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 a dynamic public Internet to route user traffic. To enhance reliability to dynamic events such as failure and DDoS attacks, large service providers overprovision to accommodate peak loads and reactively activate emergency systems for shifting excess traffic. We take a different approach with SCULPTOR, which proactively uses a service provider's global resources to handle dynamic scenarios without excessive overprovisioning. SCULPTOR ensures users have many interdomain routes to deployments, and moves excess traffic to backup paths to avoid overloading 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 28% more efficiently than other solutions, reduces overloading on links during site failures by 17% on average, and enables service providers to achieve low latency objectives for up to 17% more traffic.

1 Introduction

Compiled on 2024/07/20at 10:03:45

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 [43]) and as performance requirements become tighter (*e.g.,* virtual reality requires $\leq$ 10 ms round trip latency [60] and $\leq$ 3 ms jitter [81]). Meeting these requirements is especially challenging in a dynamic Internet. There are many changing conditions outside of Service Providers control that Service Providers have to prepare for such as peering link/site failures [27, 58], DDoS attacks [63, 89, 70], flash crowds [37, 48, 51], and route changes [61, 56, 32, 57, 65, 58]. Critically, such changes can cause resource *overload* if resources cannot handle new traffic volumes induced by the change. This overload can lead to degraded service for user traffic [62, 68, 44] making it difficult for Service Providers to satisfy the increasingly stringent performance and reliability demands of evolving Internet use cases.

The deployments' physical resources (*e.g.,* peering links, sites) are provisioned to accommodate demands from (ingress) and to (egress) user networks. Service Providers shift changing egress demands across links to minimize latency without overloading resources [74, 91, 33, 47]. However, it is difficult to similarly shift changing ingress demands 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 exposes paths that can handle changing loads. Current approaches give users at least one low latency route to Service Providers [8, 92, 43], but do not consider whether resources on these routes can support changing loads (§2.3).

Service Providers may respond to changing conditions by (a) proactively overprovisioning resources [82, 2, 53] or (b) reactively draining overloaded resources and moving some of that traffic to other resources with free capacity [27, 58]. However, these approaches do not work. We demonstrate that overprovisioning resources can require overprovisioning rates as high as 70% (§2.3). Draining traffic from overloaded links/sites either requires moving users to routes with free capacity (*e.g.,* via DNS [27]) or changing prefix announcements that are also advertised via other healthy links/sites [63, 58, 71], so that traffic destined to those prefixes is forced to arrive on other links/sites after BGP converges (within tens of seconds [96]). However, there is no guarantee that existing routes have free capacity to handle excess traffic (§5.3), and changing BGP announcements is dangerous/ineffective [27, 53] (§2.3).

To help Service Providers meet stricter performance requirements in a highly dynamic Internet, we designed a sys-

tem — SCULPTOR (**S**couring **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 avoids overloading at links/sites. Our key insight is to quickly search over the large advertisement configuration search space by modeling how different strategies perform, while predicting very little about the actual paths taken under different configurations since predicting interdomain paths is hard and measuring them is slow.

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) subject to resource constraints 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 the parallelism that gradient descent admits (§3.4). This modeling enables SCULPTOR to assess 13M configurations (6,000× more than other solutions) while only measuring tens (§5.5).

Second, we prototype and evaluate our framework in a system, SCULPTOR, at Internet scale using the PEERING testbed [73], which is now deployed at 32 Vultr cloud locations [86]. 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 to use their deployments more efficiently than using other advertisement strategies, 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 3% in steady state (§5.2), 11% during link failure, and 17% during site failure. SCULPTOR also reduces overloading on links during site failures by 17% on average, giving Service Providers more confidence that services will still be available during partial failure (§5.3). Hence, SCULPTOR can serve more traffic with less overloading and do so with almost the same latency that an optimal (but unreasonably expensive) solution can.

Providing good backup paths to handle changing 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 3× expected traffic volume, drastically reducing the amount of overprovisioning that Service Providers need, thus reducing costs. SCULPTOR can also handle large diurnal swings of almost 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 to hundreds of geo-distributed sites. 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 which have an aggregate capacity. Service Providers also connect to other networks at sites via dedicated links or shared IXP fabrics. 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 very high utilization [27, 14]. Resources can also fail completely due to, for example, physical failure and misconfiguration.

2.1 Tighter Requirements, Variable Conditions

Evolving Internet use cases 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 [43] which require very high reliability and are predicted to be a $60B industry by 2027 [36]. Similarly, gaming, an immersive application that requires latency within 50 ms [60], is an industry that now generates more revenue than the music and movie industries *combined* [6]. Moreover, Service Providers that traditionally offered less latency-sensitive services (*e.g.,* CDNs hosting static content) are increasingly pivoting to offering more latency-sensitive services like compute [17, 25, 24].

Despite their efforts to provide low latency and ultra-high reliability to support evolving Internet use cases [58, 40, 31, 23, 46, 59, 43, 53], Service Providers still face challenges in providing reliable performance especially under changing conditions caused by factors *outside* their network. For example, peering disputes can lead to congestion on interdomain links/inflated paths [27, 19, 22], and DDoS attacks still bring down sites/services [70, 62, 68, 44] despite the considerable effort in mitigating DDoS attack effects [63, 89]. 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 for than inter-datacenter demands [51, 65, 14]. Operators regularly report these traffic surges on social media and blog posts [37, 48, 61, 56, 32, 57].

2.2 Routing Traffic to Service Providers

A part of offering low latency, reliable services to users is routing service traffic over a low latency path with sufficient capacity to the deployment. Service Providers lack full control of which interdomain path traffic takes since 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 (*i.e.,* "expose paths") in different ways to increase the chance of there being good paths for users. Today, Service Providers either use `unicast` prefix advertisements to direct users to specific sites [45, 74, 12], or use `anycast` prefix advertisements to provide relatively low latency and high availability at the expense of some control [8, 42, 96, 95].

