu, and M. Flores, "Taming Anycast in the Wild Internet," in *IMC 2019*.

[29] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *IMC 2015*.

[30] Z. Li, "Diagnosing and Improving the Performance of Internet Anycast," Ph.D. dissertation, University of Maryland, College Park, 2019.

[31] R. Krishnan *et al.*, "Moving Beyond End-to-End Path Information to Optimize CDN Performance," in *IMC 2009*.

[32] F. Chen, R. K. Sitaraman, and M. Torres, "End-User Mapping: Next Generation Request Routing for Content Delivery," in *SIGCOMM 2015*.

[33] A. Flavel, P. Mani, and D. A. Maltz, "Re-Evaluating the Responsiveness of DNS-Based Network Control," in *LANMAN 2014*.

[34] H. H. Liu *et al.*, "Efficiently Delivering Online Services over Integrated Infrastructure," in *NSDI 2016*.

[35] X. Zhang *et al.*, "AnyOpt: Predicting and Optimizing IP Anycast Performance," in *SIGCOMM 2021*.

[36] M. Zhou *et al.*, "Regional IP Anycast: Deployments, Performance, and Potentials," in *SIGCOMM 2023*.

[37] J. Xue, W. Dang, H. Wang, J. Wang, and H. Wang, "Evaluating Performance and Inefficient Routing of an Anycast CDN," in *Proceedings of the International Symposium on Quality of Service*, 2019.

[38] B. Schlinker, T. Arnold, I. Cunha, and E. Katz-Bassett, "PEERING: Virtualizing BGP at the Edge for Research," in *CoNEXT 2019*.

[39] B. Kataria *et al.*, "Saving Private WAN: Using Internet Paths to Offload WAN Traffic in Conferencing Services," in *CoNEXT 2024*.

[40] J.-P. Stauffert, F. Niebling, and M. E. Latoschik, "Effects of Latency Jitter on Simulator Sickness in a Search Task," in *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*, 2018.

[41] A. Flavel *et al.*, "FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs," in *NSDI 2015*.

[42] M. Markovitch *et al.*, "TIPSY: Predicting Where Traffic Will Ingress a WAN," in *SIGCOMM 2022*.

[43] G. C. M. Moura *et al.*, "Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event," in *IMC 2016*.

[44] M. Wichtlhuber *et al.*, "IXP Scrubber: Learning from Blackholing Traffic for ML-driven DDoS Detection at Scale," in *Proceedings of the ACM SIGCOMM 2022 Conference*, 2022, pp. 707–722.

[45] D. Riley, *Internet's Rapid Growth Faces Challenges Amid Rising Denial-of-Service Attacks*, 2023.

[46] M. Jackson, *Record Internet Traffic Surge Seen by UK ISPs on Tuesday*, 2020.

[47] M. Landi, *Return of Original Fortnite Map Causes Record Traffic on Virgin Media O2 Network*, 2023.

[48] J. Li *et al.*, "Livenet: A Low-Latency Video Transport Network for Large-scale Live Streaming," in *SIGCOMM 2022*.

[49] S. Moritz, *Internet Traffic Surge Triggers Massive Outage on East Coast*, 2021.

[50] D. Madory, *GP Leak Leads to Spike of Misdirected Traffic*, 2024.

[51] A. Griffin, *Facebook, Instagram, Messenger down: Meta Platforms Suddenly Stop Working in Huge Outage*, 2024.

[52] D. Madory, *Outage Notice From Microsoft*, 2024.

[53] Y. Perry *et al.*, "DOTE: Rethinking (Predictive) WAN Traffic Engineering," in *NSDI 2023*.

[54] G. C. Moura, J. Heidemann, M. Müller, R. de O. Schmidt, and M. Davids, "When the Dike Breaks: Dissecting DNS Defenses During DDoS," in *IMC 2018*.

[55] M. Prince, *The DDoS That Knocked Spamhaus Offline (And How We Mitigated It)*, 2013.

[56] S. Kottler, *February 28th DDoS Incident Report*, 2018.

