SCULPTOR Has Your Back(up Path): Carving Interdomain Routes to Services

Paper #156, 12 pages body, **TBD: 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 installs routes so that Service Providers can use their global resources to handle dynamic scenarios without excessive overprovisioning. SCULPTOR ensures users have many interdomain routes to the Service Provider, and moves excess traffic to backup paths to avoid overloading and optimize general objectives. SCULPTOR models 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 handles dynamic loads 28% larger than other solutions using existing Service Provider infrastructure, reduces overloading on links during site failures by up to 40%, and enables service providers to achieve low latency objectives for up to 17% more traffic.

1 Introduction

Compiled on 2024/09/12at 13:34:54

Cloud and content providers (hereafter Service Providers) enable diverse Internet applications used daily by billions of users, and those applications have diverse requirements. For example, enterprise services have tight reliability requirements [50, 47], whereas new applications such as virtual reality require $\leq$ 10 ms round trip latency [71] and $\leq$ 3 ms jitter [93]. It may therefore make sense for different Service Providers to allocate resources in different ways to meet these requirements to varying degrees.

Service Providers steer traffic over different end-to-end paths through the public Internet to meet these goals. For example, Service Providers shift egress (out of the Service Provider's network) demands across links to meet performance objectives [86, 103, 37, 54, 90]. However, it is difficult to modify the path traffic takes from users to the Service Provider (*i.e.,* ingress traffic) with the same precision as in the egress case since that path can only be controlled indirectly, by conducting BGP announcements and directing user traffic to different prefixes.

Complicating matters, Service Providers must meet these requirements subject to changing conditions such as peering link/site failures [30, 66], DDoS attacks [74, 101, 82], flash crowds [42, 55, 58], new applications/business priorities (LLMs), and route changes [72, 64, 36, 65, 76, 66]. Critically, such changes can cause *overload* if the service cannot handle new traffic volumes induced by the change. This overload can lead to degraded service for users, hurting reliability [73, 80, 51, 47].

To meet different goals and respond to changing conditions, Service Providers propose solutions that are either (a) too conservative, (b) too specific, or (c) too reactive. For example, Service Providers proactively overprovision resources [94, 1, 60], but we demonstrate using traces that bursty traffic patterns can require conservative overprovisioning rates as high as 70% (§2.3). Systems such as AnyOpt and PAINTER proactively set up routes to optimize specific objectives such as steady-state latency [104, 50], but those approaches have not been demonstrated to optimize other important objectives. FastRoute and TIPSY respond to changing loads to retain reliability [30, 66], but only do so reactively leading to short-term degraded performance. It is also unclear both how to combine these approaches (*e.g.,* retaining low latency under changing traffic loads) and how to apply approaches from one domain to another (*e.g.,* applying egress traffic cost reduction systems to ingress traffic [90]).

We present Service Providers with a flexible framework, SCULPTOR (**S**couring **C**onfigurations for **U**tilization-**L**oss-and **P**erformance-aware **T**raffic **O**ptimization & **R**outing), which accepts as input cost, performance, and reliability objectives and outputs BGP advertisements and traffic allocations that

help achieve those desired objectives. Service Providers can specify constraints such as link and site capacities, and optimize objectives such as maximum link utilization, transit cost, latency, and latency under partial (*e.g.,* single link) failures, and combinations thereof. SCULPTOR allows Service Providers to specify traffic patterns or failure scenarios to consider (with varying importances), but good performance also generalizes to unseen scenarios.

We provide evidence that SCULPTOR works for objectives that can be met in intradomain settings such as latency reduction, cost reduction, and maximum link utilization, and works for constraints such as link and site capacities, all at the scale of todays' Service Providers. SCULPTOR could cover a broader set of objectives, constraints, and problem sizes but we leave such formal characterization to future work.

To solve each optimization problem, SCULPTOR efficiently searches over the large BGP advertisement search space ($> 2^{10,000}$ possibilities) by modeling how different strategies perform, without having to predict the vast majority of actual paths taken under different configurations since predicting interdomain paths is hard and measuring them is slow (§3.3). SCULPTOR then optimizes these (modeled) performance metrics using gradient descent, which is appropriate in our setting due to the high dimension of the problem and the parallelism that gradient descent admits (§3.4). This modeling enables SCULPTOR to assess 13M configurations (6,000× more than other solutions) while only measuring tens in the Internet (§5.4).

We prototype and evaluate our framework at Internet scale using the PEERING testbed [85] (§4), which is now deployed at 32 Vultr cloud locations [98]. Vultr is a global public cloud that allows our prototype to issue BGP advertisements via more than 10,000 peerings. We evaluate our framework on a specific optimization problem that proactively provides cost and latency benefits, even under failure scenarios, demonstrating the framework's general utility.

We compare SCULPTOR's performance on this specific problem to that of an unreasonably expensive "optimal" solution (computing the actual optimal is infeasible). We found that, compared to other approaches, SCULPTOR increases the amount of traffic within 10 ms of the optimal by 3% in steady-state (§5.2.1), by 11% during link failure, and by 17% during site failure. SCULPTOR also reduces overloading on links during site failures by 17% (on average over all tested scenarios), giving Service Providers more confidence that services will still be available during partial failure (§5.2.2).

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 (§5.2.3). 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.

In an era of complex, global, distributed systems, Service Providers want unified frameworks especially if those frameworks provide better results. SCULPTOR provides Service Providers with a unified framework to proactively configure good routes to optimize the routing objectives that matter to them, preparing Service Providers to provide the increasingly reliable, performant service that our applications need.

2 Background and Motivation

Setting: Service Providers offer their services from tens to hundreds of geo-distributed sites [68]. 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. Sites can forward requests to other sites via private WANs, but such forwarding is undesirable since it uses valuable private WAN capacity [47]. 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 [30, 16]. Resources can also fail completely due to, for example, physical failure and misconfiguration.

Upgrading capacity has fixed costs (*e.g.,* fiber, router backplane bandwidth, line card upgrades, CDN servers near peering routers) and variable costs that scale with demand (*e.g.,* power, transit), both of which Service Providers strive to keep low [78, 1, 16, 90]. We model deployment cost as correlated with the peering capacity over all links/sites, even though the actual relationship may be complex.

2.1 Variable, Evolving Goals

Evolving Internet use cases are pushing Service Providers to meet diverse requirements for their applications. For example, Service Providers increasingly host mission-critical services such as enterprise solutions [50], which require very high reliability and are predicted to be a $60B industry by 2027 [41]. Gaming is a similarly important industry generating more revenue than the music and movie industries combined [5], but instead requires latency within 50 ms [71]. Service Providers are also expanding the set of services they provide — for example, CDNs who traditionally hosted static content are pivoting to offering services like general compute [20, 28, 27].

As use cases evolve, Service Providers increasingly need to meet performance requirements for ingress traffic. As Service Providers move away from CDN business models, ingress traffic moves from simply TCP acknowledgements to critical, performance-sensitive traffic such as player movements in

real-time games, voice and video in enterprise conferences, and video/image uploads for AI processing in the cloud [18, 69].

