SCULPTOR Has Your Back(up Path): Resilient, Efficient Interdomain Routing for Large Service Providers

Paper #559, 12 pages body, 19 pages total

Abstract

Large cloud and content (service) providers help serve an expanding suite of applications that are increasingly integrated with our lives, but have to contend with an unreliable, unpredictable public Internet to route user requests. To enhance reliability to dynamic scenarios such as failure and DDoS attacks, large service providers overprovision to accommodate peak loads and build emergency systems for shifting excess traffic. We take a different approach with SCULPTOR, which uses a service provider's global resources to handle dynamic traffic conditions. SCULPTOR ensures users have many interdomain routes to deployments, and moves excess traffic to backup paths to avoid congestion and minimize latency. As predicting interdomain routes is challenging, SCULPTOR builds models of Internet routing to solve a large integer optimization problem at scale using gradient descent. We prototyped SCULPTOR on a global public cloud and tested it in real Internet conditions, demonstrating that SCULPTOR uses service provider infrastructure up to **TBD: 28**% more efficiently than other solutions, reduces congestion on links during site failures by **TBD: 17**% on average, and enables service providers to achieve low-latency objectives for up to **TBD: 17**% more traffic.

1 Introduction

Cloud and content providers (hereafter Service Providers) enable Internet applications used daily by billions of users. The applications have increasingly stringent performance and reliability demands, as the Internet is increasingly used for mission-critical applications (*e.g.,* enterprise services [34]) and as performance requirements become tighter (*e.g.,* virtual reality requires 10 ms round trip latency [50] and 3 ms jitter [70]). To meet these stricter requirements with the same fundamental Internet protocols, Service Providers have deployed interconnected sites and connected with thousands of networks in hundreds of locations, offering rich connectivity and distributed capacity.

The deployment's physical resources (*e.g.,* peering links, servers) are provisioned to accommodate demands from (ingress) and to (egress) user networks. Service Providers balance egress demands across links to minimize congestion and latency [64, 80, 25]. However, it is difficult to control exactly how much traffic reaches each site and how much ingresses each link into the Service Provider's network since (a) ingress traffic can only be controlled indirectly, by directing users to different *prefixes*, and (b) it is difficult to advertise prefixes in a way that balances load in dynamic traffic conditions. Current approaches to routing try to achieve low steady state latency, but do not consider whether low latency paths can support dynamic loads (§2.3).

Given this lack of control, resource overload occurs due to link failures [48], DDoS attacks [53, 78, 60], flash crowds [29, 39, 41], and route changes that cause significant traffic shifts [51, 46, 24, 47, 55, 48]. This overload can lead to degraded service for user traffic [52, 58, 35].

Service Providers may respond to resource congestion by (a) proactively overprovisioning resources [71, 1, 43] or (b) reactively draining congested links and moving some of that traffic to other resources with free capacity [48]. However, we demonstrate in Section 2.3 that overprovisioning resources can require overprovisioning rates as high as 70%, significantly increasing expenses.

Service Providers can also drain traffic from overloaded links/sites by *withdrawing* prefix announcements that are also advertised via other healthy links/sites [53, 48, 61], so that traffic destined to those prefixes is forced to arrive on other links/sites after BGP converges (within tens of seconds [85]). However, this solution to overload is error-prone [43], post-hoc, unpredictable, and ineffective.

To solve these problems, we propose a system — SCULPTOR (**S**calable **C**onfigurations for **U**tilization, **L**oss, and **P**erformance-aware **T**raffic **O**ptimization & **R**outing) — that proactively advertises multiple prefixes to peers to expose diverse routing options, then assigns user traffic to paths towards these prefixes to optimize latency in a way that balances load across links/sites. Our key insight is to frame this problem

as finding a way of advertising reachability that minimizes latency subject to capacity constraints both with and without failures as failure requires load balancing traffic on backup paths.

We make two contributions. First, we present an interdomain routing optimization framework that can be used to minimize performance metrics (*e.g.,* maximum link utilization, latency, latency under single link failures) over different sets of prefix advertisements (§3.2). Part of our framework is a model for predicting performance in unknown scenarios, which is important as changing routing configurations and then issuing measurements is slow. This model enables SCULPTOR to efficiently optimize over a large space without measuring every possible configuration (§3.3). We then optimize these (modeled) performance metrics using gradient descent which is appropriate in our setting due to the high dimension of the problem and parallelism that gradient descent admits (§3.4).

Second, we prototype and evaluate our framework in a system, SCULPTOR, at Internet scale using the PEERING testbed [63], which is now deployed at 32 Vultr cloud locations [75]. Vultr is a global public cloud that allows customers to issue BGP advertisements via more than 10,000 peerings (§4).

We demonstrate that SCULPTOR computes prefix advertisements that give users low latency both during normal operation and under variable network conditions, and allows Service Providers use their deployments more efficiently, reducing costs.

Specifically, we found that SCULPTOR reduces latency during both steady state and failure: SCULPTOR improves the amount of traffic within 10 ms of optimal by **TBD: 3**% in steady state (§5.2), **TBD: 11**% during link failure, and **TBD: 17**% during site failure . SCULPTOR also reduces congestion on links during site failures by **TBD: 17**% on average, giving Service Providers more security that services will still be available during partial failure (§5.3). Hence, SCULPTOR can serve more traffic with less congestion very close to a latency that an optimal (but unreasonably expensive) solution can accomplish.

Providing good backup paths to handle dynamic conditions improves more than just latency — we find that by load balancing traffic on backup paths during peak times, we can satisfy very high peak demands with the same infrastructure. SCULPTOR can handle flash crowds (DDoS attacks, for example) at more than **TBD: 3**× expected traffic volume, drastically reducing the amount of overprovisioning Service Providers need, thus reducing costs. SCULPTOR can also handle large diurnal swings of almost **TBD: 2**× expected load (§5.4).

Hence, SCULPTOR improves interdomain routing for users to Service Providers, uses Service Provider resources more effectively, and acts as a tool for more efficient capacity planning preparing Service Providers to provide the increasingly reliable, performant service that our applications need.

2 Motivation and Key Challenges

Service Providers offer their services from tens or hundreds of geo-distributed sites. Each site offers identical service so can serve any user, but users benefit from reaching a low-latency site for performance. Sites consist of sets of servers which have an aggregate capacity. Service Providers also connect to other networks (often thousands) at sites via dedicated links or shared IXP fabrics. Each such link also has a capacity. When utilization of a site or link (collectively, a resource) nears/exceeds 100%, performance suffers, so Service Providers strive to keep utilization reasonably low. Resources can also fail completely due to, for example, physical failure and misconfiguration.

2.1 Tighter Requirements, Variable Conditions

Evolving Internet use cases and new applications with strict performance requirements are pushing Service Providers to offer more reliable, performant service. For example, Service Providers increasingly offer mission-critical services such as enterprise solutions [34] which are predicted to be a $60B industry by 2027 [28], so ensuring performant reliable operation is more important for these services now than ever. Similarly, gaming, an immersive application that requires latency within 50 ms [50], has grown to be worth more than the music and movie industries *combined* [5].

Possibly as a result of these trends, Service Providers have invested in solving problems that compromise their ability to provide low latency, reliable service. Microsoft recently stressed the importance of resolving congestion on ingress links [48], developed a system to identify performance problems on paths [32], and is considering deployment deep within user networks to further enhance ingress routing performance and reliability [34]. Google similarly developed a system to identify performance problems on paths [36]. Research from Facebook, Google, and Microsoft all demonstrates considerable recent focus on providing service even under partial system failure [23, 18, 37, 49, 43].

Despite their focus on solving these problems, Service Providers still face challenges in providing reliable performance especially under dynamic conditions caused by factors *outside* their network. For example, peering disputes can lead to congestion on interdomain links or inflated paths [14, 17], and DDoS attacks can bring down sites/services [60, 52, 58, 35] (despite the considerable effort in mitigating DDoS attack effects [53, 78]). Moreover, recent work shows that user traffic demands are highly variable due to flash crowds and path changes and so are much harder to plan/optimize for than inter-datacenter demands [41, 55]. Operators regularly report these traffic surges on social media and blog posts [29, 39, 51, 46, 24, 47].

2.2 Routing Traffic to Satisfy Requirements

A challenge in offering low-latency, reliable services to users is routing traffic from user networks to Service Provider networks since Service Providers lack full control of which interdomain path traffic takes. BGP, the Internet's interdomain routing protocol, computes paths in a distributed fashion, giving each intermediate network a say in which paths are chosen and which are communicated to other networks. Service Providers can, however, advertise their reachability to peers/providers in different ways to increase the chance of there being good paths for users. Today, Service Providers either direct users to specific sites using `unicast` prefix advertisements [36, 64, 9], or use `anycast` prefix advertisements to provide relatively low latency and high availability at the expense of some control [6, 33, 85, 84].