[57] C. Miao *et al.*, "MegaTE: Extending WAN Traffic Engineering to Millions of Endpoints in Virtualized Cloud," in *SIGCOMM 2024*.

[58] S. S. Ahuja *et al.*, "Capacity-Efficient and Uncertainty-Resilient Backbone Network Planning with HOSE," in *SIGCOMM 2021*.

[59] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "RADWAN: Rate Adaptive Wide Area Network," in *SIGCOMM 2018*.

[60] C. Miao *et al.*, "FlexWAN: Software Hardware Co-design for Cost-Effective and Resilient Optical Backbones," in *SIGCOMM 2023*.

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

[62] U. Krishnaswamy, R. Singh, N. Bjørner, and H. Raj, "Decentralized Cloud Wide-Area Network Traffic Engineering with BLASTSHIELD," in *NSDI 2022*.

[63] Y. Wang *et al.*, "R3: Resilient Routing Reconfiguration," in *SIGCOMM 2010*.

[64] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic Engineering with Forward Fault Correction," in *SIGCOMM 2014*.

[65] Y. Chang, C. Jiang, A. Chandra, S. Rao, and M. Tawarmalani, "Lancet: Better Network Resilience by Designing for Pruned Failure Sets," 2019.

[66] C. Jiang, S. Rao, and M. Tawarmalani, "PCF: Provably Resilient Flexible Routing," in *SIGCOMM 2020*.

[67] J. P. Eason *et al.*, "Hose-based Cross-Layer Backbone Network Design with Benders Decomposition," in *SIGCOMM 2023*.

[68] B. Liu *et al.*, "CAPA: An Architecture For Operating Cluster Networks With High Availability," in *NSDI 2024*.

[69] J. C. Mogul, R. Isaacs, and B. Welch, "Thinking about Availability in Large Service Infrastructures," in *HotOS 2017*.

[70] IETF, *A Border Gateway Protocol (BGP-4)*, 2006.

[71] M. Calder, X. Fan, and L. Zhu, "A Cloud Provider's View of EDNS Client-Subnet Adoption," in *TMA 2019*.

[72] R. N. Staff, "RIPE Atlas: A Global Internet Measurement Network," *Internet Protocol Journal*, 2015.

[73] VULTR, *VULTR Cloud*, 2023.

[74] M. Calder *et al.*, "Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System," in *NSDI 2018*.

[75] H. Ballani, P. Francis, and S. Ratnasamy, "A Measurement-Based Deployment Proposal for IP Anycast," in *SIGCOMM 2006*.

[76] J. Liang, J. Jiang, H. Duan, K. Li, and J. Wu, "Measuring Query Latency of Top Level DNS Servers," in *International Conference on Passive and Active Network Measurement (PAM)*, Hong Kong: Springer, Mar. 2013.

[77] R. de Oliveira Schmidt, J. Heidemann, and J. H. Kuipers, "Anycast Latency: How Many Sites are Enough?" In *PAM 2017*.

[78] H. Ballani and P. Francis, "Towards a Global IP Anycast Service," in *Proceedings of the 2005 ACM SIGCOMM Conference*, Philadelphia, PA, USA: ACM, Aug. 2005.

[79] D. Giordano *et al.*, "A First Characterization of Anycast Traffic from Passive Traces," in *TMA 2016*.

[80] D. Cicalese, J. Augé, D. Joumblatt, T. Friedman, and D. Rossi, "Characterizing IPv4 Anycast Adoption and Deployment," in *CoNEXT 2015*.

[81] Verizon, *Edgecast*, 2020.

[82] M. Prince, *Load Balancing without Load Balancers*, 2013.

[83] L. Wei, M. Flores, H. Bedi, and J. Heidemann, "Bidirectional Anycast/Unicast Probing (BAUP): Optimizing CDN Anycast," in *NOMS 2020*.

[84] Y. Jin *et al.*, "Zooming In On Wide-area Latencies to a Global Cloud Provider," in *SIGCOMM 2019*.