`Anycast`, where Service Providers advertise a single prefix to all peers/providers at all sites, offers high availability following failures since BGP automatically reroutes most traffic to avoid the failure after tens of seconds [96]. Prior work shows that this availability comes with higher latency in some cases [52, 42]. `Unicast` can give clients lower latency than `anycast` by advertising a unique prefix at each site [12] but can suffer from reliability concerns [43].

Recent work proposes/studies hybrids of `unicast` and `anycast` which achieve better latency and/or reliability [8, 92, 95, 43]. This class of solutions advertises different prefixes to subsets of all peers/providers to offer users a lower latency path than `anycast` in cases where `anycast` paths are inflated. 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` (multiple prefixes, selectivity).

2.3 Current Approaches Do Not Consider Changing Conditions

Existing approaches to expose low latency paths do not, however, plan for changing traffic conditions. These approaches only strive to give users low latency *primary* paths. Figure 1 shows how a certain dynamic scenario, 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). In Section 5 we show that this overload happens in practice for systems that provide very low latency from users to Service Provider networks such as `AnyOpt` [92] and `PAINTER` [43] since those systems fail to plan for changing conditions.

Instead of planning for changes, the current state-of-practice to handle changing conditions is to reactively drain traffic from overloaded links/sites either by redirecting users to backup paths (*e.g.,* with DNS [27]) or by *withdrawing* prefix announcements while still advertising them via other healthy links/sites [63, 58, 71]. After BGP reconverges the traffic destined to those prefixes then arrives on other links/sites advertising those prefixes. We demonstrate in Section 5.3 that there may be no existing backup route that can handle excess traffic, so solutions like FastRoute do not work [27].

Inducing new routes by changing BGP announcements is error-prone [53], post-hoc, unpredictable (as BGP is unpredictable), and ineffective. For example, `TIPSY` requires live BGP reconfiguration during overload [58], which is dangerous considering BGP's global blast-radius [53].

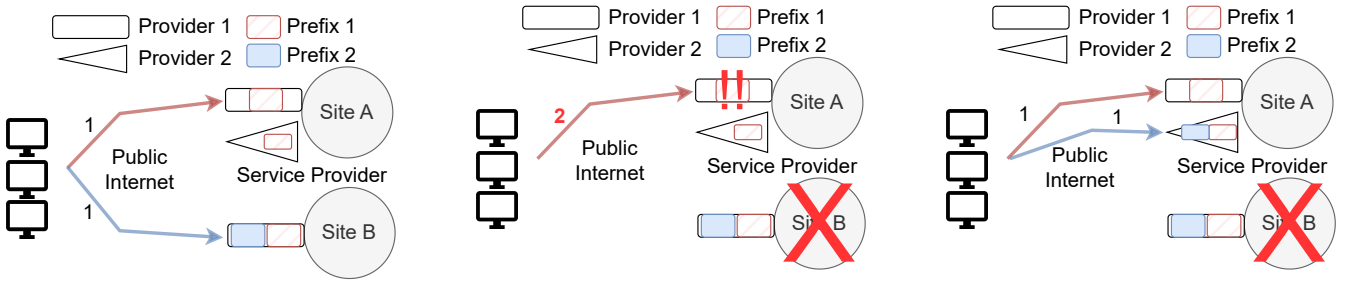
Another common solution to handling changing conditions is to overprovision resources to handle predicted peak loads [82, 2, 53], but Figure 2 demonstrates that doing so can incur excessive costs. Microsoft and Meta also stated that solely installing extra capacity without shifting traffic is not a feasible solution [27, 74].

Figure 2 computes differences in peak utilization on links between different successive time periods, using longitudinal link utilization data from OVH cloud [66]. We split the dataset into successive, non-overlapping 120-day planning periods and compute the 95^{th} percentile link utilization ("near-peak load") for each link and for each period. We then simulate assigning future capacities from period to 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. We then compute the average link utilizations in the next period and the total number of links on which we see overloading.

Figure 2 plots the median utilization across links and periods and the number of congestive events as we vary the overprovisioning factor. Figure 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 roughly 50% utilization. Using backup paths instead to handle relatively rare peak loads can help achieve both high utilization and low overloading.

2.4 Key Differences in Interdomain

Prior work finds ways to route intradomain traffic under changing conditions (like link failure) [87, 54, 10, 39, 14]. 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 resources (paths) before placing traffic on them. We explicitly address



(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).

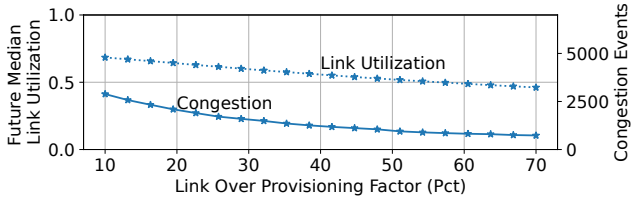


Figure 2: Planning for peak loads to avoid overloading requires inefficient overprovisioning.

the former problem and leave the latter problem as future work.

We model paths from users to the Service Provider through the public Internet as non overlapping, except at the peering link through which each path ingresses to the Service Providers' deployment. Users share the capacity of those peering links. We assume the peering link is the bottleneck of each path. We consider routing 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 overloading [74, 91, 33].

Although we do not explicitly address problem (b) above, we implicitly address it by finding interdomain routing strategies that can handle changing conditions. Paths with unexpectedly low bottleneck capacities can be seen as a "changing condition" and one could use congestion-detection mechanisms along paths to trigger traffic shifts to other, healthier (backup) paths. Other prior work on ingress [43] and egress traffic engineering [91, 33, 74, 47] also does not explicitly address this problem.