`anycast`, where Service Providers advertise a single prefix to all peers/providers at all sites, offers high availability following failures since BGP automatically ensures reachability to the deployment after tens of seconds for most networks [85]. Prior work shows that this availability comes with higher latency in some cases [42, 33]. `unicast` can give clients lower latency than `anycast` by advertising a unique prefix at each site [9], but can suffer from reliability concerns [34]. Hence, recent work proposes hybrids of `unicast` and `anycast` which achieve a tradeoff between performance and availability [6, 81, 84, 34]. This latter class of solutions advertises different prefixes to subsets of all peers/providers 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 peers/providers) and `unicast` (many prefixes, selectivity).

2.3 Current Approaches Do Not Consider Dynamic Conditions

`anycast`, `unicast`, and `selectivecast` approaches do not, however, plan for changing traffic conditions. These approaches only give users low-latency *primary* paths. Figure 1 shows how a certain type of dynamic condition — failure — can lead to performance problems, even with `selectivecast` advertisements. In normal operation, user traffic is split evenly across two prefixes (Fig. 1a). However when site B fails, BGP chooses the route through Provider 1 for all traffic causing link overutilization (Fig. 1b).

Instead, the current state-of-practice to handle dynamic conditions is to reactively drain traffic from overloaded links/sites by *withdrawing* prefix announcements that are also advertised via other healthy links/sites [53, 48, 61]. The traffic destined to those prefixes then arrives on other links/sites advertising those prefixes after BGP reconverges. However, this solution to handling dynamic conditions is error-prone [43], post-hoc, unpredictable (as BGP is unpredictable), and ineffective. For example, `TIPSY` only achieves 70% prediction accuracy and (in a case study) required several prefix withdrawals to mitigate a single congested link [48].

Another common solution to tackling dynamic conditions is to overprovision resources to handle predicted peak loads [71, 1, 43], but Figure 2 demonstrates that doing so can incur excessive costs. Figure 2 computes differences in peak utilization on links between different successive time periods, using longitudinal link utilization data from OVH cloud [56]. We split the dataset into successive, non-overlapping N-day periods and compute changes in 95^{th} percentile link utilization from one period to the next. We choose this percentile to reduce noise. Even when computing near-peak loads over successive 120-day time periods, near-peak loads can increase by 10% for 7% of links, 20% for 2% of links, and increase as high as 70%, which illustrates the amount Service Providers must overprovision to ensure they can meet resource demands. Using backup paths to handle such (rare) events can help keep costs low.

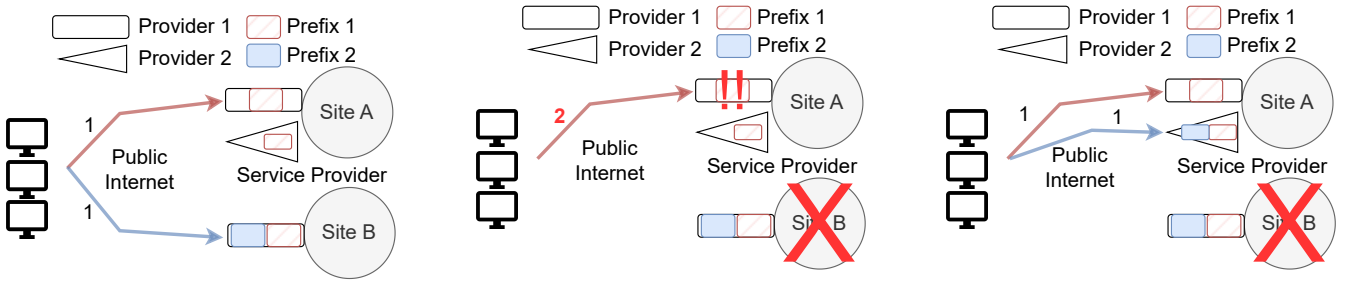
(One Service Provider also stated that solely installing extra capacity is not a feasible solution without shifting traffic [64].)

2.4 Key Problem

The key problem with current practices of routing users to Service Providers is that Service Providers have no guarantee that backup paths/sites can handle new traffic loads.

Prior work finds ways to prepare for intradomain failure even under dynamic traffic conditions [76, 44, 8, 31, 11], which implicitly requires allocating backup paths. Compared to this intradomain setting, two key differences in the interdomain setting are that (a) a Service Provider cannot precisely control the paths traffic takes since BGP relinquishes this control to other networks and (b) a Service Provider cannot easily determine the bottleneck capacity of paths before placing traffic on them. We explicitly address the former problem and leave the latter problem as future work. That is, we consider the problem of ensuring reliable (failure-resistant) interdomain routing when we know the available capacity of backup paths. We approximate the capacity of a path as the capacity of the corresponding peering link through which that path ingresses to the Service Providers' deployment. We consider traffic *ingressing* the deployment since, in the egress setting, the Service Provider has full control over traffic up until the egress point which is the only part of the egress path the Service Provider can control. Existing systems direct egress traffic to optimize performance and mitigate congestion [64, 80, 25].

Although we do not explicitly address problem (b) above, we implicitly address it by finding failure-resistant interdomain routing strategies (*i.e.,* having backup paths to route around congestion is a good thing), and there are natural extensions of our setting that consider such effects. For example,



(a) Normal operation (b) Overloaded link during failure (c) Good operation during failure

Figure 1: In normal operation traffic is split between two sites by directing half the traffic to each prefix (1a). When site B fails, there is enough global capacity to serve all traffic (each link can handle 1 unit) but no way to split traffic across multiple providers given available paths leading to link overload (1b). A *resilient* solution is to advertise prefix 2 to an additional provider at site A, allowing traffic splitting across the two links (1c).

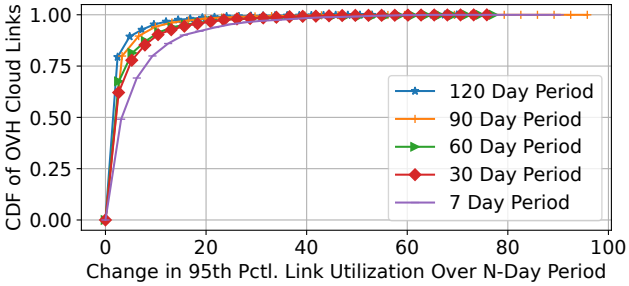


Figure 2: Planning for rare traffic spikes requires significant overprovisioning.

one could use congestion-detection mechanisms along paths to trigger traffic shifts to other, healthier (backup) paths.

Service Providers lack interdomain control for their ingress traffic, but there is hope that they can handle dynamic conditions. A Service Provider has enough global capacity to handle even large changes in traffic or sudden traffic shifts from one link to another [11**?** , 30]. With enough foresight, Service Providers can expose these paths to users a priori to avoid overload. Looking again at the site failure in Figure 1, we see that advertising prefix 2 to provider 2 at site A a priori allows the Service Provider to split traffic between the two links during failure, avoiding overload (Fig. 1c).

2.5 Key Challenges

Since Service Providers connect globally with thousands of networks, there is generally sufficient available capacity across paths/sites to satisfy user demand even under dynamic conditions. For example, an operator at a large Service Provider told us that they often have redundant connections to important customers at multiple sites. Placing traffic demand on paths to optimize performance objectives subject to capacity constraints would therefore be simple *if we could expose all the paths*.

However, exposing paths uses prefixes which are expensive [34]. IPv4 prefixes are monetarily expensive and pollute BGP routing tables. Prior work generally found that advertising O(50) prefixes was acceptable [81, 34], and most Service Providers advertise fewer than 50. IPv6 is not a good alternative, as such entries take 8× the amount of memory to store in a router so would pollute global routing tables even more. 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 satisfy performance objectives, however, is hard since there are exponentially many subsets to consider, and since measuring paths takes time. Therefore, we have to predict how different subsets of paths perform which is challenging since interdomain routing is difficult to model, and since there are too many possible dynamic traffic conditions (failures, attacks) over which to assess these predictions.

3 Methodology

3.1 SCULPTOR Overview

SCULPTOR's high level goal is to find an advertisement strategy that optimizes latency subject to capacity constraints during both steady state and dynamic network conditions. Considering capacity constraints when optimizing latency means that some traffic placement decisions become correlated since different users share the same capacity. Hence, considering capacity constraints makes computing optimized latencies a large optimization problem (O(10k) constraints, O(1M) decision variables) (§3.2.2).

Minimizing this objective function requires evaluating it with several different inputs, but performing such measurements (*i.e.,* advertising prefixes) takes time, and so is not scalable. It would take at least 20 minutes per test to avoid route flap dampening which translates to years of optimization given the problem sizes we consider (millions of evaluations). Instead, we estimate latency. Predicting interdomain paths is hard, however, so we model paths probabilistically (§3.3.1),

and update this model over time using a small number of measurements in the Internet (§3.5).

Computing latency probability distributions for each user network and then considering how correlations between those networks impact capacity *millions* of times is also intractable so, when optimizing, we compute approximately optimal user traffic placement onto paths. Our approximation assigns users in a way that balances a desire for low uncongested latency (§3.3.2), and a desire for load balancing to avoid congestion (§3.3.3).