[85] M. Fayed *et al.*, "The Ties that un-Bind: Decoupling IP from Web Services and Sockets for Robust Addressing Agility at CDN-Scale," in *SIGCOMM 2021*.

[86] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS Performance and the Effectiveness of Caching," *IEEE/ACM Transactions on networking*, 2002.

[87] T. Callahan, M. Allman, and M. Rabinovich, "On Modern DNS Behavior and Properties," *ACM SIGCOMM Computer Communication Review*, 2013.

[88] Y. Yu, D. Wessels, M. Larson, and L. Zhang, "Authority Server Selection in DNS Caching Resolvers," *ACM SIGCOMM Computer Communication Review*, 2012.

[89] M. Lentz, D. Levin, J. Castonguay, N. Spring, and B. Bhattacharjee, "D-mystifying the D-root Address Change," in *IMC 2013*.

[90] M. Thomas, *Chromium's Impact on Root DNS Traffic*, 2020.

[91] W. Hardaker, *What's in a Name? 2020.*

[92] S. Sundaresan, N. Magharei, N. Feamster, R. Teixeira, and S. Crawford, "Web Performance Bottlenecks in Broadband Access Networks," in *SIGMETRICS 2013*, APNIC, 2020.

[93] A. S. Asrese, P. Sarolahti, M. Boye, and J. Ott, "WePR: A Tool for Automated Web Performance Measurement," in *2016 IEEE Globecom Workshops*, APNIC, 2020.

[94] Internet Archive, *The HTTP Archive Project*, 2024.

[95] M. Allman, "Putting DNS in Context," in *IMC 2020*, APNIC, 2020.

[96] V. Valancius, B. Ravi, N. Feamster, and A. C. Snoeren, "Quantifying the Benefits of Joint Content and Network Routing," in *SIGMETRICS 2013*, APNIC, 2020.

[97] Y. Zhang, J. Tourrilhes, Z.-L. Zhang, and P. Sharma, "Improving SD-WAN Resilience: From Vertical Handoff to WAN-Aware MPTCP," *ToN*, 2021.

[98] P. Sun, L. Vanbever, and J. Rexford, "Scalable Programmable Inbound Traffic Engineering," in *SOSR 2015*, APNIC, 2020.

[99] Subspace, *Optimize Your Network on Subspace*, 2022.

[100] INAP, *INAP Network Connectivity*, 2022.

[101] Akamai, *Akamai Secure Access Service Edge*, 2022.

[102] Cloudflare, *Argo Smart Routing*, 2022.

[103] A. Rizvi, L. Bertholdo, J. Ceron, and J. Heidemann, "Anycast Agility: Network Playbooks to Fight DDoS," in *USENIX Security Symposium 2022*, APNIC, 2020.

[104] S. Burnett *et al.*, "Network Error Logging: Client-Side Measurement of End-to-End Web Service Reliability," in *NSDI 2020*, APNIC, 2020.

[105] Z. Zheng *et al.*, "Xlink: QoE-Driven Multi-Path QUIC Transport in Large-Scale Video Services," in *SIGCOMM 2021*, APNIC, 2020.

[106] W. Sentosa, B. Chandrasekaran, P. B. Godfrey, H. Hassanieh, and B. Maggs, "DCHANNEL: Accelerating Mobile Applications With Parallel High-bandwidth and Low-latency Channels," in *NSDI 2023*, APNIC, 2020.

[107] H. Chen *et al.*, "T-gaming: A Cost-Efficient Cloud Gaming System at Scale," *IEEE Transactions on Parallel and Distributed Systems*, 2019.

[108] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or Die: High-Availability Design Principles Drawn from Google's Network Infrastructure," in *SIGCOMM 2016*, APNIC, 2020.

[109] IANA, *Root servers*, 2020.

[110] Akamai, *Designing DNS for Availability and Resilience Against DDoS Attacks*, 2020.

[111] Amazon, *Amazon Route 53 FAQs*, 2020.

[112] Google, *Google Public DNS*, 2020.

