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

Paper #156, 13 pages body, 23 pages total

Abstract

Large cloud/content (service) providers 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, Service Providers overprovision to accommodate peak loads and activate emergency systems for shifting excess traffic. We take a different approach with SCULPTOR, which proactively advertises many prefixes for users so that Service Providers can use their global resources to meet generic objectives. SCULPTOR models Internet routing to approximately 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 route high-priority traffic with 2× less overloading.

1 Introduction

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

Increasingly, however, these services have diverse requirements that can be challenging to meet with a single path that Service Providers have little control over. For example, enterprise services have tight reliability requirements [55, 52], and new applications such as virtual reality require ≤ 10 ms round trip latency [76] and ≤ 3 ms jitter [102]. Complicating matters, Service Providers must meet these requirements subject to changing conditions such as peering link/site failures [30, 71], DDoS attacks [79, 111, 90], flash crowds [47, 60, 63], new applications/business priorities (LLMs), and route changes [77, 69, 37, 70, 84, 71]. Critically, such changes can cause *overload* if the Service Provider cannot handle new traffic volumes induced by the change, and DNS/BGP are slow mechanisms with which to change how traffic flows over paths. This overload can lead to degraded service for users, hurting reliability [78, 88, 56, 52], and may require manual intervention.

To try to meet different goals and respond to changing conditions, Service Providers adopt solutions that are either (a) too costly, (b) too reactive, or (c) too specific. For example, Service Providers proactively overprovision resources [103, 1, 65], but we demonstrate using traces that bursty traffic patterns can require costly overprovisioning rates as high as 70% despite sufficient global capacity (§2.3). TIPSY responds to changing loads to retain reliability [71], but only does so reactively, which could lead to short-term degraded performance. AnyOpt and PAINTER proactively set up routes to minimize specific objectives such as steady-state latency [115, 55], but it is challenging to extend those approaches to other objectives (§5.4). It is also unclear both how to combine these approaches (*e.g.,* retaining low latency under changing traffic loads) and how to apply approaches from one domain to another (*e.g.,* applying egress traffic cost reduction systems [99] to ingress traffic).

We present Service Providers with a flexible framework, SCULPTOR (**S**couring **C**onfigurations for **U**tilization-**L**oss-and **P**erformance-aware **T**raffic **O**ptimization & **R**outing), which accepts as input cost, performance, and reliability objectives and outputs BGP advertisements and traffic allocations that help achieve those desired objectives. SCULPTOR is the first system that finds good solutions for ingress interdomain routing objectives such as maximum link utilization, transit cost, and latency for *interdomain* traffic (existing systems work for intradomain traffic, *e.g.,* [48, 38, 72, 15]). SCULPTOR computes BGP advertisements proactively, and controls which subset of addresses handle active traffic. SCULPTOR only places live traffic on routes after convergence, and hence does not run the risk of outages.

To solve each traffic engineering problem, SCULPTOR efficiently searches over the large BGP advertisement search

space ($> 2^{10,000}$ possibilities, *i.e.,* from > 10,000 peerings) 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 finds configurations that 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 > 20M configurations (10,000× more than other solutions [115, 55]) while only measuring tens of configurations in the Internet (§5.4). Like other work that uses gradient descent with success (*e.g.,* deep learning), we sacrifice the ability to provide a formal characterization of which objective functions are possible for an approach that lets us approximately optimize multiple criteria (§5)

We prototype and evaluate our framework at Internet scale using the PEERING testbed [93] (§4), which is now deployed at 32 Vultr cloud locations [108]. Vultr is a global public cloud that allows our prototype to issue BGP advertisements via more than 10,000 peerings. We evaluate our framework on two specific objectives (computed separately): (a) minimizing latency under unseen traffic conditions, and (b) routing latency-sensitive and bulk traffic classes. We compare SCULPTOR's performance on both problems to that of an unreasonably expensive "optimal" solution (computing the actual optimal is infeasible).