SCULPTOR's power comes from its precise formulation of a global traffic engineering problem and the local approximations to this problem it makes to form a tractable solution. These approximations allow SCULPTOR to compare millions of possible advertisement strategies at milliseconds per computation, but only conduct tens in the actual Internet to refine its routing model. We now more thoroughly describe how SCULPTOR encounters and overcomes each of these challenges.

3.2 Problem Setup and Definitions

3.2.1 Setting and Goal

We aim to advertise relatively few prefixes to connected networks to offer low-latency paths from user networks to the Service Provider subject to capacity constraints on links. We assess resilience based on single-link and single-site failures, but our methodology extends to the multi-link/site failure case and our evaluation shows that solutions that optimize for these metrics naturally offer other benefits (§5). Adding capacity constraints on sites (*e.g.,* servers) in addition to links would be a straightforward extension.

We assume the Service Provider has some technology for directing traffic towards prefixes. Examples include DNS [9, 6, 36], multipath transport [59, 15], or control-points at/near user networks [62, 34]. DNS offers slow redirection due to caching [34] but is the most readily deployable by the largest number of networks, whereas Service Provider-controlled appliances offer precise control but may not be a feasible option for some Service Providers. Service Providers with stronger incentives to provide the best service to users will invest in better options with more control, and eventually multipath transport will see wide enough deployment to be used by all Service Providers. Today, MPTCP is enabled by default in iOS [4] and Ubuntu 22 [21], and MPQUIC can be installed on any device in user space with applications [83]. In our evaluations, we assume all users can be precisely directed to each prefix.

Service Providers connect to peers/providers at sites via physical connections we call peering links. 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 (and therefore peering link) is chosen via BGP. We consider users at the granularity of user groups (UG*s*), which generally refer to user networks that route to the Service Provider similarly and experience similar latency, but could mean different things to different Service Providers (*e.g.,*/24 IPv4 prefixes, metros). UG*s* generate known traffic volumes, v_{UG}, and the Service Provider provisions capacity at links/sites to accommodate this load. We assume a system run by the Service Provider measures latency from UG*s* to prefixes advertised by the Service Provider (a reasonable assumption [7, 9, 16, 81, 34]). For a given UG, paths towards prefixes may be different and so may have different latencies.

3.2.2 Precise Problem Formulation

As in prior work [81, 34], we model an advertisement configuration A as a set of ⟨peering, prefix⟩ pairs where ⟨peering, prefix⟩ $\in A$ means that we advertise that prefix via that peering. We then seek an advertisement configuration that minimizes latency ($\mathcal{L}$) during normal operation (N) and single link/site failures (F_l):

$$\min_{A} \quad \sum_{\text{UG}} \mathcal{L}_{\text{UG}}(A,N) v_{\text{UG}} + \sum_{l} \alpha_l \sum_{\text{UG}} \mathcal{L}_{\text{UG}}(A,F_l) v_{\text{UG}} \tag{1}$$

The weighting factors α_l control the tradeoff between a desire to minimize steady state latency and to minimize failure-state latencies. Having different α_l for different links/sites l is not necessary, but allows operators to, for example, optimize for failures which are more likely.

We cannot just compute $\mathcal{L}_{\text{UG}}(A,N)$ by assigning each UG to its lowest-latency path, as paths from many UG*s* share links that are capacity constrained. Instead, we need to consider other path options that may have higher uncongested latency but can be used to avoid causing congestion.

A given advertisement configuration during normal operation exposes a set of paths $P(A,N)$ to UG*s* through peering links (l), each of which we identify with tuples (UG, l). (Considering failure scenarios F_l is similar). To compute the latency $\mathcal{L}_{\text{UG}}(A,N)$ a UG experiences under an advertisement configuration A under deployment N, we solve for the globally optimal allocation of UG traffic to paths under A. This allocation is the solution to the following linear program for $w_{UG,l}$ — the allocation of UG traffic to paths.

$$\begin{aligned} \mathbf{w}^*_{\text{UG},l} = \arg\min_{w_{\text{UG},l}} \quad & \sum_{\text{UG},l \in P(A,N)} L_{\text{UG},l}\, w_{\text{UG},l} + \beta MLU \\ \text{s.t.} \quad & w_{\text{UG},l} \geq 0 \\ & \sum_{l} w_{\text{UG},l} = v_{\text{UG}} \quad \forall \text{UG} \\ & MLU \geq \frac{\sum_{\text{UG}} w_{\text{UG},l}}{c_l} \quad \forall l \end{aligned} \tag{2}$$

Latency for a UG in Equation (1) is then the weighted sum

of latencies according to this optimal allocation: $\mathcal{L}_{\text{UG}}(A,N) = \sum_l L_{\text{UG},l} w^*_{\text{UG},l}$.

The minimization term in Equation (2) is the sum of latency and maximum link utilization (*MLU*), weighted by a parameter β. β 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 (2) with $MLU = 1$ to see if we can allocate traffic to paths with zero congestion. However such congestion-free solutions do not always exist for arbitrary advertisement strategies.

The first/second constraints in Equation (2) requires that UG volume on a link $w_{\text{UG},l}$ be non-negative and sum to the UG's total volume v_{UG} across all links. Constraining *MLU* to be at least as much as the utilization of each link (the third constraint) and the minimizing (a sum including) it forces *MLU* to be the maximum link utilization. Hence, optimizing Equation (2) amounts to finding low latency assignments of UGs to links that cause minimal congestion. (We also tried similar optimization problems with different utilization penalties than *MLU*, such as total utilization across paths, and found that they led to similar evaluation results.)

By searching for solutions to Equation (2) in both normal operation N and under link/site failure F_l we encourage low-latency, resilient advertisement strategies in Equation (1).

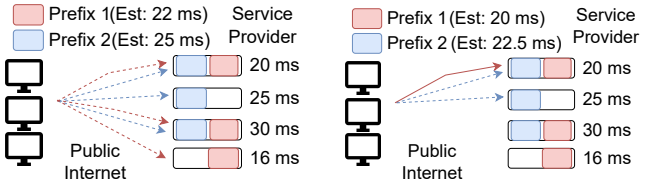
3.3 Accommodating Internet's Limitations

Solving Equation (1) is challenging because computing $\mathcal{L}_{\text{UG}}(A,N)$ requires advertising prefixes in the Internet, which can only be done infrequently to avoid route-flap-dampening. For example, in our optimization (detailed below) we compute $\mathcal{L}_{\text{UG}}(A,N)$ for millions of different A which could take tens to hundreds of years at a rate of advertising one strategy per hour. Hence we model, instead of measure, UG paths and improve this model over time through measurements.

3.3.1 Probabilistic UG Paths

Representing both Equation (1) and Equation (2) as a single mixed-integer linear program (*i.e.,* solving directly for A) would be feasible if we knew exactly how UGs routed to all possible advertisements, but routing is hard to predict [54, 3, 48, 61].

We instead model routing probabilistically and update our probabilistic model over time as we measure how UGs route to the deployment, adapting methods from prior work [81, 66, 34]. That is, in Equation (1) we model $\mathcal{L}_{\text{UG}}(A,N)$ probabilistically and denote the corresponding random variable with a tilde: $\tilde{\mathcal{L}}_{\text{UG}}(A,N)$. Our probabilistic model assumes a priori, for a given UG towards a given prefix, that all ingress options for a UG are equally likely and that we know user latencies to every ingress. 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. As we exclude more options, latency/path distributions on unmeasured scenarios converge to the true latency/path. An example of this process is shown in Figure 3, where we refine our latency estimate towards an unmeasured prefix (blue) using measurements towards other prefixes (red).



(a) (b)

Figure 3: With no knowledge of routing preferences, we estimate latency from this UG to both the red and blue prefixes with the average over possible ingress latencies (3a). We measure towards the red prefix (*e.g.,* using `traceroute`) and learn that the first ingress has higher preference than the third and fourth ingresses. We use this information to refine our latency estimate towards the blue prefix (since the third ingress is no longer a possible option (3b)) without measuring the blue path.

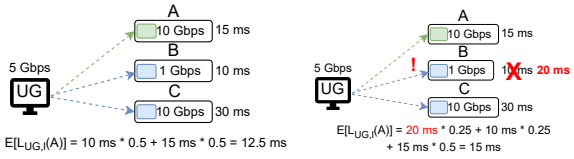
3.3.2 Capacity-Free UG Latency Calculation

To assign traffic to prefixes we solve Equation (2). However, for unmeasured advertisements, we only have an estimate of the distribution of possible (as opposed to actual) paths. For a given A, there are many possible paths from UGs to prefixes, so Equation (2) is too challenging to solve even probabilistically. Instead, when computing $\tilde{\mathcal{L}}_{\text{UG}}(A,N)$ we assume every user takes their lowest-latency path. Such decisions do not take capacity into account, which we account for later.