[113] D. Katabi and J. Wroclawski, "A Framework for Global IP-Anycast (GIA)," in *SIGCOMM 2000*, APNIC, 2020.

[114] C. Metz, "IP Anycast Point-To-(Any) Point Communication," *IEEE Internet Computing*, 2002.

[115] C. Partridge, T. Mendez, and W. Milliken, *Host Anycasting Service*, 1993.

[116] DNS-OARC, *A Day in the Life of the Internet*, 2018.

[117] L. Colitti, E. Romijn, H. Uijterwaal, and A. Robachevsky, "Evaluating the Effects of Anycast on DNS Root Name Servers," *RIPE Document RIPE-393*, 2006.

[118] IETF, *Happy eyeballs version 2: Better connectivity using concurrency*, 2017.

[119] Cloudflare, *What is DNS? 2020.*

[120] P. Mockapetris, *Domain Names - Implementation and Specification*, 1987.

[121] A. Akplogan *et al.*, "Analysis of the Effects of COVID-19-Related Lockdowns on IMRS Traffic," 2020.

[122] H. Gao *et al.*, "Reexamining DNS from a Global Recursive Resolver Perspective," *IEEE/ACM Transactions on Networking*, 2014.

[123] IANA, *IANA IPv4 Special-Purpose Address Registry*, 2020.

[124] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang, "A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers," in *NSDI 2002*, APNIC, 2020.

[125] OpenDNS, *OpenDNS Data Center Locations*, 2020.

[126] M. Gharaibeh, H. Zhang, C. Papadopoulos, and J. Heidemann, "Assessing Co-locality of IP Blocks," in *Proceedings of* 19^{th} *IEEE Global Internet Symposium*, APNIC, 2020.

[127] G. Huston, *How Big is that Network? 2014.*

[128] T. Cymru, *IP to ASN Mapping Service*, 2020.

[129] Akamai, *Akamai Compliance Programs*, 2021.

[130] M. Müller, G. C. M. Moura, R. de Oliveira Schmidt, and J. Heidemann, "Recursives in the Wild: Engineering Authoritative DNS Servers," in *IMC 2017*, APNIC, 2014.

[131] T. Arnold *et al.*, "(How Much) Does a Private WAN Improve Cloud Performance?" In *INFOCOM 2020*, APNIC, 2014.

[132] G. C. M. Moura, J. Heidemann, W. Hardaker, J. Bulten, J. Ceron, and C. Hesselman, "Old But Gold: Prospecting TCP to Engineer DNS Anycast (extended)," *ISI-TR-740, USC/Information Sciences Institute, Tech. Report*, 2020.

[133] Maxmind, *GeoIP2 Databases*, 2022.

[134] E. Katz-Bassett, J. P. John, A. Krishnamurthy, D. Wetherall, T. Anderson, and Y. Chawathe, "Towards IP Geolocation Using Delay and Topology Measurements," in *IMC 2006*, APNIC, 2014.

[135] M. Allman, "On Eliminating Root Nameservers from the DNS," in *HOTNETS 2019*, APNIC, 2014.

[136] W. Kumari and P. Hoffman, "Running a Root Server Local to a Resolver," Tech. Rep., 2014.

[137] R. NCC, *Evaluating The Effects Of Anycast On DNS Root Nameservers*, 2006.

[138] M. Akcin, *Comparing Root Server Performance Around the World*, 2015.

[139] R. Bellis, *Researching F-root Anycast Placement Using RIPE Atlas*, 2015.

[140] Y. Xu, *2017 Update: Comparing Root Server Performance Globally*, 2014.

[141] J. Heidemann, K. Obraczka, and J. Touch, "Modelling the Performance of HTTP Over Several Transport Protocols," *ACM/IEEE Transactions on Networking*, 1997.

[142] N. Cardwell, S. Savage, and T. Anderson, "Modelling TCP Latency," in *INFOCOM 2000*, APNIC, 2014.

[143] J. Rüth, C. Bormann, and O. Hohlfeld, "Large-Scale Scanning of TCP's Initial Window," in *IMC 2017*, APNIC, 2014.