Service Providers lack interdomain control for their ingress traffic, but there is hope that they can handle changing conditions. A Service Provider has enough global capacity to handle even large changes in traffic or sudden traffic shifts from one link to another [14, 1, 38], and should do so without reconfiguring BGP (§2.3). Figure 1c intuitively suggests that this planning is possible — advertising prefix 2 to provider 2 at site A a priori allows the Service Provider to split traffic between the two links during failure *without reconfiguring BGP*, avoiding overload, excessive overprovisioning, and risky reconfiguration.

2.5 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. However, making paths available uses IPv4 prefixes, which are monetarily expensive and pollute BGP routing tables. IPv6 is not a good alternative as there are fewer IPv6 paths since IPv6 peering is less common, IPv6 is not universally supported, and IPv6 routing entries take 8× the amount of memory to store in a router so would pollute global routing tables even more. Prior work noted the same but found that advertising around 50 prefixes was acceptable [92, 43], and most Service Providers advertise fewer than 50 today according to RIPE RIS BGP data [80]. 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 since measuring paths takes time. The number of subsets is on the order of 2 to the number of peers/providers Service Providers have which, for many Service Providers is $> 2^{1,000}$. 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 changing 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 changing 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 (10M constraints, 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 that we consider (millions of evaluations). Instead, we estimate latency by predicting paths. 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 computing how correlations among those networks impact utilization *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 overloading (§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 under changing conditions from user networks to the Service Provider subject to capacity constraints on links. Adding capacity constraints on sites (*e.g.,* servers) in addition to links would be a straightforward extension. We encourage low latency under changing conditions by minimizing latency during both steady state and single link/site failures, as minimizing for both intuitively encourages good primary and backup routing options.

We assume the Service Provider has some technology for directing traffic towards prefixes. Examples include DNS [12, 8, 45], multipath transport [69, 20], or control-points at/near user networks [72, 43]. DNS offers slow redirection due to caching [43] but is the most readily deployable by the largest number of Service Providers, 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 [5] and Ubuntu 22 [28], and MPQUIC can be installed on any device in user space with applications [94]. SCULPTOR assumes that 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 steady state traffic volumes, $v_{\texttt{UG}}$, 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 Service Provider peering links l enumerated as $L_{\texttt{UG},l}$. (Having these measurements is a reasonable assumption [9, 12, 21, 92, 43]). 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 [92, 43], 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):

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

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/conditions that are more likely.

We cannot just compute $\mathcal{L}_{\texttt{UG}}(A,N)$ 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 overloading.

A given advertisement configuration during normal operation exposes a set of paths $P(A,N)$ to UGs 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 the allocation of UG traffic to paths $w_{UG,l}$.

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

Latency for a UG in Equation (1) is then the weighted sum of path latencies $L_{\text{UG},l}$ according to this optimal allocation: $\mathcal{L}_{\text{UG}}(A,N) = \sum_l L_{\text{UG},l} w^*_{\text{UG},l}$.

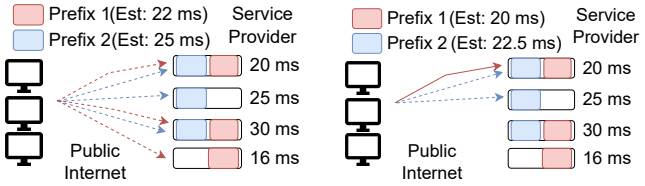
The minimization term in Equation (2) is the sum of latency and maximum link utilization (MLU), weighted by a parameter β. β represents a tradeoff between using uncongested links/sites and low propagation delay and is set by the Service Provider based on their goals. We first solve Equation (2) with $MLU = 1$ to see if we can allocate traffic to paths with zero overloading. However such overload-free solutions do not always exist for arbitrary advertisement strategies A.

The first/second constraints in Equation (2) require 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 then 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 overloading. (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 advertisement strategies in Equation (1), even under changing conditions.

3.3 Accommodating Internet 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.



(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). Apriori, no prefix is advertised. We then advertise the red prefix and measure the path 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 advertising the blue prefix, saving time.

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 20 minutes. Hence we model, instead of measure, UG paths and improve this model over time through relatively few 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 [64, 4, 58, 71].

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 [92, 76, 43]. 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 that prefix is advertised to for a UG are equally likely. 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).

3.3.2 Capacity-Free UG Latency Calculation

To assign traffic to prefixes we should 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

UG*s* 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 UG*s* experience ($\sum_{\text{UG}} \tilde{\mathcal{L}}_{\text{UG}}(A,N)\ v_{\text{UG}}$) are 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.

3.3.3 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 Equation (2) which would take too long. 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 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 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 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.

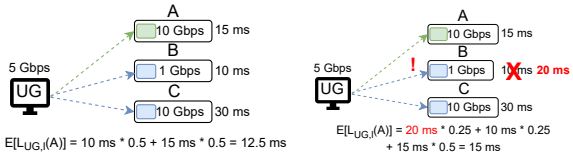


Figure 4: A UG has a path to two prefixes, green and blue. The green prefix is only advertised via ingress A and the blue prefix is advertised via two ingresses B and C, each of which are equally likely for this UG. If the UG prefers B over C, we assign the client to the blue prefix (10 ms) while if 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).

3.4 Solving with Gradient Descent

Even with our efficient objective function approximations, 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. We do not compute all entries of the gradient of Equation (1) since there are too many to compute. Instead, we use subsample entries of the gradient and track the largest entries. These methods are also used in other settings [77, 49].

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 under changing conditions, which is more important than having the *lowest* steady state latency.

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 Routing Models

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 [73], 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 [43] 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 [11]). Vultr likely peers with networks with which it exchanges a significant amount of traffic [74], 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, 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.