Despite Service Providers' efforts to meet variable service requirements [66, 46, 35, 53, 70, 50, 60], meeting requirements is still challenging often due to unexpected changes *outside* Service Provider networks. For example, peering disputes can lead to congestion on interdomain links and inflated paths [30, 22, 25], and DDoS attacks still bring down sites/services [82, 73, 80, 51], despite the considerable effort in mitigating DDoS attack effects [74, 101]. 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 [58, 76, 16]. Operators regularly report these traffic surges on social media and blog posts [42, 55, 72, 64, 36, 65].

2.2 Routing Traffic to Service Providers

A way to protect against these changes is to route traffic over different interdomain paths thus avoiding overwhelmed sites/links and balancing load. Existing systems balance traffic over egress interdomain paths [86, 103, 37, 54], but it remains unclear how to configure and balance traffic across a good set of ingress interdomain paths.

Ingress Path Model: 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. Service Providers can then balance traffic across paths using traffic engineering (§3.2.1).

We model interdomain paths from users to the Service Provider as non overlapping, except at the peering link through which each path ingresses to the Service Providers' deployment. We assume the peering link is the bottleneck of each path, as the actual bottleneck is difficult to measure, and that users share the capacity of those peering links. (We leave relaxing these assumptions for future work — see the remark at the end of this subsection). This link is one of many at a particular site. Hence a user may have many paths to a site, each with different capacities and latencies. A path to a given prefix will be through one of the corresponding ingress links over which that prefix is advertised.

Current Ingress Routing Approaches: Today, most Service Providers either use `anycast` prefix advertisements to provide relatively low latency and high availability at the expense of some control [8, 49, 108, 107], or `unicast` prefix advertisements to direct users to specific sites [52, 86, 13].

`Anycast`, where Service Providers advertise a single prefix to all peers/providers at all sites, offers some natural availability following failures since BGP automatically reroutes most traffic to avoid the failure after tens of seconds [108]. Prior work shows that this availability comes with higher latency in some cases [59, 49]. `Unicast` can give clients lower latency than `anycast` by advertising a unique prefix at each site [13], but can suffer from reliability concerns potentially taking as much as an hour to shift traffic away from bad routes [50].

Remark on Our Path Model: Although we do not explicitly account for the possibility of bottlenecks in the public Internet, we implicitly account for it by presenting a framework that gives good routes even under 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 [30, 104, 50] and egress traffic engineering [103, 37, 86, 54] also does not explicitly address this problem.

2.3 Limitations of Current Approaches

Too Specific: Some recent work has noted that specific objectives may be better met by offering better interdomain routes, achieving better latency and/or reliability [30, 8, 104, 107, 50]. These solutions advertise different prefixes to subsets of all peers/providers to offer users more paths. However, these solutions optimize specific objectives, mostly steady-state latency, and in Section 3.4 we discuss why it is challenging to extend these approaches to consider other objectives.

Figure 1 shows how this specific focus on a single objective could lead to reliability issues, *e.g.,* during a deployment failure. In normal operation, user traffic is split evenly across two prefixes and achievesg low latency (Fig. 1a). However when site B fails, BGP chooses the route through Provider 1 for all traffic, causing link overutilization and subsequent poor performance for all users (Fig. 1b). In Section 5 we show that this overload happens in practice for systems that provide very low (steady-state) latency from users to Service Provider networks such as `AnyOpt` [104] and `PAINTER` [50], since those systems optimize for specific objectives.

Too Reactive: To enhance reliability, some Service Providers propose systems that react to changing conditions — for example, `FastRoute` and `TIPSY` drain traffic from overloaded links/sites. However, reactivity still requires time to change to a new state, leading to short-term service degradation and additional uncertainty at a time where unexpected things are already occurring. Reacting by modifying BGP announcements or DNS configurations as `TIPSY` and `FastRoute` do is also undesirable since these actions are known to lead to outages [6, 44, 10, 60].

Too Conservative: Another common approach is to overprovision resources to handle transient peak loads [94, 1, 60], but Figure 2 demonstrates that doing so can incur excessive costs. Figure 2 computes differences in peak utilization on links between different successive time periods, using longi-

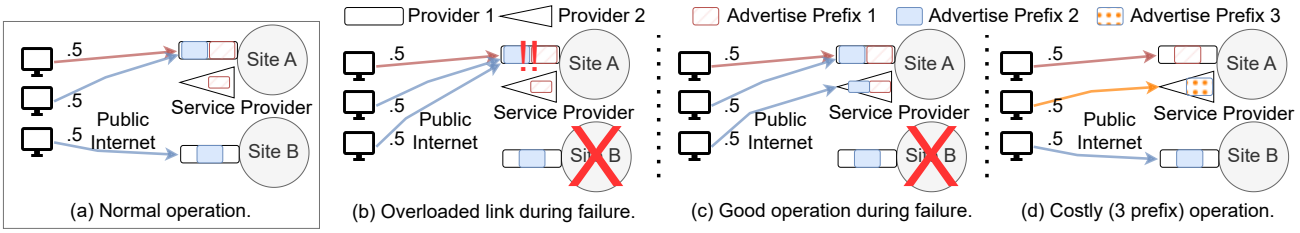


Figure 1: In normal operation traffic is split between two sites by directing half the traffic to each prefix (a). When site B fails, there is enough global capacity to serve all traffic (each link can handle 1 unit) but no way to split traffic across multiple providers given available paths leading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

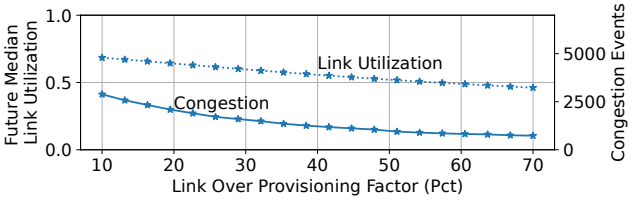


Figure 2: Planning for peak loads to avoid overloading requires inefficient overprovisioning.

tudinal link utilization data from OVH cloud [77].

To generate Figure 2 we first 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. The dataset reports link utilizations over approximately 5-minute intervals. 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 (varying the factor on the X axis). We then compute the average link utilizations in the next period and the total number of links on which we see overloading ($\geq$ 100% utilization in at least one time interval).

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. Microsoft and Meta also stated that solely installing extra capacity without shifting traffic is not always a feasible solution [30, 86].

Efficient, Proactive Planning is Possible: Despite these limitations, Figure 1c shows that by advertising prefix 2 to provider 2 at site A a priori, the Service Provider can split traffic between the two links during failure avoiding overloading without (a) reacting to the failure and (b) upgrading deployment capacity.

2.4 Key Challenges

Since a Service Provider has lots of global capacity, dynamically placing traffic on paths to optimize performance objectives subject to capacity constraints would therefore be simple if all the paths to the Service Provider were always available to all users as in Figure 1d. However, making paths available uses IPv4 prefixes, which are monetarily expensive and pollute BGP routing tables. The solution in Figure 1d uses 50% more prefixes than the solution in Figure 1c and, at $20k per prefix for 10,000 peerings would cost $200M [75]. IPv6 is not a good alternative as 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 [104, 50], and most Service Providers advertise fewer than 50 today according to RIPE RIS BGP data [92]. We also verified with engineers at six large service providers that this was a valid limitation.