[144] G. Combs, *TShark*, 2020.

[145] Mozilla, *Window: Load Event*, 2020.

[146] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan, "LatLong: Diagnosing Wide-Area Latency Changes for CDNs," *IEEE Transactions on Network and Service Management*, 2012.

[147] F. Wohlfart, N. Chatzis, C. Dabanoglu, G. Carle, and W. Willinger, "Leveraging Interconnections for Performance: The Serving Infrastructure of a Large CDN," in *SIGCOMM 2018*, APNIC, 2014.

[148] CAIDA, *Inferred AS to Organization Mapping Dataset*, 2020.

[149] ICANN, *Packet Clearing House*, 2020.

[150] T. Arnold *et al.*, *FAQ on RIRS Node Hosting*, 2014.

[151] R. NCC, *Hosting a K-root Node*, 2018.

[152] T. Arnold *et al.*, "Beating BGP is Harder than we Thought," in *HOTNETS 2019*, APNIC, 2014.

[153] Google, *Cloud Load Balancing*, 2021.

[154] H. Kim and A. Gupta, "ONTAS: Flexible and Scalable Online Network Traffic Anonymization System," in *Workshop on Network Meets AI & ML 2019*, APNIC, 2014.

[155] C. Satten, *Lossless Gigabit Remote Packet Capture with Linux*, 2008.

[156] J. Xu, J. Fan, M. H. Ammar, and S. B. Moon, "Prefix-Preserving IP Address Anonymization: Measurement-Based Security Evaluation and a New Cryptography-Based Scheme," in *ICNP 2002*, APNIC, 2014.

[157] G. C. M. Moura, J. Heidemann, R. d. O. Schmidt, and W. Hardaker, "Cache Me If You Can: Effects of DNS Time-to-Live," in *IMC 2019*, APNIC, 2014.

[158] P. Bhowmick, M. I. A. Khan, C. Deccio, and T. Chung, "TTL Violation of DNS Resolvers in the Wild," in *PAM 2023*, APNIC, 2014.

[159] L. Wei and J. Heidemann, "Does Anycast Hang Up on You?" In *TMA 2017*, APNIC, 2014.

[160] GTmetrix, *The Top 1,000 Sites on the Internet*, 2019.

[161] M. Intelligence, *Network as a Service Market - Growth, Trends, COVID-19 Impact, and Forecasts*, 2022.

[162] 3GPP, *System Architecture for the 5G System (5GS)*, 2020.

[163] L. Peterson and O. Sunay, "5G Mobile Networks: A Systems Approach," *Synthesis Lectures on Network Systems*, 2020.

[164] C. Labovitz, A. Ahuja, A. Bose, and F. Jahanian, "Delayed Internet Routing Convergence," *ToN*, 2001.

[165] Tessares, *Introducing a Client Library for 0-RTT Converter*, 2019.

[166] Microsoft, *SD-WAN Connectivity Architecture with Azure Virtual WAN*, 2022.

[167] Amazon, *Simplify SD-WAN Connectivity With AWS Transit Gateway Connect*, 2022.

[168] Google, *Network Connectivity Center*, 2021.

[169] IBM, *IBM SevOne Network Performance Management*, 2022.

[170] Apple, *Configuring Network Extensions*, 2021.

[171] C. Wang *et al.*, "Experience: A Three-Year Retrospective of Large-Scale Multipath Transport Deployment for Mobile Applications," in *MobiCom 2023*, APNIC, 2014.

[172] IETF, *IP Proxying Support for HTTP*, 2022.

[173] P. Sattler, J. Aulbach, J. Zirngibl, and G. Carle, "Towards a Tectonic Traffic Shift? Investigating Apple's New Relay Network," in *IMC 2022*, APNIC, 2014.

[174] IETF, *Architectural Guidelines for Multipath TCP Development*, 2011.