For the first objective, we found that, compared to other approaches, SCULPTOR increases the amount of traffic within 10 ms of the optimal by 19.3% in steady-state (meeting that target for 95% of traffic) (§5.2.1), by 11% during link failure, and by 17% during site failure. SCULPTOR also reduces overloading on links during site failures by up to 41% over PAINTER, giving Service Providers more confidence that services will still be available during partial failure (§5.2.2). We also find that, by load balancing traffic on backup paths during peak times, we can satisfy high peak demands with the same infrastructure. SCULPTOR can handle flash crowds (*e.g.,* DDoS attacks) at more than 3× expected traffic volume, reducing the amount of overprovisioning that Service Providers need, thus reducing costs.

For the second objective, SCULPTOR routes bulk low-priority traffic in ways that avoid congesting high-priority traffic, achieving near-optimal latency and reducing congestion by up to 2× (§5.3).

After decades of intradomain traffic engineering using programmable networking primitives such as virtual output queueing for differentiated service [82] (and others, §7), SCULPTOR brings such control closer to realization in the interdomain setting with a unilaterally deployable framework. Service Providers can use flexible frameworks such as SCULPTOR to help bring us the resilient, performant service that our diverse applications increasingly need.

2 Motivation

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 [55], which require high reliability and are predicted to be a $60B industry by 2027 [46]. Gaming is another important industry generating more revenue than the music and movie industries combined [5], but instead requires latency within 50 ms [76]. Service Providers are also expanding the set of services they provide—for example, CDNs that traditionally hosted static content are pivoting to offering services like compute [19, 28, 27].

Moreover, as use cases evolve, Service Providers increasingly need to meet performance requirements for *ingress* traffic since that traffic includes, for example, player movements in real-time games, voice and video in enterprise conferences, and video/image uploads for AI processing in the cloud [17, 74]. This reality is a departure from traditional CDN traffic patterns where ingress traffic was primarily small requests and TCP acknowledgments.

Despite Service Providers' efforts to meet variable service requirements [71, 51, 36, 58, 75, 55, 65], meeting requirements is still challenging due to issues *outside* Service Provider control.

First, Service Providers lack control over which ingress path traffic takes since BGP, the Internet's interdomain routing protocol, computes paths in a distributed fashion, giving each intermediate network a say in the ingress path. BGP chooses a single path which may not be the best one for meeting a given requirement. Service Providers cannot even unilaterally control the part of the path closest to them—upgrading peering capacity requires coordination among multiple parties [95].

Second, dynamic factors outside Service Provider control make satisfying requirements even harder. Peering disputes can lead to congestion on interdomain links and inflated paths [30, 21, 25], and DDoS attacks still bring down sites/services [90, 78, 88, 56], despite the considerable effort in mitigating DDoS attack effects [79, 111]. Moreover, recent work [63, 84, 15] and blog posts [47, 60, 77, 69, 37, 70] show that user traffic demands are highly variable due to flash crowds and path changes and so can be hard to plan for.

2.2 Routing Traffic to Service Providers

One way of meeting service requirements, despite this lack of control, is to (a) configure a good set of interdomain paths and (b) balance traffic on these paths to satisfy requirements. Existing systems balance traffic over egress interdomain paths [95, 114, 38, 59], but it remains unclear how to configure and balance traffic across a good set of *ingress* interdomain paths.

Today, most Service Providers either use anycast pre-

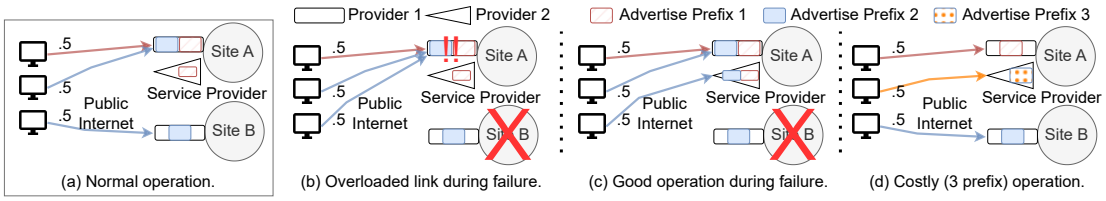


Figure 1: In normal operation traffic is split between two sites by directing half the traffic to each prefix (a). Even though there is enough global capacity to serve all traffic when site B fails, there is no way to split traffic across multiple providers given available paths, leading to overload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

fix advertisements to provide low latency and high availability at the expense of some control [8, 54, 119, 118], or `unicast` prefix advertisements to direct users to specific sites [98, 57, 95, 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 [119]. Prior work shows that this availability comes with higher latency in some cases [64, 54]. `Unicast` gives clients lower latency than `anycast` by advertising a unique prefix at each site [98, 13], but can suffer from reliability concerns, taking as much as an hour to shift traffic away from bad routes [55].