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.

We implement SCULPTOR with Nesterov's Accelerated Gradient Descent in Python [**?**]. We set $\alpha_l = 4.0$ and set the learning rate to 0.01 with decay over iterations. We solve **??** with Gurobi.

4.2.1 Experiments in the Internet

We assess how SCULPTOR performs on the Internet using RIPE Atlas probes [79], 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).

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 [8, 55, 43, 65]. (Service Providers could also scale compute.)

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 that 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 in 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 [73], and measure actual ingress links to those prefixes using traceroutes from RIPE Atlas probes [79]. 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.

The second method we use to determine ingress links is simulating user paths by assuming we know all user routing 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 preference assignments), 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 [92].

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 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). We then demonstrate that `SCULPTOR` can handle changing conditions such as flash crowds or diurnal patterns 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 [30, 15, 8, 85, 67, 90, 95].

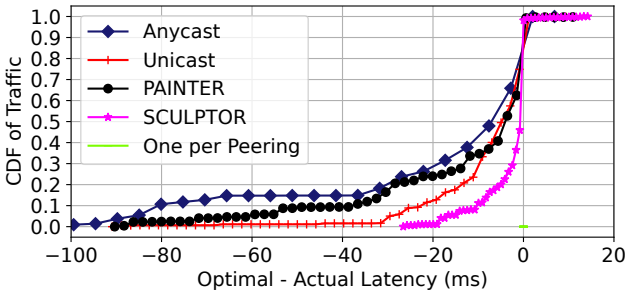
`unicast`: A single prefix announcement to all peers/providers at each site (one per site). Another common strategy used in today's deployments [45, 12]. `Unicast` also provides an upper bound on FastRoute's performance.

`AnyOpt/PAINTER`: Two proposed strategies for reducing steady state latency compared to `anycast` [92, 43].

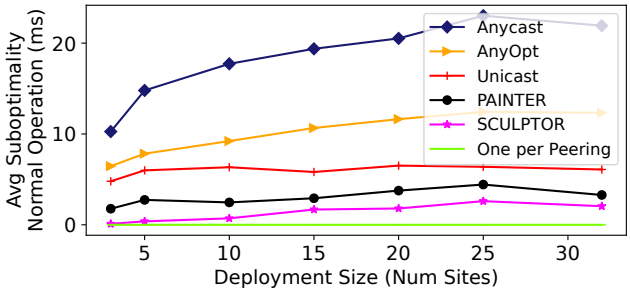
`One-per-Peering`: A unique prefix advertisement to each peer/provider, so many possible paths are always available from users. This solution serves as our performance upper-bound, even though it is prohibitively expensive. (We do not know an optimal solution with fewer prefixes.)

For our evaluation on the public Internet (§4.2.1), we limit the scale of our deployment to 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. 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. We use 12 prefixes, which is low, since we do not have many prefixes. 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 ingresses.

For our simulated evaluations (§4.2.2), 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 10 times with random subsets of `UG`*s*, `UG` demands, and routing preferences. We use one tenth of the number of peers/providers as the number of prefixes in our budget for all solutions (except



(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 (5b).

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 [92, 43]. Over all studied deployments, we consider paths from clients in 42k /24s to 868 peers/providers.

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 overloading 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.

5.2 Lower Latency in Steady State

5.2.1 Internet Deployment

We compute the latency that 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 realistic routing. SCULPTOR is only 3.0 ms worse than (the unreasonably expensive) One-per-Peering solution whereas the next best solution, unicast, is 8.4 ms worse. SCULPTOR also serves 88.8% of traffic within 10 ms of One-per-Peering, whereas the same is true for only 69.5% of traffic with a unicast solution.

PAINTER does not do as well as unicast here (but does in simulation), which may be because of our relatively low budget of 12 prefixes. We use more prefixes in our simulations, and prior work noted that the most significant gains when using roughly 40 prefixes [43].

5.2.2 Simulations

Figure 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 2.5 ms worse than (the undeployable) One-per-Peering. The next-best solution (PAINTER) is on average 1.2 ms and 2.4 ms worse than SCULPTOR.

anycast comparatively performs the worst with UGs being on average 17.9 ms worse than One-per-Peering whereas SCULPTOR performs the best with UGs being 1.3 ms worse than One-per-Peering. Interestingly, unicast (6.1 ms worse than One-per-Peering) actually performs better than AnyOpt (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 Appendix B.1). SCULPTOR has on average 94.8% traffic within 10 ms of the One-per-Peering solution, 99.2% within 50 ms, and 99.9% within 100 ms. These percentages compare to the next-best solution, PAINTER, which has on average 91.7% traffic within 10 ms of the One-per-Peering solution, 97.4% within 50 ms, and 99.2% within 100 ms.

5.3 Better Resilience to 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. Figure 6 demonstrates that SCULPTOR shifts traffic without overloading alternate links/sites more effectively than any other solution *without reconfiguring BGP* which could cause further failure (§2.3).

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 UGs who use that component in steady state. For

example, if the Tokyo site fails, we report on the post-failure latency of UGs served from Tokyo before the failure and not of other UGs.

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.

5.3.1 Internet Deployment

Figure 6a demonstrates that SCULPTOR offers lower latency paths for more UGs during single link failure in realistic routing conditions. On average, SCULPTOR is only 7.2 ms higher latency than One-per-Peering while the next best solution, unicast, is 12.6 ms worse than One-per-Peering. PAINTER struggles to find sufficient capacity for UGs, with 40.8% overloading.

Single site failures (shown in Appendix B.2) show similar results. On average, SCULPTOR is 18.8 ms higher latency than One-per-Peering while the next best solution, unicast, has 16% overloading and PAINTER has 86% overloading during site failure.