Since we model paths as probabilistic, user path choices and what latency UGs experience ($\sum_{\text{UG}} \tilde{\mathcal{L}}_{\text{UG}}(A,N)\ v_{\text{UG}}$) is also probabilistic. We model the objective function in Equation (1) as the *expected value* of the summations. That is, if the latencies for UG across different paths under advertisement A are $\tilde{\mathcal{L}}_{\text{UG}}(A,N)$, we compute the distribution of the minimum choice, $\min_l \tilde{\mathcal{L}}_{\text{UG}}(A,N)$, and then compute the distribution over the sum $\sum_{\text{UG}} \min_l \tilde{\mathcal{L}}_{\text{UG}}(A,N)\ v_{\text{UG}}$. We use the minimum latency choice as a simple approximation of how traffic is directed to sites for which we can compute a probability distribution (as opposed to solving Equation (2), which we cannot do probabilistically).

3.3.3 Imposing Capacity Violation Penalties

It could be that such minimum latency assignments lead to congested links, especially during link/site failures. We would like to penalize such advertisement scenarios, and favor those that distribute load more evenly without solving Equation (2) which would take too long. Hence, before computing the expected latency, we first compute the probability that links are



(a) (b)

Figure 4: 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 (4a). 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 (4b).

congested by computing the distribution of link utilizations from the distribution of UG assignments to paths. We then inflate latency for UG*s* on 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 congestion by inflating latency in this calculation for UG*s* on those paths proportional to the overutilization factor so that minimum latency path choices would push users to another path. After emulating the effect of this congestion, we recompute the distribution of average UG latency **without changing UG decisions**, and take the expected value of this random variable in Equation (1) as a proxy for the true latency. We do not change UG decisions as doing so could induce an infinite calculation loop if new decisions also lead to congestion. 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 4.

During optimization we observed that these heuristics tended to yield accurate estimates of overall benefit when averaged across the entire deployment (*i.e.,* Equation (2)), despite inaccuracies in predicting any individual UG's latency or link's utilization.

3.4 Solving with Gradient Descent

We cannot solve Equation (1) directly/exhaustively since the optimization variable is an integer matrix. Instead, we approximate the optimization variable A as a continuous variable with entries between 0 and 1 and threshold its entries at 0.5 to determine if a prefix is advertised/not advertised to a certain peer/provider. We then solve Equation (1) using gradient descent, approximating gradients between adjacent advertisements using sigmoids, adopting methods used in the optimization literature [40].

Gradient descent is appropriate for two reasons. First, it is parallelizable, which is important given the high dimension of A (tens to hundreds of thousands of entries representing all ⟨peering, prefix⟩ pairs). Second, it simultaneously weighs several (possibly opposing) goals in Equation (1) according to their global importance in minimizing the objective. Intuitively, Equation (1) has opposing goals, since advertising prefixes to expose backup paths may push some UG*s* off their lowest-latency route but may also encourage demand spreading during failures, which is more important than having "optimal" steady state latency. Gradient descent takes into account both of these advantages across all ⟨peering, prefix⟩ pairs simultaneously.

We do not compute all entries of the gradient of Equation (1) since there are too many to compute. Instead, we use Monte-Carlo techniques, subsampling entries of the gradient to estimate the gradient as in prior work [67].

Upon thresholding entries of our optimization variable A at 0.5, we obtain an advertisement strategy to test in the Internet. We conduct such advertisements during optimization if we have not advertised another strategy in the last hour (to avoid route flap dampening). Conducting such advertisements refines our latency estimates as they inform us of which ingresses are more preferred.

Minimization scales linearly with the number of UG*s*, quadratically with the number of peers/providers, and linearly with the number of prefixes. Despite this high complexity, in practice the implementation runs quickly (minutes per gradient step) relative to the rate at which we can advertise prefixes (once an hour). Although solving Equation (1) does not have convergence guarantees, we have found that it finds good solutions over a wide range of simulated topologies, and converges quickly with thousands of UG*s* and ⟨peering, prefix⟩ pairs (§5).

3.5 Exploration to Improve Model

As, a priori, we do not know UG preferences, we may be uncertain about the benefit of other advertisement strategies which could make solving our model using gradient descent noisy/unstable. For example, an adjacent strategy could have better *expected* latency than the current one but have much worse *actual* latency. Gradient descent would then push us towards a worse strategy.

To improve convergence, we periodically compute benefit distributions on adjacent advertisement schemes and look for strategies for which we are very uncertain about whether it will be better or worse. One could use different measures of uncertainty — we choose entropy. Testing advertisement

strategies in the Internet takes time (as measuring all paths takes time) but we observed that it improves convergence.

4 Implementation

We prototype SCULPTOR on the PEERING testbed [63], 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) demonstrate SCULPTOR's potential in an actual Service Provider and our open/reproducible methodology provides value to the community.)

4.1 Simulating Clients and Traffic Volumes

To simulate client performances, we measured actual latency from IP addresses to our PEERING prototype as in prior work [34] 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 exhaustively 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.

After tabulating 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 [**?**]). Vultr likely peers with networks with which it exchanges a significant amount of traffic [64], 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 peers. 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, we believe that by 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.

4.2 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.

4.2.1 Experiments in the Internet

We assess how SCULPTOR performs in the Internet using RIPE Atlas probes [69], 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 (§3). However, RIPE Atlas does not have large coverage, as probes are only in 3,000 networks, and we are limited by RIPE Atlas probing rate constraints (15k traceroutes/day). Choosing RIPE Atlas probes to maximize network coverage and geographic diversity, we select probes from **TBD: 1,992** networks in **TBD: 121** countries which have paths to **TBD: 580** ingresses.

4.2.2 Simulations

We also evaluate SCULPTOR by simulating user paths. Simulating user paths allows us to conduct more extensive evaluations as experiments take less time and use clients in more networks. To limit computational complexity, we select 100 clients with a route through each ingress. Service Providers could similarly limit computational complexity by optimizing over, for example, a certain percent of the traffic which is a common practice in the networking optimization literature [6, 45, 34, 55]. Over all studied deployments, we consider paths from clients in 42k /24s to 868 peers/providers.

4.3 Setting Resource Capacities

We assume that resource capacities are overprovisioned proportional to their usual load. However, we do not know the usual load of links and cannot even determine which peering link traffic to one of our prefixes arrives on, as Vultr does not give us this information. (This limitation only exists since we are not a Service Provider, as a Service Provider could measure this using IPFIX, for example to measure steady state link loads.) We overcome this limitation using two methods (corresponding to our two deployments — Section 4.2), each with their pros and cons.

For our first method of inferring client ingress links, we advertise prefixes into the Internet using the PEERING testbed [63], and measure actual ingress links to those prefixes using traceroutes from RIPE Atlas probes [69]. 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 the 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.

The second method we use to determine ingress links is simulating user paths by assuming we know all users' *preference models* (§3.3). 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. However, we found that our key evaluation results (§5) hold regardless of how we simulated routing conditions (we also tried completely random routing), suggesting that our methodology is robust to such assumptions. Prior work also found the preference model to be valid in 90% of cases they studied [81].

Given either method of inferring client ingress links we then measure paths to an anycast prefix and assign resource capacities as some overprovisioned percentage of this catchment. (Discussions with operators from CDNs suggested that they overprovision using this principle.) We choose an overprovisioning rate of 30%.

5 Evaluation

SCULPTOR compares favorably to all other solutions on all studied dimensions. In particular, it yields an advertisement strategy that gives clients lower latency paths and fewer congested links during both steady state (§5.2) and failure (§5.3). Hence, SCULPTOR uses the same routing resources (prefixes, sites, links) more efficiently than other solutions, improving performance and reducing cost. We then demonstrate how planning for failure improves infrastructure utilization *even during steady state* by exposing well-performing backup paths for users (§5.4).

5.1 General Evaluation Setting

We compare SCULPTOR's advertisement strategy to other advertisement strategies. The strategies include

anycast: A single prefix announcement to all peers/providers at all sites, which is a very common strategy used by Service Providers today [22, 12, 6, 74, 57, 79, 84].

unicast: A single prefix announcement to all peers/providers at each site (one per site). Another common strategy used in today's deployments [36, 9].

AnyOpt/ PAINTER: Two strategies proposed in the literature for reducing steady state latency compared to anycast [81, 34].

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.)

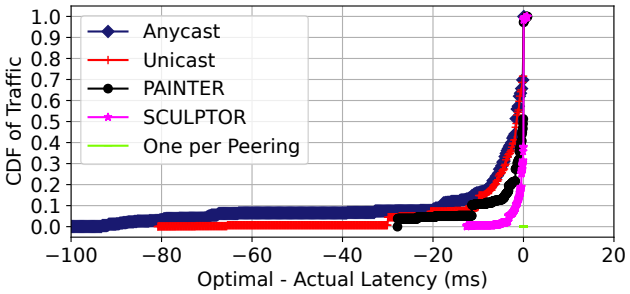
We conduct evaluations both on the public Internet (*i.e.,* with real routing dynamics) and in simulation (with simulated user path preferences). Evaluating SCULPTOR in the public Internet demonstrates that SCULPTOR provides real, tangible benefits, but these evaluations take longer than simulated ones as advertising prefixes is slow. Evaluating SCULPTOR using simulated routing dynamics allows us to conduct micro-evaluations in a variety of hypothetical scenarios.