[175] T. K. Dang, N. Mohan, L. Corneo, A. Zavodovski, J. Ott, and J. Kangasharju, "Cloudy With a Chance of Short RTTs: Analyzing Cloud Connectivity in the Internet," in *IMC 2021*, APNIC, 2014.

[176] A. Feldmann *et al.*, "Implications of the COVID-19 Pandemic on the Internet Traffic," in *IMC 2020*, APNIC, 2014.

[177] A. Lutu, D. Perino, M. Bagnulo, E. Frias-Martinez, and J. Khangosstar, "A Characterization of the COVID-19 Pandemic Impact on a Mobile Network Operator Traffic," in *IMC 2020*, APNIC, 2014.

[178] Microsoft, *Microsoft Azure Private 5G Core*, 2023.

[179] Nokia, *Nokia 5G Core Software as a Service in Practice*, 2022.

[180] A. Spreadbury and N. Singh, *Azure Operator Voicemail: Take the First Step to Move Voice Workloads to the Cloud*, 2023.

[181] ATT, *ATT Moves 5G Mobile Network to Microsoft Cloud*, 2021.

[182] HashiCorp, *Architecting Geo-Distributed Mobile Edge Application With Consul*, 2022.

[183] S. Narayana, W. Jiang, J. Rexford, and M. Chiang, "Joint Server Selection and Routing for Geo-Replicated Services," in *International Conference on Utility and Cloud Computing 2013*, APNIC, 2014.

[184] H. J. Son, *Comparison of the SD-WAN Vendor Solutions*, 2017.

[185] A. Akella, S. Seshan, and A. Shaikh, "Multihoming Performance Benefits: An Experimental Evaluation of Practical Enterprise Strategies," in *NSDI 2004*, APNIC, 2014.

[186] Megaport, *Agile Networking for Real-Time IT Transformation*, 2023.

[187] VMware, *VMware SD-WAN*, 2023.

[188] Barracuda, *Accelerate Your Business With Secure SD-WAN*, 2023.

[189] IETF, *Multipath Extension for QUIC*, 2020.

[190] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and Evaluation," in *CoNEXT 2017*, APNIC, 2014.

[191] Apple, *Improving Network Reliability Using Multipath TCP*, 2020.

[192] J. Jiang *et al.*, "Via: Improving Internet Telephony Call Quality Using Predictive Relay Selection," in *SIGCOMM 2016*, APNIC, 2014.

[193] F. Aschenbrenner, T. Shreedhar, O. Gasser, N. Mohan, and J. Ott, "From Single Lane to Highways: Analyzing the Adoption of Multipath TCP in the Internet," in *IFIP Networking Conference 2021*, APNIC, 2014.

[194] IETF, *Path Aware Network RG*, 2017.

[195] X. Yang, D. Clark, and A. W. Berger, "NIRA: A New Inter-Domain Routing Architecture," *ToN*, 2007.

[196] NANOG, *Panel: Buying and Selling IPv4 Addresses*, 2022.

[197] G. Huston, *BGP Routing Table Analysis Reports*, 2023.

[198] BGP.us, *Full View or Not Full View: The Benefits and Dangers of the Full BGP Table*, 2016.

[199] Z. A. Uzmi *et al.*, "SMALTA: Practical and Near-Optimal FIB Aggregation," in *CoNEXT 2011*, APNIC, 2014.

[200] H. Asai and Y. Ohara, "Poptrie: A Compressed Trie with Population Count for Fast and Scalable Software IP Routing Table Lookup," in *SIGCOMM 2015*, APNIC, 2014.

[201] T. Stimpfling, N. Belanger, J. P. Langlois, and Y. Savaria, "SHIP: A Scalable High-Performance IPv6 Lookup Algorithm that Exploits Prefix Characteristics," *ToN*, 2019.

[202] H. Chen, Y. Yang, M. Xu, Y. Zhang, and C. Liu, "Neurotrie: Deep Reinforcement Learning-Based Fast Software IPv6 Lookup," in *ICDCS 2022*, APNIC, 2014.