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 6b and Figure 6c (further results are in Appendix B.2).

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

When looking at routing inefficiency, SCULPTOR has 78.2% of traffic within 10 ms of the One-per-Peering solution on average, 93.0% within 50 ms, and 97.2% within 100 ms during link failure. These statistics compare to the next-best solution, PAINTER, which has 66.9% within 10 ms, 83.4% within 50 ms, and 90.4% within 100 ms. Site failures show similar trends (full results are in Appendix B.2).

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 [74]. Large Service Providers recently emphasized that these seemingly small gains make large differences [26, 88].

5.4 Efficient Infrastructure Utilization

One response to increased link/site utilization is to install more capacity so that there is sufficient headroom to satisfy peak demand. **??** 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 overloading, where peak load is distributed according to two realistic traffic patterns: flash crowds and diurnal effects.

5.4.1 Methodology

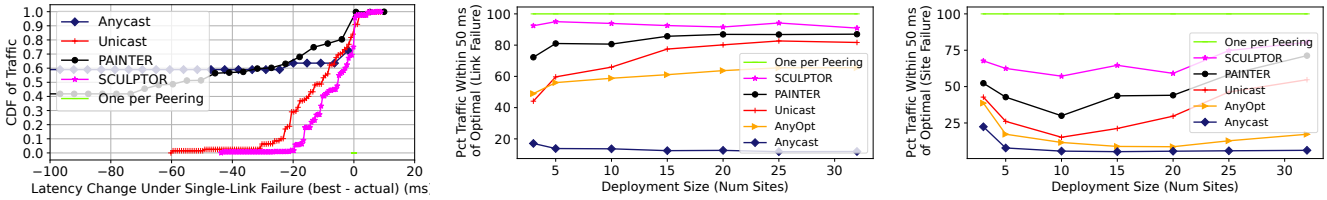
We define a flash crowd as a transient traffic increase at a single site. Examples of flash crowds include content releases (games, software updates) that spur downloads in a particular region. Since increased demand is highly localized, we can spread excess demand to other sites if paths to those sites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

To generate **??**, 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 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 overloading when we (separately) scale volume in any region. For example, if Atlanta experiences overloading with a 60% increase in traffic for Atlanta clients, but no other region does, we call $M = 60\%$ the critical value of M.

Our diurnal analysis in Figure 7b 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 large Service Provider demonstrates that these effects can cause large differences between peak and mean site volume [14]. 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 $0.1M$. The full curve is shown in Appendix B.3. Similar to our flash crowd analysis, we increase M until at least one link experiences overloading at least one hour of the day.

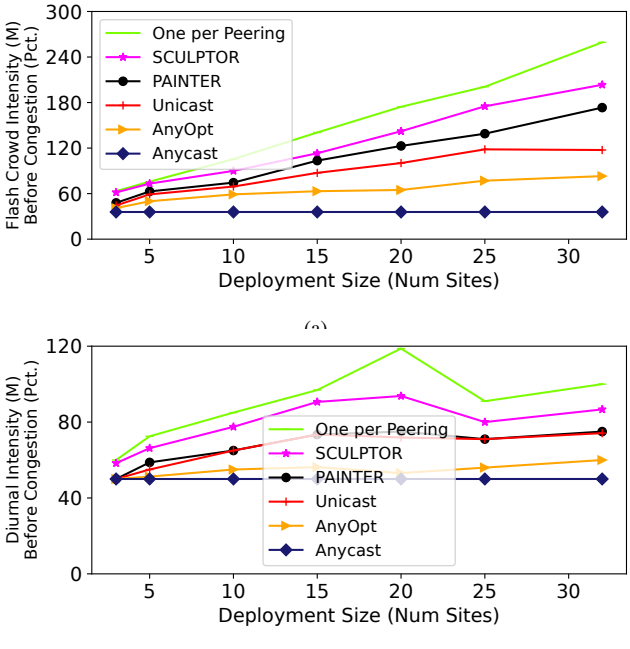
5.4.2 Results

Figure 7 plots the average over simulations of critical M values that cause overloading for each deployment size under flash crowds and diurnal effects computing using simulated deployments.



(a) (b) (c)

Figure 6: SCULPTOR improves resilience to failure both on the Internet (6a) and in simulation (6b, 6c).



(a)

(b)

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

Figure 7a shows that SCULPTOR finds ways to route more traffic during a flash crowd without overloading than other solutions. SCULPTOR can handle single-site flash crowds at 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 (22% more than PAINTER). Figure 7b shows that SCULPTOR also handles more intense diurnal traffic swings, allocating traffic to paths without overloading for between 11% and 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. SCULPTOR usually increases latency by less than 5 ms on average up to the critical M value. 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 [38, 34, 33].

Prior work noted similar benefits by satisfying user requests in distant sites [14], 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 [74] and Fastroute [27]) do not always shift traffic on their backbones since 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 quickly predicts how other advertisement strategies perform so that it can pick the best ones. This prediction is an approximation of true performance (§3.1) and the question we aim to answer

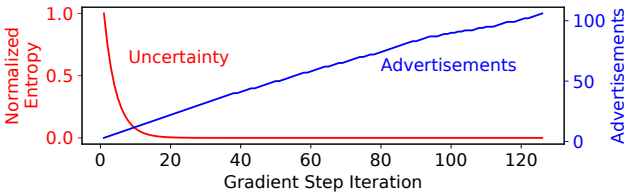


Figure 8: SCULPTOR's model entropy decreases exponentially with few path measurements.

here is why those approximations work.