For our evaluation on the public Internet, we limit the scale of our deployment to **TBD: 10** large sites to avoid reaching RIPE Atlas daily probing limits. These 10 sites were Miami, New York, Chicago, Dallas, Atlanta Paris, London, Stockholm, Sao Paulo, and Madrid as we found it valuable to thoroughly evaluate how our system works in a few countries, rather than sparsely evaluate how it works over many countries. We do not compute the solution for AnyOpt for our evaluations on the Internet since AnyOpt did not perform well compared to any other solution in our simulations, and since computing AnyOpt takes a long time.

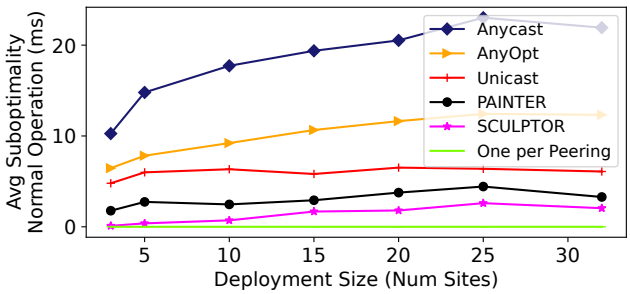
For our simulated evaluations, we compute solutions over many random routing preferences, demands, and subsets of sites to demonstrate that SCULPTOR's benefits are not limited to a specific deployment property. We evaluate SCULPTOR over deployments of size 3, 5, 10, 15, 20, 25, and 32 sites. Each size deployment was run at least **TBD: 10** times with random subsets of UGs, UG demands, and routing preferences. We use twice the base-2 logarithm of the number of peers/providers as the number of prefixes in our budget for all solutions (except **One-per-Peering**). For larger deployments (20-32 sites) we use approximately 60 prefixes, and for smaller ones (3-15 sites) we use between 10 and 30. Prior work found that using this many prefixes to improve performance was a reasonable cost [81, 34].

In solving Equation (2), the lowest link utilization solution may have overloaded resources in failure scenarios. We say all traffic arriving on a congested link is congested and do not include this traffic in latency comparisons (actual latency would be a complicated function of congestion control protocols and queueing behavior). We comment separately on how much traffic is congested.

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.



(a)



(b)

Figure 5: SCULPTOR finds advertisement solutions that yield low steady state latency subject to capacity constraints both on the Internet (5a) and in simulation (8a).

5.2 SCULPTOR Finds Good Steady State Paths

We first assess SCULPTOR's ability to find low-latency solutions during steady state. Figure 5a and Figure 8 demonstrate that SCULPTOR finds lower latency paths during steady state for more traffic than any other solution.

5.2.1 Internet Deployment

We compute the latency each advertisement solution achieves compared to the One-per-Peering solution for each UG. Figure 5a shows a CDF of these differences weighted by traffic, demonstrating that SCULPTOR outperforms other solutions with **TBD: nums**.

5.2.2 Simulations

Figure 8a 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 **TBD: 0.1** to **TBD: 2.5** ms worse than (the undeployable) One-per-Peering. The next-best solution (PAINTER) is on average **TBD: 1.2** ms and **TBD: 2.4** ms worse than SCULPTOR.

anycast comparatively performs the worst with users being on average **TBD: 17.9 ms** worse than One-per-Peering whereas SCULPTOR performs the best with users being **TBD: 1.3 ms** worse than One-per-Peering. Interestingly, unicast (**TBD: 6.1 ms** worse than One-per-Peering) actually performs better than AnyOpt (**TBD: 10.1 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 a small number of provider connections which does not capture the realities of many Service Providers).

These average latency improvements translate into quantifiable routing inefficiency for different fractions of traffic (we include full results in \Cref{ap:. SCULPTOR has on average **TBD: 94.8**% traffic within 10 ms of the One-per-Peering solution, **TBD: 99.2**% within 50 ms, and **TBD: 99.9**% within 100 ms. These percentages compare to the next-best solution, PAINTER, which has on average **TBD: 91.7**% traffic within 10 ms of the One-per-Peering solution, **TBD: 97.4**% within 50 ms, and **TBD: 99.2**% within 100 ms.

Providing lower steady state latency than all other solutions was not an explicitly stated goal of SCULPTOR, but happens anyway, demonstrating the power of SCULPTOR's efficient path modeling and optimization methodology.

5.3 SCULPTOR Improves Resilience to Ingress/Site Failure

We next assess SCULPTOR's ability to provide resilience to ingress failures and site failures. Examples of such failures include excessive DDoS traffic on the link/site (thus using the link/site as a sink for the bogus traffic), physical failure, draining sites, and/or planned maintenance. We now demonstrate that SCULPTOR shifts traffic without overloading alternate links/sites more effectively than any other solution *without reconfiguring BGP* which is error-prone [43] and slow [85].

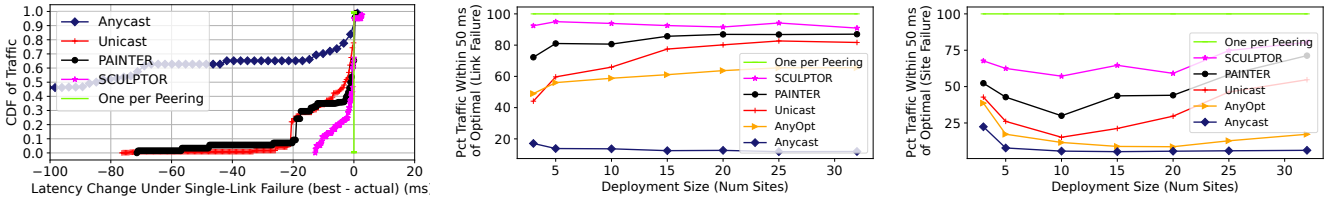
In these evaluations, we fail each ingress/site once and compute UG to link allocations using Equation (2). For each advertisement strategy and failed component, we compute the difference between achieved latency and One-per-Peering latency for users who use that component in their steady state solution. For example, if the Tokyo site fails, we report on the post-failure latency of users served from Tokyo before the failure. We do not include the latencies of users who were already served from other sites.

We separately note the fraction of traffic that lands on congested links for each solution. We do not include congested traffic in average latency computations but say such traffic does *not* satisfy 10 ms, 50 ms, or 100 ms objectives.

Figure 6 demonstrates that SCULPTOR finds better backup paths using our prototype on the Internet (Fig. 6a) and in simulation (Fig. 10c, Fig. 10g). We comment on all results here, but include more figures in **??**).

5.3.1 Internet Deployment

Figure 6a demonstrates that SCULPTOR offers lower latency paths for more users **TBD: nums**.



(a) (b) (c)

Figure 6: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of their optimal latency during normal operation.

5.3.2 Simulations

We show the fraction of traffic within 50 ms of the One-per-Peering solution for link and site failures in Figure 10c and Figure 10g.

For single-link failures, anycast comparatively performs the worst with users being **TBD: 55.4 ms** worse than One-per-Peering on average, whereas SCULPTOR ranges from **TBD: 1.4 ms** and **TBD: 9.8 ms** worse than One-per-Peering on average. SCULPTOR also avoids more congestion, with only **TBD: 0.6**% of traffic being congested on average while PAINTER (the next-best solution) leads to **TBD: 2.3**% of traffic being congested on average. Single-site failure exhibits similar trends where SCULPTOR is **TBD: 10.5** ms worse than One-per-Peering and has **TBD: 22.9**% traffic congestion, on average, while PAINTER leads to **TBD: 19.5** ms worse than One-per-Peering and **TBD: 39.9**% congestion on average.

When looking at routing inefficiency, SCULPTOR has **TBD: 78.2**% of traffic within 10 ms of the One-per-Peering solution on average, **TBD: 93.0**% within 50 ms, and **TBD: 97.2**% within 100 ms (during link failure). These statistics compare to the next-best solution, PAINTER, which has **TBD: 66.9**% within 10 ms, **TBD: 83.4**% within 50 ms, and **TBD: 90.4**% within 100 ms. Even differences of a few percent are significant, as they could represent large numbers of users in a global-scale deployment. Site failures show similar trends.

Differences quoted above may seem small (for example, 11.3% in the amount of traffic within 10 ms of One-per-Peering between SCULPTOR and PAINTER), but represent potential fractions of *trillions* of requests per day made across the whole world [64]. Large Service Providers frequently emphasize that these seemingly small gains make large differences [19, 77].