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

Finding that right subset of paths to expose that satisfies performance objectives, however, is hard since there are exponentially many subsets to consider and each subset needs to be tested (*i.e.,* advertised via BGP) to see how it performs. The number of subsets is on the order of 2 to the number of peers/providers Service Providers have which, for many Service Providers, is $> 2^{1,0000}$. As measuring this many advertisements is intractable, we have to predict how different subsets of paths perform which is challenging since interdomain routing is difficult to model, 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 goal is to find an advertisement strategy that gives users good interdomain paths (relative to some objective function) and a traffic allocation to those paths. SCULPTOR should

also allocate good backup paths to handle non steady-state conditions. For objective functions we consider, such optimization problems are often large with more than 100M constraints and 2M decision variables (§3.2.2).

Minimizing an objective function requires evaluating it with several different inputs, but performing such measurements (*i.e.,* advertising prefixes) could take years at our problem size (§3.3.1), and so is not scalable. Instead, we estimate the objective by modeling paths probabilistically (§3.3.2), and update this model over time using a small number of measurements in the Internet (§3.3.3).

We then minimize the objective function using gradient descent which, at each iteration, requires solving millions of subproblems (§3.4). To avoid expending the resources to solve all these sub-problems, we also implement efficient heuristics that find good approximate solutions (appendix A.4), although these heuristics are specific to the objective function we use in our evaluations. These heuristics speed computation at the cost of some convergence (§5.4), but are not necessary.

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 enable SCULPTOR to compare millions of possible advertisement strategies at milliseconds per computation, but only conduct tens in the actual Internet.

3.2 Problem Setup and Definitions

3.2.1 Goal and Setting

We aim to advertise prefixes to connected networks to optimize an objective function subject to constraints such as link capacity. Adding capacity constraints on sites (*e.g.,* servers) in addition to links would be a straightforward extension.

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 to a prefix (and therefore peering link) is chosen via BGP. We fix the maximum number of allowable prefixes according to the Service Provider's budget, and is generally much less than the number of peerings.

We consider users at the granularity of user groups (UG*s*), which generally refer to user networks that route to the Service Provider similarly, but could mean different things (*e.g.,*/24 IPv4 prefixes, metros). UG*s* generate steady-state traffic volumes, $v_{\texttt{UG}}$, and the Service Provider provisions capacity at links/sites to accommodate this load. A system run by the Service Provider measures latency from UG*s* to Service Provider peering links l, which is a reasonable assumption [9, 13, 24, 34, 104, 50].

We assume that the Service Provider has some technology for directing traffic towards prefixes. Examples include DNS [13, 8, 52], multipath transport [81, 23], or control-points at/near user networks [84, 50]. DNS offers slow redirection due to caching [50] 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 [3] and Ubuntu 22 [31], and MPQUIC can be used in any application [106]. SCULPTOR assumes that a UG can be precisely directed to each prefix.

3.2.2 General Formulation

We cast the problem of finding advertisement configurations that admits good assignments of user traffic to paths as a multivariate optimization over both configurations and traffic assignments.

We represent an advertisement configuration A as a binary vector. Each entry indicates whether we advertise/do not advertise a particular prefix to a particular peer/provider (similar to prior work [104, 50]). Implementing this advertisement configuration (*i.e.,* advertising prefixes via BGP sessions) 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, whose entries specify traffic allocation for each user along each route.

Although we do not know exactly which routes will result from a given configuration, assume that this mapping is well defined. Let R be the mapping from advertisement configurations to functions that map $\langle$`prefix, UG`$\rangle$ pairs to links. 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 UG*s* 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 0 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, maximum latency, and their combinations (see appendix A.3 for a discussion of which metrics work better than others). The joint minimization over configurations and traffic assignments can then be expressed as the following.

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

The first constraint requires that traffic assignments be non-negative and the second requires that configurations are binary. The third constraint requires that all user traffic $v(\texttt{UG})$ is assigned, and none is assigned to nonexistent paths. 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 (or at least the ones we could think of, like site capacity) is straightforward.

3.2.3 Specific Objectives

In our evaluations we focus on two specific objectives: (1) minimizing latency and maximum link utilization in unseen scenarios, and (2) optimally routing different traffic classes. One could choose any objective that can be met in intra-domain settings. With our interdomain path model (§2.2), path latency for a UG to a prefix is uniquely determined by UG and the corresponding peering link over which the UG ingresses. Let the latency for a user UG via a link l be $\mathcal{L}(\texttt{UG},l)$. G is then given by the following.

$$\begin{aligned}
G(R(A),w) &= \sum_{p,\texttt{UG}} \mathcal{L}(\texttt{UG},R(A)(p,\texttt{UG}))w(p,\texttt{UG}) \; + \; \beta M \\
\text{s.t.} \quad & \frac{\sum_{R(A)(p,\texttt{UG})=l} w(p,\texttt{UG})}{c(l)} \leq M \quad \forall l
\end{aligned} \tag{2}$$

Constraining maximum link utilization, M, to be at least as much as the utilization of each link and then minimizing a sum including it forces M to be the maximum link utilization. The sum of average latency and maximum link utilization is weighted by a parameter, β, which represents a tradeoff between using uncongested links/sites and low propagation delay and is set by the Service Provider based on their goals. We first solve Equation (2) with $M = 1$ to see if we can allocate traffic to paths with zero overloading. However such overload-free solutions may not exist for all A.

Unseen Scenarios To encourage good solutions to Equation (1) not only in steady-state but also in unseen conditions (failures, shifting traffic distributions), we add a regularizer to G given by $\sum_l \alpha_l G(R(A * F_l),w)$. F_l is the binary vector where $F_{l,i} = 0$ if index i of A corresponds to advertising a prefix through link i and 1 otherwise. The optional weights α_i allow operators to specify which failure events may be more impactful or likely. In other words, multiplying by F_l simulates a failure, since it effectively withdraws all advertisements through link l, blackholing that traffic. This regularizer intuitively encourages advertisement solutions to provide UGs good primary and backup paths, thus preparing for dynamic conditions. In Appendix A.2 we discuss why this regularizer helps us find good global minima, despite Equation (1)'s non-convexity, by drawing analogies to the deep learning literature.

Different Traffic Classes We consider a separate problem where we try to route traffic with different performance requirements. One traffic class (latency-sensitive) should be routed with low latency, and the other (bulk) should not congest the latency-sensitive traffic. The objective function is a weighted combination of the average latency of latency-sensitive traffic and the amount of latency-sensitive traffic that is congested. We also constrain the maximum bulk over-provisioning on any link.

3.3 Predicting Interdomain Routes

Solving Equation (1) is challenging because we need to compute $G(R(A),w)$ but computing $R(A)$ requires advertising prefixes in the Internet which can only be done infrequently to avoid route-flap-dampening. In our optimization (detailed below) we compute $R(A)$ 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.