First, SCULPTOR compares far more advertisement strategies than the other solutions. In our 32 site deployments over 200 gradient steps SCULPTOR estimates latencies in approximately 13M scenarios across every UG. PAINTER only considers thousands (2k), and AnyOpt considers 1k (a configurable number). Tango [7], a system that exposes better paths by advertising prefixes, also measures roughly thousands of different advertisements but that system is meant for a different setting and so is not directly comparable to SCULPTOR.

PAINTER has a simple method 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 (§3.5) or for strategies that SCULPTOR thinks will yield good performance (§3.4). Figure 8 shows how SCULPTOR's model entropy exponentially decreases as it makes these measurements on the Internet. (We plot the maximum entropy of unmeasured advertisement strategies (§3.5).) Intuitively, this quick convergence is likely since latency is much easier to predict than paths — UG*s* have similar latency to many ingresses at the same site, so being uncertain about which paths UG*s* take usually does not matter from a latency optimization perspective.

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 [91, 74, 47]. 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 [93, 78, 14]. 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 [8, 55, 27, 92, 96, 95, 43], but optimizes for steady state conditions. SCULPTOR optimizes for changing conditions, reducing costs and enhancing performance compared to these solutions as we demonstrate in Section 5. FastRoute finds a simple way to handle traffic changes by shifting excess traffic to other sites [27]. Our solution is more complex, but more effective as shown by our comparisons to unicast (§5.3). Pecan [84] and Tango [7] 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, 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 [83, 35, 3, 16, 51]. 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 [71], 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 changing conditions over thousands of peering links.

Microsoft withdraws prefixes to mitigate overloading on links [58]. However, as they demonstrate in their paper, such actions are unpredictable and can lead to further overloading. SCULPTOR instead plans ahead, minimizing overloading 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 [94]. DChannel uses both of 5G's high bandwidth and low latency channels to optimize web performance [75]. TGaming [13] 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 [23, 31, 46, 53]. 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 [87, 54, 10, 39, 14]. 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 Internet services play an increasingly critical role in our lives, 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 is a step towards providing the truly resilient, performant service that more people increasingly need.

References

[1] Satyajeet Singh Ahuja, Vinayak Dangui, Kirtesh Patil, Manikandan Somasundaram, Varun Gupta, Mario Sanchez, Guanqing Yan, Max Noormohammadpour, Alaleh Razmjoo, Grace Smith, et al. [n.d.]. Network entitlement: contract-based network sharing with agility and SLO guarantees. In *SIGCOMM 2022*.

[2] 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*.

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

[4] 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*.

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

[6] 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

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

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

[9] 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*.

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

[11] Nikolaos Chatzis, Georgios Smaragdakis, Anja Feldmann, and Walter Willinger. 2013. There is more to IXPs than meets the eye. *SIGCOMM Computer Communication Review* (2013).

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

[13] 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).

[14] 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*.

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

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

[17] Cloudflare. 2024. Cloudflare Workers. https://workers.cloudflare.com/

[18] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems* 2, 4 (1989), 303–314.

[19] 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

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

[21] 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*.

[22] 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.

[23] 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*.

[24] Edgecast. 2020. The CDN Edge brings Compute closer to where it is needed most. https://edgecast.medium.com/the-cdn-edge-brings-compute-closer-to-where-it-is-needed-most-d4a3f4107b11

[25] Fastly. 2024. Fastly Compute. https://fastly.com/products/edge-compute

[26] 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*.

[27] 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*.

[28] 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

[29] Jonas Geiping, Micah Goldblum, Phillip E Pope, Michael Moeller, and Tom Goldstein. 2021. Stochastic training is not necessary for generalization. *arXiv preprint arXiv:2109.14119* (2021).

[30] 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*.

[31] 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*.

[32] 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

[33] 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*.

[34] 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*.

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

[36] 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

[37] 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

[38] 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*.

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

[40] 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*.

[41] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[42] 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*.

[43] 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*.

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

[45] 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*.

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

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

[48] 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

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

[50] Haochuan Li, Jian Qian, Yi Tian, Alexander Rakhlin, and Ali Jadbabaie. 2024. Convex and non-convex optimization under generalized smoothness. *Advances in Neural Information Processing Systems* 36 (2024).

[51] 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*.

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

[53] 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*.

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

[55] 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*.

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

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

[58] 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*.

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

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

[61] 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

[62] 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*.

[63] 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*.

[64] 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.]).

[65] 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*.

[66] 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*.

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

[68] 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

[69] 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*.

[70] 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

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

[72] 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*.

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

[74] 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*.

[75] 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*.

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

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

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

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

[80] RIPE NCC Staff. 2023. RIS Live. (2023). https://ris-live.ripe.net

[81] 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)*.

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

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

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

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

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

[87] 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*.

[88] 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*.

[89] 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.

[90] 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*.

[91] 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*.

[92] 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*.

[93] 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*.

[94] 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*.

[95] 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*.

[96] 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 Optimization Extensions

In our formulation of the advertisement configuration problem (Eq. (1)), we try to minimize average latency. It could be, however, that Service Providers wish to minimize other/additional metrics. The question we try to answer here is which metrics our framework will more likely work for. We do not provide any theoretical guarantees, but give valuable intuition backed by experimental demonstrations.

Our high-level finding is that SCULPTOR works with most reasonable metrics, but that considering metrics other than latency could introduce considerable computational costs.

A.1 Generalizations

SCULPTOR tries to find an advertisement configuration that admits good assignments of user traffic to paths. Hence SCULPTOR tries to solve a multivariate optimization over both configurations and traffic assignments (we placed more explicit focus on optimizing configurations in Section 3). We now adopt this view of the optimization problem to consider metrics other than average latency.

Suppose that the advertisement configuration A can be represented by a binary vector. Each entry indicates whether we advertise/do not advertise a particular prefix to a particular peer/provider. Implementing this advertisement configuration results in routes from users to the Service Provider. The assignment of traffic to resulting routes from this advertisement configuration is given by the nonnegative real valued vector w. The entries of the vector w specify traffic allocation for each user along each route.