2.3 Limitations of Current Approaches

Too Specific: Recent work has similarly noted that, by offering better interdomain routes, Service Providers can achieve better performance [30, 8, 115, 118, 55]. These solutions advertise different prefixes to subsets of all peers/providers to offer users more paths. However, these solutions optimize specific objectives, mostly steady-state latency, and it is unclear how to extend their approaches to other objectives (§5.4), especially those with capacity constraints.

Figure 1 shows how this specific focus on a single objective could lead to reliability issues, *e.g.,* during a site failure. In normal operation, user traffic is split evenly across two prefixes and achieves low latency (Fig. 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 state-of-the-art systems that provide low (steady-state) latency from users to Service Provider networks such as `AnyOpt` [115] and `PAINTER` [55].

Too Reactive: To enhance reliability, some Service Providers propose systems that react to changing conditions—for example, `TIPSY` drains traffic from overloaded links/sites [71]. However, reactivity still requires time to change to a new state, which can lead to short-term service degradation and additional uncertainty at a time where unexpected things are already occurring. A more desirable approach would be to never change announcements carrying live traffic, which is known to sometimes lead to outages [6, 49, 10, 65].

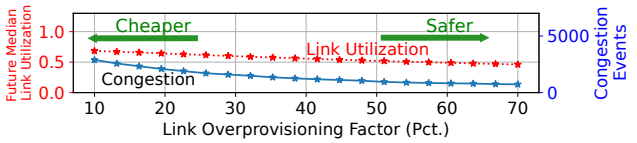


Figure 2: Planning for peak loads to avoid overloading requires inefficient overprovisioning.

Too Costly: Another approach is to overprovision resources to handle transient peak loads [103, 1, 65]. Figure 2 demonstrates however, using longitudinal link utilization data from OVH cloud [85], that doing so can incur excessive costs. Meta also stated that peering capacity is hard to change since it cannot be unilaterally upgraded [95], and Microsoft said the same because of lead time [30].

Upgrading capacity has fixed costs (*e.g.,* fiber, router backplane bandwidth, line card upgrades, CDN servers near peering routers) and variable costs (*e.g.,* power, transit), both of which scale with demand and Service Providers strive to keep low [86, 1, 15, 99]. We model deployment cost as correlated with the peering capacity over all links/sites, even though the actual relationship may be complex and additional peering capacity itself is not prohibitively expensive.

To generate Figure 2 we first split the dataset into successive, non-overlapping 120-day planning periods and compute the 95^{th} percentile ingress link utilization ("near-peak load") for each link/period. The dataset reports utilizations over 5-minute intervals. We then assign future capacities for each period by setting each link's capacity for the next period as the near-peak load in the current period multiplied by some overprovisioning factor (X-axis variable). We then compute the average link utilization in the next period and the total number of links on which we see $\geq$ 100% utilization in at least one 5-minute interval.

Figure 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 50% utilization.

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 satisfy 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 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 [80]. IPv6 is not a good alternative, (a) as IPv6 is not supported in all access networks [35] and (b) IPv6 addresses take 8× the amount of memory to store in a router so would pollute global tables even more. Prior work noted the same but found that advertising around 50 prefixes was acceptable [115, 55], and most Service Providers advertise fewer than 50 today according to RIPE RIS BGP data [101]. We also verified with engineers at six Service Providers that this challenge important to address.

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

Finding that right subset of paths to expose that satisfies performance objectives, however, is hard since there are exponentially many subsets to consider, and each subset currently needs to be tested (*i.e.,* advertised via BGP) to see how it performs. The number of subsets is on the order of 2 to the number of ingresses Service Providers have, which, for some Service Providers (including Vultr, which we measure) [105], is $> 2^{10{,}000}$. As measuring this many advertisements is intractable, we have to predict how different subsets of paths perform, which is challenging since interdomain routing is difficult to model [96, 71].