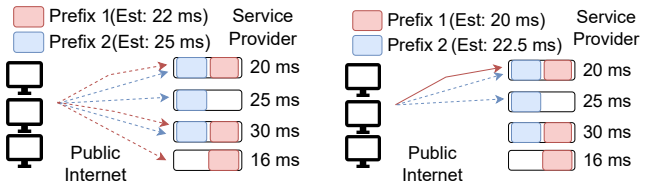
3.3.1 Initialization and Measuring $R(A)$

We initialize our strategy to be `anycast` on one prefix, and `unicast` on remaining prefixes (*i.e.,* one prefix per site). If we have more prefixes in our budget, we advertise those prefixes to randomly chosen peers/providers.

During optimization we periodically measure $R(A)$ by advertising the corresponding prefixes via peerings as specified by A and measuring routes taken by UGs to each prefix. The time between successive advertisements is between 20 minutes and 2 hours to avoid route flap dampening.

3.3.2 Probabilistic UG Paths

We model the routing function, and therefore our objective functions, probabilistically and update our probabilistic model over time as we measure how UGs route to the deployment, adapting methods from prior work [104, 88, 50]. When, during optimization, we require a value for $G(R(A),w)$, we compute its expected value given our current probabilistic model. We compute this expected value either approximately via Monte-Carlo methods, or exactly if G's structure admits efficient computation (appendix A.4).



(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 (3b). We use this information to refine our latency estimate towards the blue prefix (since the third ingress is no longer a possible option) without advertising the blue prefix, saving time.

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.

A useful property of many choices of objective function G is that we do not need to approximate the routing function perfectly to solve Equation (1). For example, most UG*s* have similar latency via their many paths to a single site, so when optimizing latency as in Equation (2) we effectively only need to predict the site the user ingresses at. As we exclude more options of where UG*s* could possibly ingress for each prefix, our objective function's distribution on unmeasured scenarios converges to the true value.

An example of this process is shown in Figure 3, where we refine a latency estimate towards an unmeasured prefix (blue) using measurements towards other prefixes (red). Even though we only have 50% confidence in exactly which path the user takes (*i.e.,* in $R(A)$), we have high confidence that the user's latency towards the blue prefix will be about 22.5 ms.

3.3.3 Exploration to Refine $R(A)$

As, a priori, we do not know UG preferences, estimates of G could be very noisy until we learn preferences. To bootstrap our model, we periodically measure $R(A)$ on "adjacent" advertisement configurations to our current scheme and look for strategies for which we are very uncertain about whether it will be better or worse. By adjacent schemes, we mean configurations that differ from A by one entry, or configurations representing failures ($A \times F_l$). One could use various measures of uncertainty — we choose entropy. Measuring $R(A)$ in the Internet takes time (§3.3.1), but we observed that it improves convergence.

3.4 A Two Pronged Approach

Even with our probabilistic model, we cannot solve Equation (1) directly/exhaustively since it is a mixed-integer program with millions of constraints which is too large for off-the-shelf solvers. Greedy [50] and random [104] solvers find good solutions for simple objectives such as steady-state latency, but do not generalize to other objectives with constraints (§5).

Instead, we split Equation (1) into an outer and inner optimization, solving for configurations, A, on the outer loop using gradient descent and traffic assignments, w, on the inner loop using a general-purpose solver. Intuitively, this approach works because it breaks up a challenging optimization into many sub-problems that can be solved in parallel, and is guaranteed to push us towards *a* (perhaps locally optimal) solution.

3.4.1 Outer Loop: Gradient Descent

To apply gradient descent to our integer configuration variable, we extend A to be 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 approximate gradients between adjacent advertisements A_i and A_j by computing the expected value of $G(R(A), w)$ at each advertisement and continuously interpolate G at intermediate configurations 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 continuous extension and dimension reduction methods are also used in other settings [89, 56].

Scaling: Minimization in the outer loop 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). Gradient entry computations are parallelizable so this loop scales horizontally.

Convergence: Gradient descent converges to a local minima for bounded objectives [57]. Our evaluations show that SCULPTOR finds good solutions over a wide range of simulated topologies, and converges quickly with thousands of UG*s* and ⟨`peering, prefix`⟩ pairs (§5). We comment on which problem properties will likely lead to better convergence in Appendix A.2 by drawing analogies to the deep learning literature, but find that allocating higher prefix budgets and adding richer advertisement capabilities (*e.g.,* BGP community tagging) can improve convergence, the latter of which is an area for future work.

3.4.2 Inner Loop: General Purpose

Each iteration of the outer loop requires solving for traffic allocations on advertisements corresponding to the gradient

entries that we wish to compute. We refer to the process of solving for these traffic allocations as an inner loop.

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

Convergence depends on the objective, G, and the solver. Many useful objectives are expressed as linear programs which are convex (*e.g.,* our choices of G), so solvers can find globally optimal solutions. Useful, non-convex objectives such as `Cascara`'s sometimes have approximately optimal solutions [90].

4 Implementation

We prototype `SCULPTOR` on the PEERING testbed [85], 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 [50], 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 [12]). Vultr likely peers with networks with which it exchanges a significant amount of traffic [86], 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 $\alpha_I = 4.0$ and set the learning rate to 0.01 with decay over iterations. We solve Equation (2) with Gurobi.

Experiments in the Internet We assess how `SCULPTOR` performs on the Internet using RIPE Atlas probes [91], 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).

Simulations We also evaluate `SCULPTOR` by simulating user paths which allows us to conduct more extensive evaluations, as experiments take less time and use clients in more networks. To limit computational complexity, we select 100 clients with a route through each peer. 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, 62, 50, 76]. (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 Vultr links and cannot even determine which peering link that traffic to one of our prefixes arrives on, as Vultr does not give us this information. (This limitation only exists since we are not a Service Provider, as a Service Provider could measure this using `IPFIX`, for example.) 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 [85], and measure actual ingress links to those prefixes using traceroutes from RIPE Atlas probes [91]. 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 infer ingress links is simulating user paths by assuming we know all user routing preference models (§3.2). We use a preference model where clients prefer peers over providers, and clients have a preferred provider. When choosing among multiple ingresses for the same peer/provider, clients prefer the lowest-latency option. We also add in random violations of the model. This second approach allows us to evaluate our model on all client networks but may not represent actual routing conditions. 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 [104].

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

5 Evaluation

We evaluate `SCULPTOR` on two objective functions (§3.2.3). In optimizing the first objective (§5.2), `SCULPTOR` yields advertisement strategies that give clients lower latency paths and fewer congested links than other solutions during steady-state (§5.2.1), failure (§5.2.2), and conditions (§5.2.3). In optimizing the second objective, `SCULPTOR` successfully routes latency-sensitive traffic and excessive amounts of lower priority bulk traffic.

5.1 General Evaluation Setting

We compare `SCULPTOR` to other solutions.

`anycast`: A single prefix announcement to all peers/providers at all sites, which is a common strategy used by Service Providers today [33, 17, 8, 97, 79, 102, 107].

`unicast`: A single prefix announcement to all peers/providers at each site (one per site). Another common strategy used in today's deployments [52, 13]. `Unicast` also provides an upper bound on FastRoute's performance.

`AnyOpt/PAINTER`: Two proposed strategies for reducing steady-state latency compared to `anycast` [104, 50].