Although we do not know how configurations map to routes, assume that this mapping is well defined. Let R be the mapping from advertisement configurations to functions that map $\langle \texttt{prefix, UG} \rangle$ pairs to links (*i.e.,* R is a mapping that creates mappings). For example, say $R(A) = f$, and $f(p, \texttt{UG}) = l$ — this notation means that the output of an advertisement configuration A under routing R is a function f that maps prefixes p and UGs to a 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 infinity otherwise.

Now suppose the overall metric we want to minimize is G where G is a function of both configurations and traffic assignments. Examples include traffic cost, average latency, and maximum latency. With this notation, and ignoring maximum link utilization as in Equation (2) for the moment, the joint minimization of configurations and traffic assignments can be expressed as the following.

$$\begin{aligned} \min_{A,w} \quad & G(R(A), w) \\ \text{s.t.} \quad & w_{\texttt{UG},p} \geq 0 \\ & A \in \{0,1\} \\ & \sum_{p} w(p, \texttt{UG}) e(R(A)(p, \texttt{UG})) = v(\texttt{UG}) \quad \forall \texttt{UG} \\ & \sum_{R(A)(p,\texttt{UG})=l} w(p, \texttt{UG}) \leq c(l) \quad \forall l \end{aligned} \tag{3}$$

The first constraint requires that traffic assignments be nonnegative and the second requires that configurations are binary. The third constraint requires that all user traffic (UG) is assigned, and none is assigned to nonexistent paths (we assume that $\lim_{w(p,\texttt{UG})\to 0} w(p, \texttt{UG}) e(R(A)(p, \texttt{UG})) = 0$). The fourth constraint specifies that total traffic from all users towards a link, l, does not exceed the link's capacity, $c(l)$. Considering other capacity-related constraints is straightforward.

A.2 Using Gradient Descent

A.2.1 Challenges

To use gradient descent to solve Equation (3), we require that our objective function is a differentiable function of real variables which, as we have presented it, is not. Ignoring possible problems with the objective function G, there are two problems: first, that our configuration A is a binary variable and second, that the function R is not well-behaved.

To see what we mean by this second point, consider 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" in a fundamental way, or at least not differentiable everywhere. Gradient descent only converges to minima for functions whose second derivatives are locally bounded [50].

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 as in Section 3.

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 to pick an initialization where users have paths to many links/sites, and to heavily discourage routeless configurations during gradient descent.

A.2.2 Optimality and Representation Power

With relatively light conditions (bounded domains, bounded second derivatives [50]), gradient descent converges to *a* min-

ima and can converge relatively quickly with good initializations and choices of the learning rate. However, unless the objective function is convex (ours is not) then gradient descent may converge to local minima.

Before we discuss why this might be a problem in our setting, we first draw an analogy to the Deep Learning setting which has used (stochastic) gradient descent with lots of success, despite their exclusive focus on nonconvex objectives.

Gradient Descent in Deep Learning 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 over an approximately "average" gradient path [41]. 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 [29].

Functions f_α contain potentially billions of parameters and are generally non-convex. They 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 [18], despite the non-convexity. 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 very 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. Practitioners thus strive to choose functions that have sufficiently many parameters to represent complex functions, but not so many so as to overfit.

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 perform gradient descent 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".

Analogies to SCULPTOR We now return to our discussion of when/why gradient descent converges to "good" minima in SCULPTOR. In our setting, the parameters α are the configuration and traffic assignment variables since those are the parameters we perform gradient descent over. Our training data corpus consists of path/deployment metrics (*e.g.,* latency), link capacities, and traffic volumes.

One concern with convergence to a local minima is that the objective function value at that point will be much higher than it could otherwise be. 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 minima giving an average latency of, for example, 50 ms.

Deep learning identifies that one way of overcoming this challenge is to consider richer functions with more parameters. In our setting, this means increasing the number of "parameters" in both advertisement configurations and path options. Practically, this option means increasing the number of prefixes, peerings, or types of configurations. By this last point we mean, for example, using AS path prepending to potentially offer richer path options. Our investigation in **TBD: fig** confirms this intuition, as using more prefixes allowed us to converge to better solutions.

Another potential problem with converging to local minima is overfitting. However, SCULPTOR's explicit goal is to prevent overfitting. SCULPTOR aims to still provide good routes under both different traffic distributions and failures which, with our analogy, is SCULPTOR's notion of a data corpus. SCULPTOR considers various types of failures (thus training data) during gradient descent to encourage generalization to other scenarios, just as deep learning algorithms consider random batches of large corpuses of training data to converge to good solutions.

We Do Not Learn Routing with GD Another type of "learning" we discuss in Section 3 is refining our estimate of the routing function R. However, this is not descent based learning and is independent of this discussion of gradient descent. In Section 3 we model the function R as a random function, but one that follows a specific structure (preference routing model), and opportunistically remove randomness using Internet measurements. As we demonstrate in Figure 8, the randomness is essentially zero for most gradient descent iterations.

A.3 So, Which Metrics Work?

The above challenges, solutions, and intuitions are valid independent of the specific metric we care to optimize. However, certain metrics could introduce either convergence or computational challenges. Before discussing those challenges, we first present choices of G other than average latency.

A.3.1 Possible Metric Choices

Any Function of Latency: Configuring routes that minimize any function of latency are trivial extensions of SCULPTOR. The objective function would take the form $G(R(A), w) = \sum_{p,\texttt{UG}} f(L(R(A)(p,\texttt{UG}),\texttt{UG}))w(p,\texttt{UG})$ where the function f is

some arbitrary mapping on latency. The resulting problem is a linear program in w and hence convex in w. A simple variation is maximizing the amount of traffic below a latency threshold, which may be useful for enabling specific applications.