[203] Danny Pinto, *What Will Happen When the Routing Table Hits 1024K? 2021.*

[204] Y. Jin, C. Scott, A. Dhamdhere, V. Giotsas, A. Krishnamurthy, and S. Shenker, "Stable and Practical AS Relationship Inference with ProbLink," in *NSDI 2019*, APNIC, 2021.

[205] M. Luckie, B. Huffaker, A. Dhamdhere, V. Giotsas, and K. Claffy, "AS Relationships, Customer Cones, and Validation," in *IMC 2013*, APNIC, 2021.

[206] W. B. De Vries, R. de O. Schmidt, W. Hardaker, J. Heidemann, P.-T. de Boer, and A. Pras, "Broad and Load-Aware Anycast Mapping with Verfploeter," in *IMC 2017*, APNIC, 2021.

[207] Cisco, *Cisco SD-WAN*, 2023.

[208] R. Gao, C. Dovrolis, and E. W. Zegura, "Avoiding Oscillations Due to Intelligent Route Control Systems," in *INFOCOM 2006*, APNIC, 2021.

[209] A. Baliga *et al.*, "VPMN: Virtual Private Mobile Network Towards Mobility-as-a-Service," in *MCS 2011*, APNIC, 2021.

[210] K. Wang, M. Shen, J. Cho, A. Banerjee, J. Van der Merwe, and K. Webb, "MobiScud: A Fast Moving Personal Cloud in the Mobile Network," in *AllThingsCellular 2015*, APNIC, 2021.

[211] flexiWAN, *flexiWAN Documentation*, 2022.

[212] RIPE, *RIPE IPmap*, 2022.

[213] M. Luckie, B. Huffaker, A. Marder, Z. Bischof, M. Fletcher, and K. Claffy, "Learning to Extract Geographic Information from Internet Router Hostnames," in *CoNEXT 2021*, APNIC, 2021.

[214] R. Sommese *et al.*, "MAnycast2: Using Anycast to Measure Anycast," in *IMC 2020*, APNIC, 2021.

[215] P. Gigis *et al.*, "Seven Years in the Life of Hypergiants' Off-Nets," in *SIGCOMM 2021*, APNIC, 2021.

[216] CAIDA, *BGP Stream*, 2023.

[217] B. Schlinker, I. Cunha, Y.-C. Chiu, S. Sundaresan, and E. Katz-Bassett, "Internet Performance from Facebook's Edge," in *IMC 2019*, APNIC, 2021.

[218] M. Apostolaki, A. Singla, and L. Vanbever, "Performance-Driven Internet Path Selection," in *SOSR 2021*, APNIC, 2021.

[219] R. N. Staff, "RIS Live," 2023.

[220] F. Wang, Z. M. Mao, J. Wang, L. Gao, and R. Bush, "A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance," 2021.

[221] L. Gao and J. Rexford, "Stable Internet Routing Without Global Coordination," *ToN*, 2001.

[222] C. Stokel-Walker, *Case Study: How Akamai Weathered a Surge in Capacity Growth*, 2021.

[223] J. Networks, *Understanding CoS Virtual Output Queues (VOQs)*, 2023.

[224] K. Arora, *The Gaming Industry: A Behemoth With Unprecedented Global Reach*, 2023.

[225] Cloudflare, *Cloudflare Workers*, 2024.

[226] Fastly, *Fastly Compute*, 2024.

[227] Edgecast, *The CDN Edge brings Compute closer to where it is needed most*, 2020.

[228] G. Cloud, *Cloud video intelligence api*, 2024.

[229] Microsoft, *Emirates Global Aluminiium cuts cost of manufacturing AI by 86 percent with the introduction of Azure Stack HCL*, 2024.

[230] W. Davis, *Netflix Ends a Three-year Legal Dispute over Squid Game Traffic*, 2023.

[231] A. Dhamdhere *et al.*, "Inferring Persistent Interdomain Congestion," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, APNIC, 2018, pp. 1–15.

[232] P. Shuff, *Building a Billion User Load Balancer*, 2021.