It is interesting that both PAINTER and unicast provide good resilience compared to anycast, or at least better than we expected, especially as the number of sites increases. Speculating on why these solutions provide decent resilience, we note that unicast (unlike SCULPTOR and PAINTER) exposes one path to all sites for all users, and so we expect that, on average, most users will have at least one decent backup path, even if that backup path is not the best one. Similarly, PAINTER tries to expose the best path for every user. We hypothesize that this path exposure randomly exposes good backup paths for most users, even though PAINTER did not consider that benefit in its allocation process.

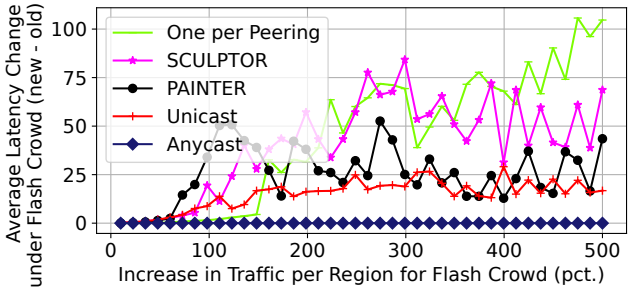
5.4 SCULPTOR Improves Infrastructure Utilization

One response to increased link/site utilization is to install more capacity so that there is sufficient headroom to satisfy peak demand. Figure 7 shows that this response is not always necessary with better routing — SCULPTOR finds ways to distribute load over existing infrastructure to accommodate increased demand without adversely affecting latency. We quantify this improved infrastructure utilization by showing how much extra peak load each routing strategy can help satisfy without congestion, where peak load is distributed according to two realistic traffic patterns: flash crowds and diurnal effects.

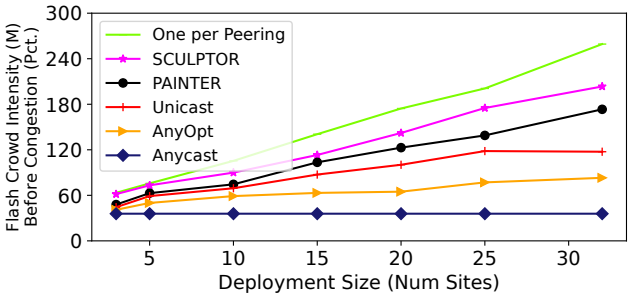
We define a flash crowd as a momentary traffic increase at a single site. Examples of flash crowds might include content releases (games, software updates) that spur downloads in a particular region. Since increased demand is highly localized, we can spread excess demand to other sites if paths to those sites exist.

To generate Figure 7, we identify each client with a single "region" (*i.e.,* site) 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 using Equation (2). 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. We increase M until any link experiences congestion when we (separately) scale volume in any region. For example, if Atlanta experiences congestion with a 60% increase in traffic for Atlanta clients, but no other region does, we stop at that critical value of $M = 60\%$.

Our diurnal analysis uses a similar methodology. We comment on results here and include figures showing those results in Appendix A.3. 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 large Service Provider demonstrates that these effects can cause large differences between peak and



(a)



(b)

Figure 7: SCULPTOR improves infrastructure utilization under flash crowds so that Service Providers can underprovision compared to peak loads.

mean site volume [11]. 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 $0.1M$. As in our flash crowd analysis, we increase M until at least one link experiences congestion at least one hour of the day.

5.4.1 Internet Deployment

Figure 7a shows the change in latency and critical M value for each solution computed on the Internet.

5.4.2 Simulations

Figure 7b plots the average over simulations of critical M values that cause congestion for each deployment size under flash crowds.

Figure 7b shows that SCULPTOR finds ways to route more traffic without congestion than other solutions. SCULPTOR can handle single-site flash crowds at **TBD: 200**% more than the expected volume for deployments with 32 sites, creating a 3× savings in provisioning costs assuming Service Providers plan for peak loads (**TBD: 22**% more than PAINTER). SCULPTOR also handles more intense diurnal traffic swings (appendix A.3), allocating traffic to paths without congestion for between **TBD: 11**% and **TBD: 23**% 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.

Moving traffic to backup paths does not necessarily mean reduced performance. In practice, Service Providers can move less latency-sensitive traffic onto these backup paths so as to not impact high-level application goals. Some intradomain traffic engineering systems use similar mechanisms [30, 26, 25].

Prior work noted similar benefits by satisfying user requests in distant sites [11], but moved traffic on their own backbone to realize these gains. 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. From conversations with operators from both Meta and Microsoft, we also learned that their systems (Edge Fabric [64] and Fastroute [20]) do not shift traffic on their backbones as backbone capacity is precious.

It is interesting that PAINTER provides similar benefits to unicast— this result is likely since PAINTER only aims to reveal low-latency primary paths for users, whereas unicast guarantees reachability to all sites guaranteeing users a minimum number of paths (one of which may have capacity).

5.5 Why SCULPTOR Works

SCULPTOR tries to solve a challenging integer optimization problem with an objective function it can only approximate and, as our analysis demonstrates, finds solutions that outperforms other approaches to the problem.

SCULPTOR's power comes from its ability to quickly predict how many other advertisement strategies perform so that it can pick the best ones. This prediction is an approximation in several senses (§3.1) and the question we aim to answer here is why those approximations work.

First, SCULPTOR compares far more advertisement strategies than the other solutions. In our 32 site deployments over approximately 200 gradient steps SCULPTOR estimates latencies in approximately 13M scenarios across every single UG. PAINTER only considers thousands (2k), and AnyOpt considers 1k (a configurable number).

PAINTER has a simple way of estimating latencies in unseen scenarios, but its greedy approach means that it only considers a single strategy per measurement iteration. AnyOpt can estimate latencies in unseen scenarios, but randomly searches through the space. Hence, both AnyOpt and PAINTER conduct a large number of measurements on the Internet relative to the configurations they estimate. SCULPTOR, on the other hand, only conducts measurements for exploration (**??**) or for strategies for which SCULPTOR thinks will yield good performance (§3.4). **TBD: fig** shows how SCULPTOR's modeling uncertainty exponentially decreases as it makes these measurements on the Internet.

6 Related Work

Egress Traffic Engineering Prior work noted that traffic from large service providers to users sometimes experienced suboptimal performance (congestion, high latency) due to BGP's limitations [80, 64, 38]. 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 [82, 68, 11]. SCULPTOR adapts this idea in a new way but instead uses the public Internet to carry traffic to lower costs.

Ingress Traffic Engineering Other work aims to overcome the limitations of BGP to perform ingress traffic engineering [6, 45, 20, 81, 85, 84, 34], but optimizes for steady state conditions. SCULPTOR optimizes for dynamic traffic conditions, reducing costs and enhancing performance compared to these solutions as we demonstrate in Section 5. Pecan [73] and Tango {tango} exhaustively expose 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 expose, and doing so would require too many prefixes.

Other work and companies create overlay networks and balance load through paths in these overlay networks to satisfy latency requirements [72, 27, 2, 13, 41]. 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.

Prior work built a BGP playbook to mitigate DDoS attacks [61], but it is unclear if those strategies would scale to large service providers (they tested on a few sites and a few providers). SCULPTOR provides a scalable approach to dealing with arbitrary dynamic traffic conditions over thousands of peering links.

Microsoft withdraws prefixes to mitigate congestion on links [48]. However, as they demonstrate in their paper, such actions are unpredictable and can lead to further congestion. SCULPTOR instead plans ahead, minimizing congestion time and providing better resilience guarantees.

New Last-Mile Technology to Improve Performance Xlink uses MPQUIC over 5G, WiFi, and LTE to offer improved bandwidth for video streaming services on mobile devices [83]. DChannel uses both of 5G's high bandwidth and low latency channels to optimize web performance [65].TGaming [10] uses feedback from a 5G network telemetry system to improve performance. All of these systems can work alongside SCULPTOR, as SCULPTOR uses multiple endpoints (as opposed to multiple sources) to enhance performance.

Intradomain Failure Planning Large Service Providers have shown significant interest in reducing the frequency/impact of failures in their global networks [18, 23, 37, 43]. SCULPTOR works alongside these systems by, for example, enabling Service Providers to shift traffic away from failed components/regions while still retaining good performance.

Other prior work tried to plan intradomain routes to minimize the impact of k-component failures [76, 44, 8, 31, 11]. SCULPTOR uses different strategies to accomplish a similar goal that account for the unique challenges in interdomain routing (see Section 2.4 for a more thorough comparison).

7 Conclusions

As the role that Service Providers play in our daily lives increases, and as applications require increasingly demanding performance from the network, Service Providers need better guarantees that their services will continue to work well in an unpredictable Internet. SCULPTOR uses efficient Internet models and optimization techniques to provide these better guarantees by tapping into the unused capacity of a Service Provider's global resources, offering better resilience to link/site failure and traffic growth. SCULPTOR provides better benefits from prior work, but it is unclear if/how much better other solutions to this problem could do. We hope that SCULPTOR spurs work investigating further optimization in resilient interdomain routing, to see whether better strategies exist and, if so, how those strategies can be computed.