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). Hence the objective function would take the form $G(R(A), w) = \max_l \sum_{p,\texttt{UG}:G(R(A)(\texttt{UG},p)=l)} w(p, \texttt{UG})$ which can be expressed as a linear program in w.

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 $G(R(A), w) = \sum_{p,\texttt{UG}} C(w(p, \texttt{UG}))$ where C is some cost function on traffic. Examples include polynomials and exponents.

A.3.2 Convergence Concerns

We expect functions G whose gradients do not blow up in the region of interest to converge well. For configurations, A, we encourage gradients to be well-behaved with a good initialization (appendix A.2). 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, in **TBD: fig** we compare 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 struggles to find a good solution. (We could not think of why a Service Provider would choose this latter metric, but it is a possible choice.)

A.3.3 Computational Concerns

A key focus in Section 3 was finding *scalable* computational approaches to optimization, rather than worrying about their convergence. For example, instead of solving Equation (2) during gradient descent, we approximate its expectation using efficient heuristics that take advantage of the nice structure of our choice of metric G. Without these heuristics, but with a choice of G that results in a linear program with respect to w, one would have to solve potentially millions of linear programs per iteration of gradient descent.

Although such computation is not feasible for us at the problem sizes we consider, Meta recently showed that they could leverage their infrastructure to solve millions of linear programs in minutes [23]. As demonstrated by Meta, it may be reasonable to expend significant computational resources solving millions of linear programs if expending those resources leads to cost savings down the line. Hence, these computational concerns may not be problems at the scale of actual Service Providers, even though they were for us.

B Further Results

B.1 Steady State Latency

In addition to quantifying average latency during steady state, we also compute the fraction of traffic within 10 ms, 50 ms, and 100 ms of the One-per-Peering solution during steady state. Figure 9 shows how SCULPTOR compares more favorably to the One-per-Peering solution than other advertisement strategies.

B.2 Latency During Link/Site Failure

We show additional evaluations of SCULPTOR during link and site failures. **??** shows that SCULPTOR provides good resilience to site failures on the Internet, and Figure 11 shows similar results for link and site failures in our simulations.

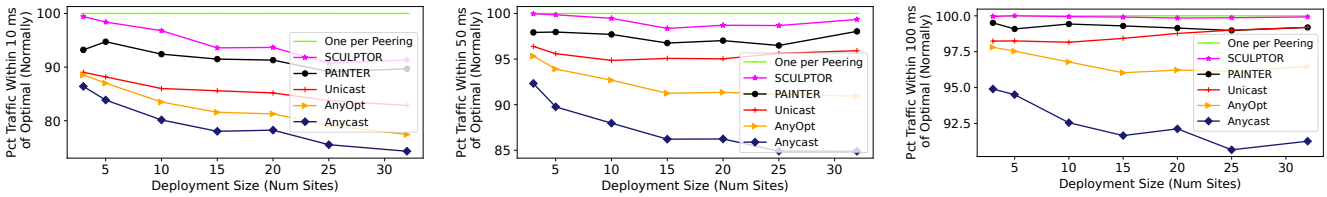
B.2.1 Internet Deployment

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.

B.2.2 Simulations

B.3 Infrastructure Utilization Further Results

In **??** we show the shape of our diurnal curve taken from observed diurnal traffic patterns at a large service provider [14]. 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.



(a) (b) (c)

Figure 9: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of their optimal latency during normal operation.

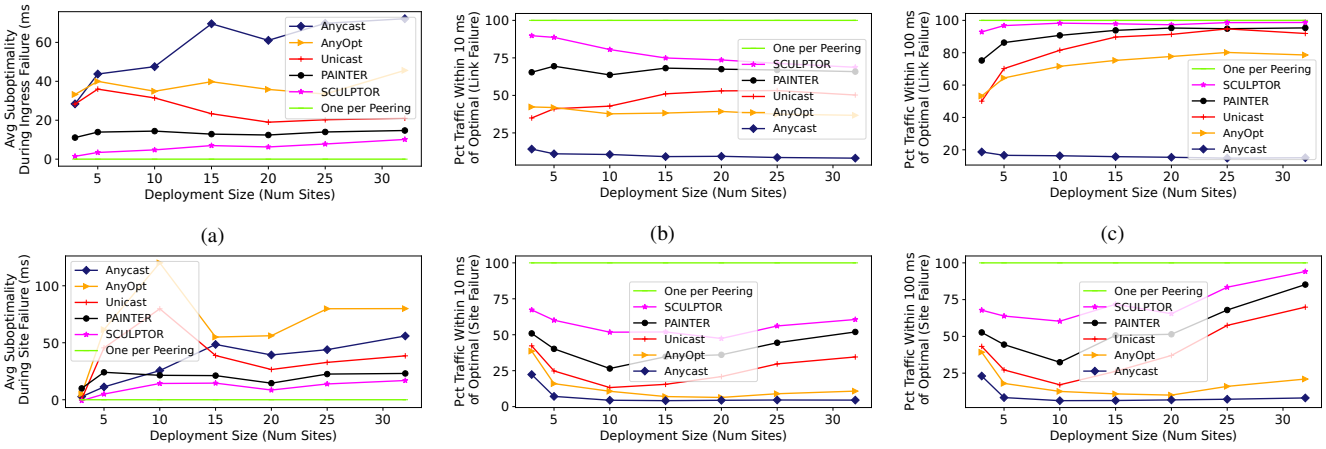


Figure 11: `SCULPTOR` lowers average latency and increases the fraction of traffic within critical thresholds of `One-per-Peering` latency during both link and site failures over many simulated deployments.