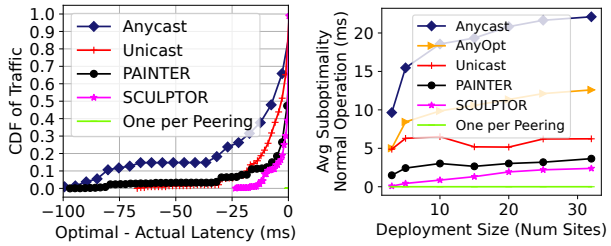
`One-per-Peering`: A unique prefix advertisement to each peer/provider, so many possible paths are always available from users. This solution serves as our performance upper-bound, even though it is prohibitively expensive. (We do not know an optimal solution with fewer prefixes.)

For the public Internet implementation (§4.2), 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 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), 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 **`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 [104, 50]. Over all studied deployments, we consider paths from clients in 42k /24s to 868 peers/providers.

In solving for traffic allocations during failure scenarios, links may be overloaded. We say all traffic arriving on a congested link is congested and do not include this traffic in latency comparisons (congested traffic latency would be a complicated function of congestion control protocols and queueing behavior). We comment separately on how much traffic is congested.

We compute both average overall latency and the fraction of traffic within 10 ms (very little routing inefficiency), 50 ms (some routing inefficiency), and 100 ms (lots of routing inefficiency) of the `One-per-Peering` solution for each advertisement strategy.



(a) (b)

Figure 4: SCULPTOR provides the lowest steady-state latency both on the Internet (4a) and in simulation (4b).

5.2 Handling Unseen Conditions

5.2.1 Lower Latency in Steady-State

Internet Deployment We compute the latency that each advertisement solution achieves compared to the One-per-Peering solution for each UG. Figure 4a 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 [50].

Simulations Figure 4b 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.

5.2.2 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 5 demonstrates that SCULPTOR shifts traffic without overloading alternate links/sites more effectively than any other solution *without BGP announcements/withdrawals* which could cause further failure (§2.3).

In these evaluations, we fail each ingress/site once and compute traffic allocations. 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.

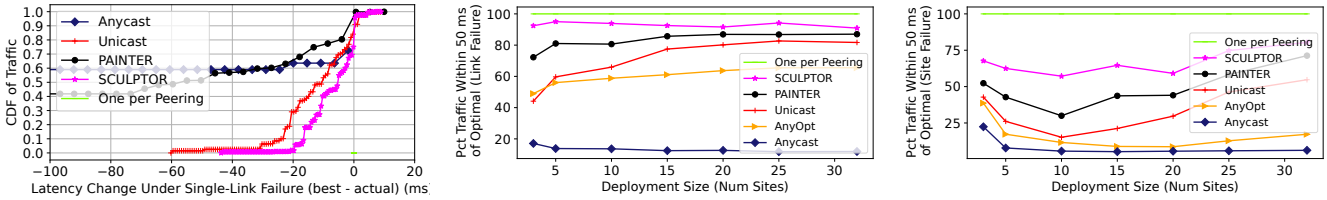
We separately note the fraction of traffic that lands on congested links for each solution. We do not include congested traffic in average latency computations but say such traffic does *not* satisfy 10 ms, 50 ms, or 100 ms objectives.

Internet Deployment Figure 5a 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.

Simulations We show the fraction of traffic within 50 ms of the One-per-Peering solution for link and site failures in Figure 5b and Figure 5c (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



(a) (b) (c)

Figure 5: SCULPTOR improves resilience to failure both on the Internet (5a) and in simulation (5b, 5c).

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).

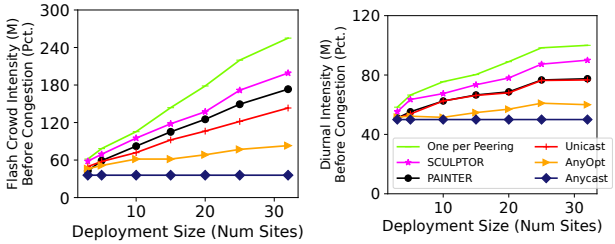
Differences quoted above may seem small (for example, 11.3% in the amount of traffic within 10 ms of One-per-Peering between SCULPTOR and PAINTER), but represent potential fractions of *trillions* of requests per day made across the whole world [86]. Large Service Providers recently emphasized that these seemingly small gains make large differences [29, 100].

5.2.3 Efficient Infrastructure Utilization

Figure 6 shows that installing more capacity to handle peak loads 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 under two realistic traffic patterns: flash crowds and diurnal effects.

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 Figure 6, we identify each client with a single "region" corresponding to the site at which they have the lowest possible latency ingress link. For each region individually, we then scale each client's traffic in that region by M% and compute traffic to path allocations. If there are S sites in the deployment, we thus compute S separate allocations per M value where each allocation assumes inflated traffic in exactly one region. 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,



(a) (b)

Figure 6: SCULPTOR improves infrastructure utilization under flash crowds and diurnal traffic patterns so that Service Providers can underprovision compared to peak loads.

we call $M = 60$% the critical value of M.

Our diurnal analysis in Figure 6b 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 [16]. 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.

Results Figure 6 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.

Figure 6a 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 6b 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 [43, 38, 37].

Prior work noted similar benefits by satisfying user requests in distant sites [16], 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 [86] and Fastroute [30]) do not always shift traffic on their backbones since backbone capacity is precious.

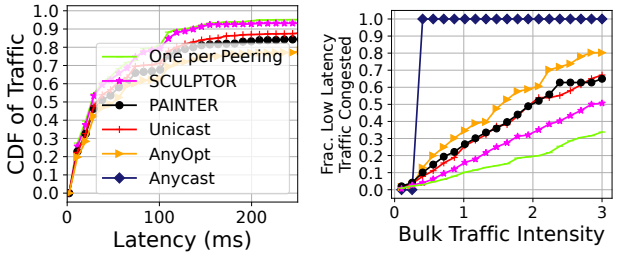
It is interesting that PAINTER provides similar benefits to unicast— this result is likely since PAINTER only aims to reveal low latency primary paths for users, whereas unicast guarantees reachability to all sites guaranteeing users a minimum number of paths (one of which may have capacity).

5.3 Handling Multiple Traffic Classes

We also evaluate SCULPTOR's ability to satisfy multiple traffic classes. In addition to the traffic considered in Section 5.2 ("latency-sensitive"), we add lower priority traffic ("bulk"). The objective is to route latency-sensitive traffic to low-latency routes without congestion from bulk traffic. We do not penalize cases where bulk traffic is congested (since it is lower priority), but do limit the maximum amount of any traffic on a link to 10× the capacity of the link to avoid trivial solutions where all bulk traffic is placed on one link. We solve SCULPTOR on a single **TBD: 32 PoP** deployment (We could not use our speedup heuristics (appendix A.4) for this objective function and so did not expend compute to simulate many deployments).

In Figure 7b we vary the amount of bulk traffic as a multiplier over the amount of high priority traffic. Links are still overprovisioned to 30% higher than latency-sensitive traffic volume, so there is insufficient global capacity to route all traffic for all "Bulk Traffic Intensities" over 0.3. SCULPTOR finds strategies that allow us to route more bulk traffic with less congestion than all other approaches, and stay close to the One-per-Peering solution. For example, when routing 2× the amount of bulk traffic as latency-sensitive traffic, SCULPTOR achieves **TBD: 20**% less congestion.