References

[1] Satyajeet Singh Ahuja, Varun Gupta, Vinayak Dangui, Soshant Bali, Abishek Gopalan, Hao Zhong, Petr Lapukhov, Yiting Xia, and Ying Zhang. [n.d.]. Capacity-efficient and uncertainty-resilient backbone network planning with hose. In *SIGCOMM 2021*.

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

[3] Ruwaifa Anwar, Haseeb Niaz, David Choffnes, Ítalo Cunha, Phillipa Gill, and Ethan Katz-Bassett. [n.d.]. Investigating interdomain routing policies in the wild. In *IMC 2015*.

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

[5] Krishan Arora. 2023. The Gaming Industry: A Behemoth With Unprecedented Global Reach. https://forbes.com/sites/forbesagencycouncil/2023/11/17/the-gaming-industry-a-behemoth-with-unprecedented-global-reach

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

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

[8] Yiyang Chang, Chuan Jiang, Ashish Chandra, Sanjay Rao, and Mohit Tawarmalani. 2019. Lancet: Better network resilience by designing for pruned failure sets. (2019).

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

[10] Hao Chen, Xu Zhang, Yiling Xu, Ju Ren, Jingtao Fan, Zhan Ma, and Wenjun Zhang. 2019. T-gaming: A cost-efficient cloud gaming system at scale. *IEEE Transactions on Parallel and Distributed Systems* (2019).

[11] David Chou, Tianyin Xu, Kaushik Veeraraghavan, Andrew Newell, Sonia Margulis, Lin Xiao, Pol Mauri Ruiz, Justin Meza, Kiryong Ha, Shruti Padmanabha, et al. [n.d.]. Taiji: Managing Global User Traffic for Large-Scale Internet Services at the Edge. In *SOSP 2019*.

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

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

[14] Wes Davis. 2023. Netflix ends a three-year legal dispute over Squid Game traffic. https://theverge.com/2023/9/18/23879475/netflix-squid-game-sk-broadband-partnership

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

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

[17] Amogh Dhamdhere, David D Clark, Alexander Gamero-Garrido, Matthew Luckie, Ricky KP Mok, Gautam Akiwate, Kabir Gogia, Vaibhav Bajpai, Alex C Snoeren, and Kc Claffy. 2018. Inferring persistent interdomain congestion. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*. 1–15.

[18] John P Eason, Xueqi He, Richard Cziva, Max Noormohammadpour, Srivatsan Balasubramanian, Satyajeet Singh Ahuja, and Biao Lu. [n.d.]. Hose-based cross-layer backbone network design with Benders decomposition. In *SIGCOMM 2023*.

[19] Tobias Flach, Nandita Dukkipati, Andreas Terzis, Barath Raghavan, Neal Cardwell, Yuchung Cheng, Ankur Jain, Shuai Hao, Ethan Katz-Bassett, and Ramesh Govindan. [n.d.]. Reducing web latency: the virtue of gentle aggression. In *SIGCOMM 2013*.

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

[21] Marten Gartner. 2022. How to setup and configure mptcp on Ubuntu. https://medium.com/high-performance-network-programming/how-to-setup-and-configure-mptcp-on-ubuntu-c423dbbf76cc

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

[23] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. [n.d.]. Evolve or die: High-availability design principles drawn from googles network infrastructure. In *SIGCOMM 2016*.

[24] Andrew Griffin. 2024. Facebook, Instagram, Messenger down: Meta platforms suddenly stop working in huge outage. https://independent.co.uk/tech/facebook-instagram-messenger-down-not-working-latest-b2507376.html

[25] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. [n.d.]. Achieving High Utilization with Software-Driven WAN. In *SIGCOMM 2013*.

[26] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Chandan Bhagat, Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, et al. [n.d.]. B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for Availability and Scale in Google's Software-Defined WAN. In *SIGCOMM 2018*.

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

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

[29] Mark Jackson. 2020. Record Internet Traffic Surge Seen by UK ISPs on Tuesday. https://ispreview.co.uk/index.php/2020/11/record-internet-traffic-surge-seen-by-some-uk-isps-on-tuesday.html

[30] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. [n.d.]. B4: Experience with a Globally-Deployed Software Defined WAN. In *SIGCOMM 2013*.

[31] Chuan Jiang, Sanjay Rao, and Mohit Tawarmalani. [n.d.]. PCF: provably resilient flexible routing. In *SIGCOMM 2020*.

[32] Yuchen Jin, Sundararajan Renganathan, Ganesh Ananthanarayanan, Junchen Jiang, Venkata N Padmanabhan, Manuel Schroder, Matt Calder, and Arvind Krishnamurthy. [n.d.]. Zooming in on wide-area latencies to a global cloud provider. In *SIGCOMM 2019*.

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

[34] Thomas Koch, Shuyue Yu, Ethan Katz-Bassett, Ryan Beckett, and Sharad Agarwal. [n.d.]. PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks. In *SIGCOMM 2023*.

[35] Sam Kottler. 2018. February 28th DDoS Incident Report. https://github.blog/2018-03-01-ddos-incident-report/

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

[37] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. [n.d.]. Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *NSDI 2022*.

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

[39] Martyn Landi. 2023. Return of original Fortnite map causes record traffic on Virgin Media O2 network. https://independent.co.uk/tech/fortnite-twitter-b2442476.html

[40] Sophie Langer. 2021. Approximating smooth functions by deep neural networks with sigmoid activation function. *Journal of Multivariate Analysis 2021* (2021).

[41] Jinyang Li, Zhenyu Li, Ri Lu, Kai Xiao, Songlin Li, Jufeng Chen, Jingyu Yang, Chunli Zong, Aiyun Chen, Qinghua Wu, et al. [n.d.]. Livenet: a low-latency video transport network for large-scale live streaming. In *SIGCOMM 2022*.

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

[43] Bingzhe Liu, Colin Scott, Mukarram Tariq, Andrew Ferguson, Phillipa Gill, Richard Alimi, Omid Alipourfard, Deepak Arulkannan, Virginia Jean Beauregard, Patrick Conner, et al. [n.d.]. CAPA: An Architecture For Operating Cluster Networks With High Availability. In *NSDI 2024*.

[44] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. [n.d.]. Traffic engineering with forward fault correction. In *SIGCOMM 2014*.

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

[46] Doug Madory. 2024. GP Leak Leads to Spike of Misdirected Traffic. https://kentik.com/analysis/BGP-Routing-Leak-Leads-to-Spike-of-Misdirected-Traffic

[47] Doug Madory. 2024. Outage Notice From Microsoft. https://twitter.com/DougMadory/status/1768286812894605517

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

[49] Jeffrey C Mogul, Rebecca Isaacs, and Brent Welch. [n.d.]. Thinking about availability in large service infrastructures. In *HotOS 2017*.

[50] Nitinder Mohan, Lorenzo Corneo, Aleksandr Zavodovski, Suzan Bayhan, Walter Wong, and Jussi Kangasharju. [n.d.]. Pruning Edge Research With Latency Shears. In *HotNets 2020*.

[51] Scott Moritz. 2021. Internet Traffic Surge Triggers Massive Outage on East Coast. https://bloomberg.com/news/articles/2021-01-26/internet-outage-hits-broad-swath-of-eastern-u-s-customers

[52] Giovane CM Moura, John Heidemann, Moritz Müller, Ricardo de O. Schmidt, and Marco Davids. [n.d.]. When the dike breaks: Dissecting DNS defenses during DDoS. In *IMC 2018*.

[53] Giovane C. M. Moura, Ricardo de Oliveira Schmidt, John Heidemann, Wouter B. de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. [n.d.]. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *IMC 2016*.

[54] Wolfgang Mühlbauer, Anja Feldmann, Olaf Maennel, Matthew Roughan, and Steve Uhlig. [n.d.]. Building an AS-topology model that captures route diversity. *SIGCOMM 2006* ([n. d.]).

[55] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. [n.d.]. DOTE: Rethinking (Predictive) WAN Traffic Engineering. In *NSDI 2023*.

[56] Maxime Piraux, Louis Navarre, Nicolas Rybowski, Olivier Bonaventure, and Benoit Donnet. [n.d.]. Revealing the evolution of a cloud provider through its network weather map. In *IMC 2022*.

[57] Matthew Prince. 2013. Load Balancing without Load Balancers. https://blog.cloudflare.com/cloudflares-architecture-eliminating-single-p/

[58] Matthew Prince. 2013. The DDoS That Knocked Spamhaus Offline (And How We Mitigated It). https://blog.cloudflare.com/the-ddos-that-knocked-spamhaus-offline-and-ho

[59] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. [n.d.]. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *NSDI 2012*.

[60] Duncan Riley. 2023. Internet's rapid growth faces challenges amid rising denial-of-service attacks. https://siliconangle.com/2023/09/26/internets-rapid-growth-faces-challenges-amid-rising-ddos-attacks

[61] ASM Rizvi, Leandro Bertholdo, João Ceron, and John Heidemann. [n.d.]. Anycast Agility: Network Playbooks to Fight DDoS. In *USENIX Security Symposium 2022*.