[233] A. Bednarz, *Global Microsoft Cloud-Service Outage Traced to Rapid BGP Router Updates*, 2023.

[234] S. Janardhan, *More details about the october 4 outage*, 2021.

[235] B. Cartwright-Cox, *Grave flaws in bgp error handling*, 2023.

[236] M. Piraux, L. Navarre, N. Rybowski, O. Bonaventure, and B. Donnet, "Revealing the Evolution of a Cloud Provider Through its Network Weather Map," in *IMC 2022*, APNIC, 2021.

[237] L. Poutievski *et al.*, "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *SIGCOMM 2022*, APNIC, 2021.

[238] Google, *Google IPv6*, 2024.

[239] P. Sermpezis and V. Kotronis, "Inferring Catchment in Internet Routing," 2019.

[240] P. Goenka, K. Zarifis, A. Gupta, and M. Calder, "Towards client-side active measurements without application control," *SIGCOMM CCR 2022*, 2022.

[241] C. Raiciu *et al.*, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *NSDI 2012*, APNIC, 2021.

[242] Q. De Coninck and O. Bonaventure, "Multiflow QUIC: A Generic Multipath Transport Protocol," *IEEE Communications Magazine*, 2021.

[243] M. Gartner, *How to setup and configure mptcp on ubuntu*, 2022.

[244] S. Langer, "Approximating Smooth Functions by Deep Neural Networks with Sigmoid Activation Function," *Journal of Multivariate Analysis 2021*, 2021.

[245] Y. Shechtman, A. Beck, and Y. C. Eldar, "GESPAR: Efficient phase retrieval of sparse signals," *IEEE transactions on signal processing*, 2014.

[246] H. Li, J. Qian, Y. Tian, A. Rakhlin, and A. Jadbabaie, "Convex and Non-Convex Optimization Under Generalized Smoothness," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[247] N. Chatzis, G. Smaragdakis, A. Feldmann, and W. Willinger, "There is More to IXPs Than Meets the Eye," *SIGCOMM Computer Communication Review*, 2013.

[248] Y. Nesterov *et al.*, *Lectures on convex optimization*. Springer, 2018, vol. 137.

[249] G. Optimization, *Gurobi Optimizer*, 2024.

[250] T. Flach *et al.*, "Reducing Web Latency: The Virtue of Gentle Aggression," in *SIGCOMM 2013*, APNIC, 2021.

[251] D. Wetherall *et al.*, "Improving Network Availability with Protective ReRoute," in *SIGCOMM 2023*, APNIC, 2021.

[252] P.-y. Chiang *et al.*, "Loss Landscapes Are All You Need: Neural Network Generalization Can be Explained Without the Implicit Bias of Gradient Descent," in *The Eleventh International Conference on Learning Representations*, APNIC, 2022.

[253] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[254] J. Geiping, M. Goldblum, P. E. Pope, M. Moeller, and T. Goldstein, "Stochastic Training is not Necessary for Generalization," *arXiv preprint arXiv:2109.14119*, 2021.

[255] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[256] W. R. Huang *et al.*, "Understanding Generalization Through Visualizations," 2020.

[257] S. Lotfi, M. Finzi, S. Kapoor, A. Potapczynski, M. Goldblum, and A. G. Wilson, "PAC-Bayes Compression Bounds So Tight That They Can Explain Generalization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 459–31 473, 2022.

[258] IETF, *Fast Reroute Extensions to RSVP-TE for LSP Tunnels*, 2005.

[259] IETF, *BGP/MPLS IP Virtual Private Networks (VPNs)*, 2006.

[260] IETF, *A Framework for Integrated Services Operation over Diffserv Networks*, 2000.

[261] IETF, *Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture*, 2023.

[262] N. Merrill and T. N. Narechania, "Inside the Internet," *Duke Law Journal Online (forthcoming 2023)*, 2023.

[263] M. Blumenthal *et al.*, "Can We Save the Public Internet?" *SIGCOMM CCR*, 2024.

[264] M. M. Waldrop, "The Chips are Down for Moore's law," *Nature News*, 2016.