SCULPTOR also routes latency-sensitive traffic with lower latency — Figure 7a shows that SCULPTOR routes traffic with nearly as low latency and as little congestion as the One-per-Peering solution (y-intercepts show fractions of latency-sensitive traffic congested). Figure 7a shows latency and congestion for a bulk traffic intensity of 0.5.



(a) (b)

Figure 7: SCULPTOR routes high priority traffic with low latency (7a) and minimal congestion from lower priority traffic (7b).

5.4 Why SCULPTOR Works

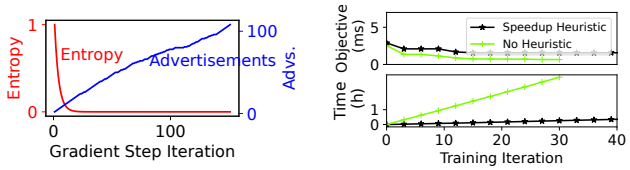
SCULPTOR solves a challenging integer optimization problem in a way that yields solutions better than other approaches. SCULPTOR quickly predicts how other advertisement strategies perform so that it can pick the best ones. This prediction is an approximation of true performance (§3.1) and the question we aim to answer 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 (§3.3.3) or for strategies that SCULPTOR thinks will yield good performance (§3.4). Figure 8a 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 (§3.3.3).) 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. **TBD: add something about convergence sacrifices**

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 [103, 86, 54]. SCULPTOR works in tandem with



(a) (b)

Figure 8: SCULPTOR's model entropy decreases exponentially with few path measurements (8a). Heuristic speedups can also speed computation, but may sacrifice convergence (**??**.

these systems, similarly working with BGP, but to optimize ingress traffic. Other work also shifts traffic to other links/sites to lower cost [105, 90, 16]. 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, 62, 30, 104, 108, 107, 50], 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 [30]. Our solution is more complex, but more effective as shown by our comparisons to `unicast` (§5.2.2). Pecan [96] 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 [95, 40, 2, 19, 58]. 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 [83], 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 [66]. However, this approach is reactive and so will lead to some downtime. 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 [106]. DChannel uses both of 5G's high bandwidth and low latency channels to optimize web performance [87]. TGaming [14] 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 [26, 35, 53, 60]. 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 [99, 61, 11, 45, 16]. SCULPTOR solves different challenges that arise due to the lack of visibility/control into potential paths and their capacities in the interdomain setting.

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

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

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

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

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

[6] Ann Bednarz. 2023. Global Microsoft cloud-service outage traced to rapid BGP router updates. https://networkworld.com/article/971873/global-microsoft-cloud-service-outage-traced-to-rapid-bgp-router-updates.html

[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] Ben Cartwright-Cox. 2023. Grave flaws in BGP Error handling. https://edgecast.medium.com/the-cdn-edge-brings-compute-closer-to-where-it-is-needed-most-d4a3f4107b11

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

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

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

[14] 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).

[15] Ping-yeh Chiang, Renkun Ni, David Yu Miller, Arpit Bansal, Jonas Geiping, Micah Goldblum, and Tom Goldstein. 2022. Loss landscapes are all you need: Neural network generalization can be explained without the implicit bias of gradient descent. In *The Eleventh International Conference on Learning Representations*.

[16] 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*.

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

[18] Google Cloud. 2024. Cloud Video Intelligence API. https://cloud.google.com/video-intelligence

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

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

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

[22] 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

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

[24] 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*.

[25] 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.

[26] 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*.

[27] 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

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

[29] 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*.

[30] 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*.

[31] 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

[32] 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).

[33] 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*.

[34] Palak Goenka, Kyriakos Zarifis, Arpit Gupta, and Matt Calder. 2022. Towards client-side active measurements without application control. *SIGCOMM CCR 2022* (2022).

[35] 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*.

[36] 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

[37] 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*.

[38] 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*.

[39] W Ronny Huang, Zeyad Emam, Micah Goldblum, Liam Fowl, Justin K Terry, Furong Huang, and Tom Goldstein. 2020. Understanding generalization through visualizations. (2020).

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

[41] 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

[42] 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

[43] 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*.

[44] Santosh Janardhan. 2021. More details about the October 4 outage. https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/

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

[46] 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*.

[47] Bhaskar Kataria, Palak LNU, Rahul Bothra, Rohan Gandhi, Debopam Bhattacherjee, Venkata N Padmanabhan, Irena Atov, Sriraam Ramakrishnan, Somesh Chaturmohta, Chakri Kotipalli, et al. 2024. Saving Private WAN: Using Internet Paths to Offload WAN Traffic in Conferencing Services. *arXiv preprint arXiv:2407.02037* (2024).

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

[49] 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*.

[50] 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*.

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

[52] 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*.

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

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

[55] 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

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

[57] 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).

[58] 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*.

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

[60] 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*.

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

[62] 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*.

[63] Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew G Wilson. 2022. PAC-Bayes compression bounds so tight that they can explain generalization. *Advances in Neural Information Processing Systems* 35 (2022), 31459–31473.

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

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

[66] 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*.

[67] Congcong Miao, Zhizhen Zhong, Yunming Xiao, Feng Yang, Senkuo Zhang, Yinan Jiang, Zizhuo Bai, Chaodong Lu, Jingyi Geng, Zekun He, et al. 2024. MegaTE: Extending WAN Traffic Engineering to Millions of Endpoints in Virtualized Cloud. (2024).

[68] Microsoft. 2023. Microsoft Datacenters. https://datacenters.microsoft.com/

[69] Microsoft. 2024. Emirates Global Aluminiium cuts cost of manufacturing AI by 86 percent with the introduction of Azure Stack HCL. https://customers.microsoft.com/en-us/story/1777264680029793974-ega-azure-arc-discrete-manufacturing-en-united-arab-emirates

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

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

[72] 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

[73] 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*.

[74] 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*.

[75] NANOG. 2022. Panel: Buying and Selling IPv4 Addresses. https://youtube.com/watch?v=8FlTJEct9_s

[76] 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*.

[77] 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*.

[78] Leon Poutievski, Omid Mashayekhi, Joon Ong, Arjun Singh, Mukarram Tariq, Rui Wang, Jianan Zhang, Virginia Beauregard, Patrick Conner, Steve Gribble, et al. [n.d.]. Jupiter evolving: transforming google's datacenter network via optical circuit switches and software-defined networking. In *SIGCOMM 2022*.

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

[80] 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

[81] 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*.

[82] 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

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

[84] 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*.

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

[86] 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*.

[87] 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*.

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

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

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

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

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

[93] 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)*.

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

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

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

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

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

[99] 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*.

[100] 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*.

[101] 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.

[102] 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*.

[103] 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*.

[104] 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*.

[105] 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*.

[106] 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*.

[107] 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*.

[108] 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. (2)), we try to minimize average latency. It could be, however, that Service Providers wish to optimize other/additional metrics. Here we try to determine which metrics our framework will more likely work for and, in doing so, identify what about SCULPTOR gives good performance. We do not provide any theoretical guarantees but give valuable intuition backed by experimental demonstrations.