[62] Patrick Sattler, Juliane Aulbach, Johannes Zirngibl, and Georg Carle. [n.d.]. Towards a Tectonic Traffic Shift? Investigating Apple's New Relay Network. In *IMC 2022*.

[63] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. [n.d.]. PEERING: Virtualizing BGP at the Edge for Research. In *CoNEXT 2019*.

[64] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. [n.d.]. Engineering Egress with Edge Fabric: Steering Oceans of Content to the World. In *SIGCOMM 2017*.

[65] William Sentosa, Balakrishnan Chandrasekaran, P Brighten Godfrey, Haitham Hassanieh, and Bruce Maggs. [n.d.]. DCHANNEL: Accelerating Mobile Applications With Parallel High-bandwidth and Low-latency Channels. In *NSDI 2023*.

[66] Pavlos Sermpezis and Vasileios Kotronis. 2019. Inferring catchment in internet routing. (2019).

[67] Yoav Shechtman, Amir Beck, and Yonina C Eldar. 2014. GESPAR: Efficient phase retrieval of sparse signals. *IEEE transactions on signal processing* (2014).

[68] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. [n.d.]. Cost-Effective Cloud Edge Traffic Engineering with Cascara. In *NSDI 2021*.

[69] RIPE NCC Staff. 2015. RIPE Atlas: A Global Internet Measurement Network. *Internet Protocol Journal* (2015).

[70] Jan-Philipp Stauffert, Florian Niebling, and Marc Erich Latoschik. 2018. Effects of latency jitter on simulator sickness in a search task. In *2018 IEEE conference on virtual reality and 3D user interfaces (VR)*.

[71] Chris Stokel-Walker. 2021. Case study: How Akamai weathered a surge in capacity growth. https://increment.com/reliability/akamai-capacity-growth-surge/

[72] Subspace. 2022. Optimize Your Network on Subspace. https://subspace.com/solutions/reduce-internet-latency

[73] Vytautas Valancius, Bharath Ravi, Nick Feamster, and Alex C Snoeren. [n.d.]. Quantifying the Benefits of Joint Content and Network Routing. In *SIGMETRICS 2013*.

[74] Verizon. 2020. Edgecast. https://verizondigitalmedia.com/media-platform/delivery/network/

[75] VULTR. 2023. VULTR Cloud. https://vultr.com/

[76] Ye Wang, Hao Wang, Ajay Mahimkar, Richard Alimi, Yin Zhang, Lili Qiu, and Yang Richard Yang. [n.d.]. R3: resilient routing reconfiguration. In *SIGCOMM 2010*.

[77] David Wetherall, Abdul Kabbani, Van Jacobson, Jim Winget, Yuchung Cheng, Charles B Morrey III, Uma Moravapalle, Phillipa Gill, Steven Knight, and Amin Vahdat. [n.d.]. Improving Network Availability with Protective ReRoute. In *SIGCOMM 2023*.

[78] Matthias Wichtlhuber, Eric Strehle, Daniel Kopp, Lars Prepens, Stefan Stegmueller, Alina Rubina, Christoph Dietzel, and Oliver Hohlfeld. 2022. IXP scrubber: learning from blackholing traffic for ML-driven DDoS detection at scale. In *Proceedings of the ACM SIGCOMM 2022 Conference*. 707–722.

[79] Jing'an Xue, Weizhen Dang, Haibo Wang, Jilong Wang, and Hui Wang. [n.d.]. Evaluating Performance and Inefficient Routing of an Anycast CDN. In *International Symposium on Quality of Service 2019*.

[80] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. [n.d.]. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *SIGCOMM 2017*.

[81] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, and Xiaowei Yang. [n.d.]. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *SIGCOMM 2021*.

[82] Zheng Zhang, Ming Zhang, Albert G. Greenberg, Y. Charlie Hu, Ratul Mahajan, and Blaine Christian. [n.d.]. Optimizing Cost and Performance in Online Service Provider Networks. In *NSDI 2010*.

[83] Zhilong Zheng, Yunfei Ma, Yanmei Liu, Furong Yang, Zhenyu Li, Yuanbo Zhang, Jiuhai Zhang, Wei Shi, Wentao Chen, Ding Li, et al. [n.d.]. Xlink: QoE-Driven Multi-Path QUIC Transport in Large-Scale Video Services. In *SIGCOMM 2021*.

[84] Minyuan Zhou, Xiao Zhang, Shuai Hao, Xiaowei Yang, Jiaqi Zheng, Guihai Chen, and Wanchun Dou. [n.d.]. Regional IP Anycast: Deployments, Performance, and Potentials. In *SIGCOMM 2023*.

[85] Jiangchen Zhu, Kevin Vermeulen, Italo Cunha, Ethan Katz-Bassett, and Matt Calder. [n.d.]. The Best of Both Worlds: High Availability CDN Routing Without Compromising Control. In *IMC 2022*.

A Further Results

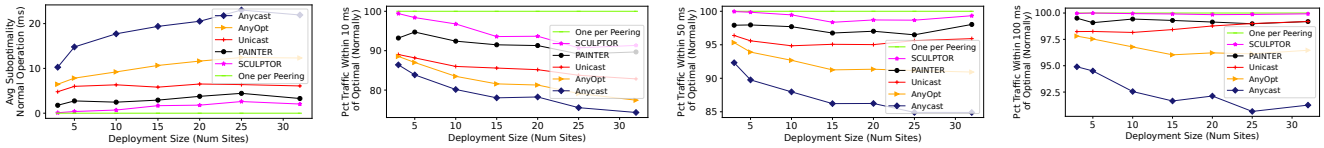
A.1 Steady State Latency

A.2 Latency During Link/Site Failure

A.2.1 Internet Deployment

A.2.2 Simulations

A.3 Infrastructure Utilization Further Results



(a) (b) (c) (d)

Figure 8: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of their optimal latency during normal operation.

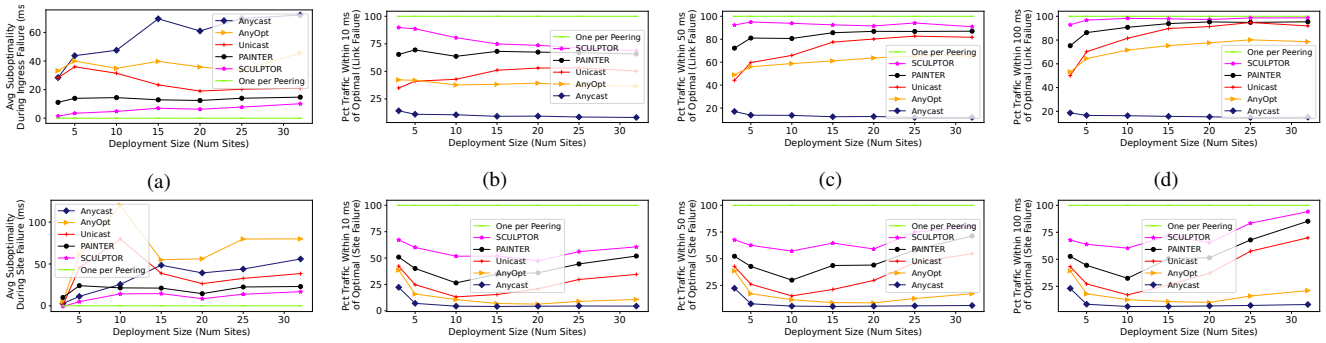


Figure 10: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of One-per-Peering latency during both link and site failures.

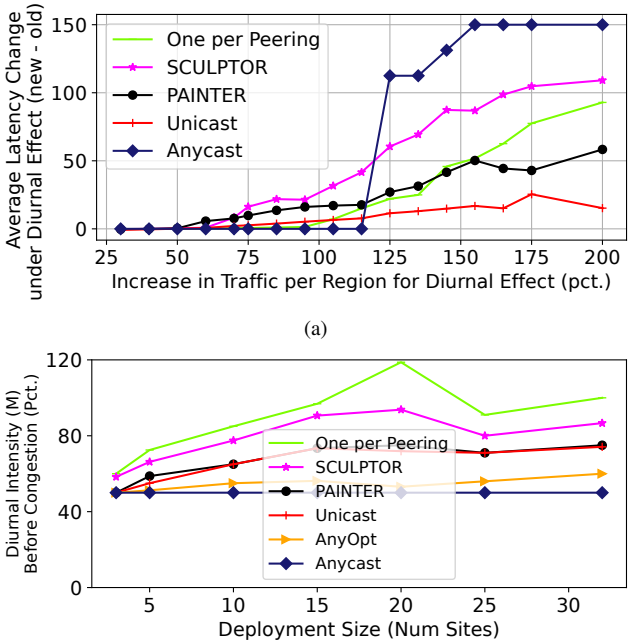


Figure 11: SCULPTOR improves infrastructure utilization under diurnal traffic patterns so that Service Providers can underprovision compared to peak loads.