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

A.1 Gradient Descent Challenges

We propose using gradient descent to solve Equation (1) primarily because of its parallelizability which is important given the large problem sizes we consider and the complexity of predicting routes. Recent work suggests that gradient descent may not be important for finding good minimizers [15], so other minimizers may also work well for SCULPTOR.

To use gradient descent to solve Equation (1), we require that our objective function is a differentiable function of real variables. Ours, as we have presented it, does not satisfy that condition. Ignoring possible problems with the objective function G, there are two problems: first, our configuration A is a binary variable, and, second, the function R is not well-behaved.

To see what we mean by this second point, consider computing a gradient near a configuration that only advertises one prefix to one provider. A "small" change in the configuration (withdrawing that prefix) would lead to zero reachability and thus no solution, which suggests that R is not "differentiable" in a fundamental way, or at least not differentiable everywhere. Gradient descent converges to minima for functions whose second derivatives are locally bounded [57]. Intuitively, this condition means that a gradient should not push the descent in significantly different directions in small regions since descent could, for example, oscillate and never decrease the loss.

To solve the first problem, we apply gradient descent to continuous extensions of configurations and the routing function. There are many ways of performing this extension. One way of continuously extending these ideas that we found works well is to treat configuration components as real-valued, rather than binary, variables between 0 and 1 and interpolate objective function values between "adjacent" components of configurations using sigmoids (see Section 3 for details). Sigmoids likely work well since their configurable parameter controls steepness, which allows us to control the magnitude of gradients (*i.e.,* not making them too large which could harm convergence).

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

This solution is reasonable as Service Providers often connect to several transit providers at each site to provide global reachability to all users even under partial congestion or failure [4].

A.2 The Key Challenge: Convergence

With bounded second derivatives of the objective function gradient descent converges to a minima [57], and can converge to that minima quickly with good initializations and choices of the learning rate. However, unless the objective function is convex (ours is not because the routing function is not) then gradient descent may converge to local minima. We find that SCULPTOR likely converges to good minima because of our choice of loss function and that this convergence could improve with more prefixes and richer BGP configuration options.

A.2.1 Nonconvexity in Deep Learning

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

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

In practical settings we only have a corpus of data, rather than the distribution, which serve as samples of that distribution. During gradient descent, deep learning algorithms randomly sample from this corpus and move the parameter gradient roughly along the average of the gradients evaluated

at each corpus point [48]. 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 [32] (or gradient descent at all [15]).

Functions f_α contain potentially billions of parameters and are generally nonconvex. The functions can be compositions of many (tens to hundreds) of functions and so may be "deep". Since these functions contain so many parameters, they can both theoretically and experimentally find a global minimizer of the loss [21], despite nonconvexity. One risk of using too many parameters, however, is "overfitting" — even though a learned function f_α minimizes loss on the corpus of data, it may have 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. While it is generally believed that using too many parameters leads to overfitting **TBD: cite**, recent work suggests that it may relate closer to the geometry of the loss function [15].

Given a task to complete, training a deep learning model to accomplish this task consists of choosing a model, loss function, and optimization method (*e.g.,* gradient descent). To test the efficacy of a training methodology, practitioners mimic applying their learned model to a new setting by splitting their corpus into a *train* and *test* set. Practitioners then optimize using the training set, and evaluate the performance of their learned model on the test set. If practitioners observe good test performance, they can be more confident that in other settings their algorithms did not overfit and hence "generalize".

A.2.2 Nonconvexity in SCULPTOR

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

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

Although the question of why neural networks offer generalization is open, one compelling theory relates to the geometry of the loss landscape. Prior work observes that "wide" loss regions generalize much better than thin/sharp loss regions [39]. One way of interpreting wideness is that perturbations in the parameters, α, do not significantly increase the value of the loss.

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

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

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

Underfitting Another 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 at the global minimum. For example, when minimizing average latency it could be that a solution to achieve 35 ms average latency exists, but gradient descent arrives at a local minima giving an average latency of, for example, 50 ms. In Deep Learning, this problem is called underfitting and can happen when the chosen function does not have sufficient representational complexity [63].

SCULPTOR underfits the training data as shown by the suboptimality compared to the One-per-Peering solution (§5.2.2).

One way of fitting the data better is to consider richer functions with more parameters. In our setting, this option means increasing the number of "parameters" in both advertisement configurations and path options. In other words, this option means increasing the number of prefixes, or types of configurations to offer UG*s* more paths. By this last point we mean, for example, using AS path prepending to potentially offer richer path options. Our investigation in **TBD: fig** confirms this intuition, as using more prefixes allowed us to converge to better solutions whereas using too few prefixes led to similar/worse performing configurations than PAINTER. Other work has used richer configurations to solve problems [96, 7], but adopting such methods to SCULPTOR would require solving scalability challenges.

We Do (Should) Not Learn Routing with GD Another type of "learning" we discuss in Section 3 is refining our estimate of the routing function R. However, this is not descent based learning and is independent of this discussion. In Section 3 we model the function R as a random function, but one that follows a specific structure (preference routing

model), and opportunistically removes randomness using Internet measurements. As we demonstrate in Figure 8a, the randomness is essentially zero for most gradient descent iterations.

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

A.3 So, Which Metrics Work?

The above challenges, solutions, and intuitions are valid independent of the specific metric we care to optimize. Certain metrics could introduce either convergence or computational challenges. Before discussing those challenges, we first present choices of G other than 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 $G(R(A), w) = \sum_{p,\texttt{UG}} L(R(A)(p,\texttt{UG}))w(p,\texttt{UG})$ where the function L is some arbitrary mapping on latency. The resulting problem is a linear program in w and hence convex in w. A simple variation is maximizing the amount of traffic below a latency threshold, which may be useful for enabling specific applications [71].

Traffic Priority: It could be that some traffic is a higher priority and so would benefit more from low latency, high reliability routing. Optimization over traffic allocations, w, would then be an iterated process — first allocating high priority traffic, then lower priority traffic as in prior work [103, 90, 67].

Transit Cost: Configuring routes that reduce expected transit cost would roughly amount to minimizing the *maximum* traffic allocation on any provider (possibly weighted by that provider's rate, possibly only considering certain traffic). Hence the objective function would take the form $G(R(A), w) = \max_l \sum_{p,\texttt{UG}:G(R(A)(\texttt{UG},p)=l)} w(p,\texttt{UG})$ 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 which may not be convex in w but can still often be solved with descent-based methods (see the following subsection).

End-to-End Considerations: It could be that, for example, a particular site has low ingress latency for a UG but large egress latency and so may not be a good choice for that UG. As Service Providers have knowledge of these limitations (for example, they can measure egress link utilization or egress latency [86]), they can serve these variables as inputs into SCULPTOR and solve for path allocations, w subject to those variales.

A.3.2 Convergence Concerns

We expect functions G whose gradients do not explode in the region of interest to converge well. For configurations, A, we encourage gradients to be well-behaved with a good initialization (appendix A.1). For traffic assignments w, it depends on how the metric, G, incorporates w.

Examples of metrics G of w that offer good convergence include low-order polynomials and exponents (within a reasonable range such that gradients are not too large) whereas bad choices are ones with ill-behaved gradients. As an example, 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

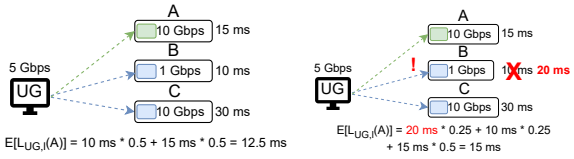
A key focus in Section 3 was finding *scalable* computational approaches to optimization, rather than worrying about their convergence. For example, instead of solving **??** during gradient descent, we approximate its expectation using efficient heuristics that take advantage of the nice structure of 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 as academic researchers at the global problem sizes we consider, Meta recently showed that they could leverage their infrastructure to solve millions of linear programs in minutes [26]. 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.

Hence SCULPTOR provides valuable, quantifiable latency and cost benefits for Service Providers (§5) and, given more computational resources, one could demonstrate that it may be even more valuable with respect to other metrics.

A.4 Heuristics for Fast Computation

With sufficient resources, solving Equation (1) using the two pronged approach outlined in Section 3.4 is feasible. Given



(a) (b)

Figure 9: 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 (9a). 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 (9b).

our advertisement rate limitations (§3.3.1) we wish to compute gradients in tens of minutes. With 100 Monte Carlo simulations per gradient and tens of thousands of gradients to compute, we thus need to solve millions of linear programs at each gradient step. Meta recently demonstrated that they could solve millions of linear programs in minutes using their infrastructure [26].

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

A.4.1 Capacity-Free UG Latency Calculation

Minimizers of Equation (2) balance traffic across links to satisfy capacity constraints. We instead temporarily ignore capacity constraints and assign each UG to their lowest-latency path. This approximation solves two problems at once. First, we do not need to solve a linear program to compute traffic assignments. Second, we can now analytically compute the distribution of our objective function, G, over all possible realizations of $R(A)$. Analytically computing this distribution is useful, since it lets us compute entropy (§3.3.3) and expected value (§3.3.2) without Monte Carlo methods.

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

A.4.2 Imposing Capacity Violation Penalties

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

That is, for each possible outcome of UG assignments to links, we compute link utilizations and note the probability those UGs 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 UGs from choosing paths that are likely congested using a heuristic — we emulate the impact of overloading by inflating latency in this calculation for UGs on those paths proportional to the overutilization factor. After emulating the effect of this overloading, we recompute the expected average UG latency *without changing UG decisions*. We do not change UG decisions as doing so could induce an infinite calculation loop if new decisions also lead to overloading. This heuristic penalizes advertisement strategies that would lead to many overutilized links if every user were to choose their lowest latency option. We show an example in Figure 9.

B Further Results

B.1 Steady-State Latency

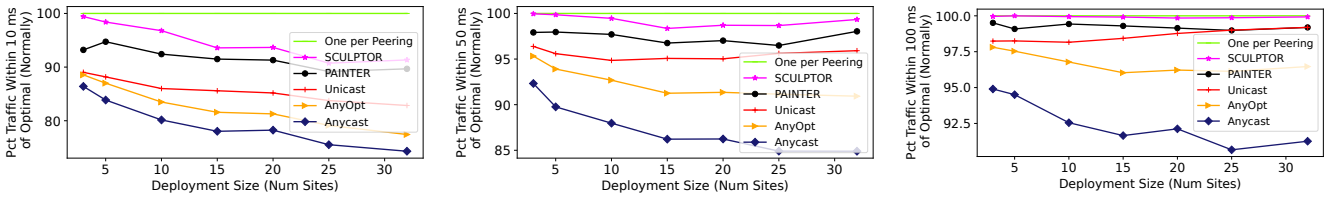
In addition to quantifying average latency during steady-state, we also compute the fraction of traffic within 10 ms, 50 ms, and 100 ms of the One-per-Peering solution during steady-state. Figure 10 shows how SCULPTOR compares more favorably to the One-per-Peering solution than other advertisement strategies.

B.2 Latency During Link/Site Failure

We show additional evaluations of SCULPTOR during link and site failures. Figure 11 shows that SCULPTOR provides good resilience to site failures on the Internet, and Figure 12 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.



(a) (b) (c)

Figure 10: SCULPTOR lowers average latency and increases the fraction of traffic within critical thresholds of their optimal latency during normal operation.

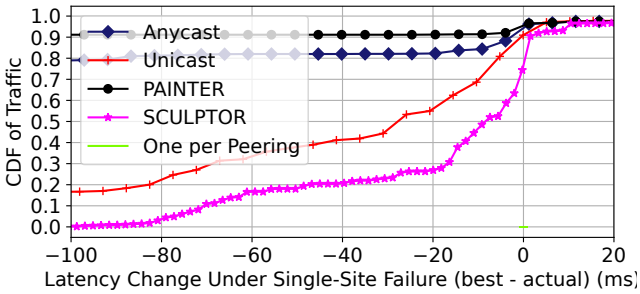


Figure 11: Our prototype on the public Internet shows that SCULPTOR offers low latency backup paths to users during site failures.

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 [16]. 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.

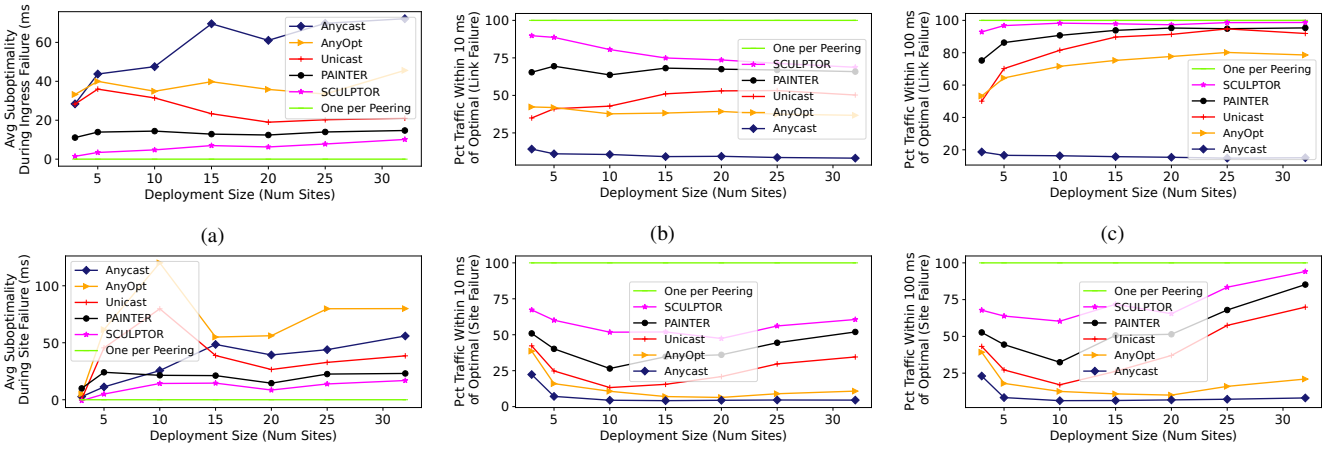


Figure 12: `SCULPTOR` lowers average latency and increases the fraction of traffic within critical thresholds of `One-per-Peering` latency during both link and site failures over many simulated deployments.