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) and a traffic allocation to those paths. SCULPTOR computes BGP advertisements proactively, only placing live traffic on them after convergence. For objectives we consider, resulting optimization problems are often large with more than 100M constraints and 2M decision variables (§3.2.2). Our framework breaks this challenging problem into manageable components — Figure 3 shows the interactions among these components.

Minimizing an objective function requires evaluating it with many different inputs, but performing such measurements (*i.e.,* advertising prefixes) could take years at our problem size (§3.3.1) and so is not scalable. Instead, we estimate the objective using a "Probabilistic Routing Model" (§3.3.2) and update this model over time by "Advertising & Measuring" a small number of advertisements in the Internet using entropy-based "Exploration" (§3.3.3).

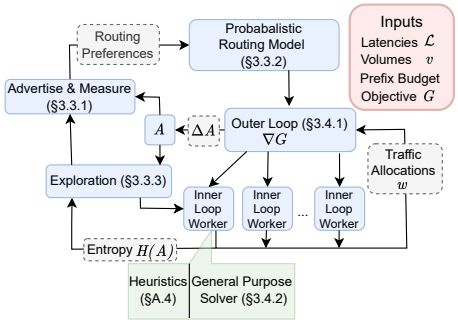


Figure 3: SCULPTOR advertisement computation overview.

We then minimize the objective function using gradient descent ("∇G") which, at each iteration, requires solving millions of sub-problems for traffic allocations ("Inner Loop Workers", §3.4.1). These sub-problems can either be solved exactly with a "General Purpose Solver" (§3.4.2) or approximately with efficient heuristics (**??**).

3.2 Problem Setup and Definitions

3.2.1 Setting

Service Providers offer their services from tens to hundreds of geo-distributed sites [73]. The sites for a particular service can serve any user, but users benefit from reaching a low-latency site for performance. Sites consist of sets of servers that have an aggregate capacity. Service Providers also connect to other networks at sites via dedicated links or shared IXP fabrics which we call peering links. Each such link also has a capacity. When utilization of a site or link nears/exceeds the capacity, performance suffers, so Service Providers strive to avoid high utilization [30, 15]. Resources can also fail due to, for example, physical failure and misconfiguration.

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

We model interdomain paths from users to the Service Provider as non-overlapping, except at the peering link through which each path ingresses to the Service Providers' deployment. This model is different from considering paths as equivalent to inter-AS adjacencies since many ASes peer with Service Providers at multiple locations.

We assume the peering link is the bottleneck of each path,

for two reasons. First, this link increasingly represents the most important part of the path for Service Providers due to Internet flattening [4]. Second, other parts of the path are less interesting for traffic engineering—last-mile bottlenecks are common to all paths for a user, and existing systems perform traffic engineering in intermediate networks. Handling unexpected bottlenecks can also be viewed as an "Unseen Scenario", which we evaluate (§3.2.3), demonstrating that SCULPTOR handles violations of this assumption. A user may have many paths to a site through different links. A path to a prefix will be through one of the corresponding ingress links over which that prefix is advertised.

We consider users at the granularity of user groups (UG*s*), where a UG refers to user networks that route to the Service Provider similarly, but could mean different things in different instantiations of our system (*e.g.,*/24 IPv4 prefixes, metros). UG*s* generate steady-state traffic volumes, $v_{\texttt{UG}}$, 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, 115, 55].

We assume that the Service Provider exposes some technology to SCULPTOR for directing traffic towards prefixes. Examples include DNS [98, 13, 8, 57], multipath transport [89, 22, 23], or control-points at/near user networks [92, 55]. DNS offers slow redirection due to caching [55] but is the most deployable, whereas Service Provider-controlled appliances offer precise control but may be hard to deploy. Multipath transport will eventually see wide enough deployment to be used by all Service Providers. Today, MPTCP is installed in iOS [3] and Ubuntu 22 [31]; all applications can use MPQUIC [117]. As noted in PAINTER [55], Service Providers with more resources and incentives can invest in more effective redirection mechanisms (such as developing MPQUIC tooling). Moreover, not every objective function that SCULPTOR optimizes needs high-speed redirection (such as satisfying different traffic classes).

3.2.2 General Formulation

The problem of finding advertisement configurations that admit good assignments of user traffic to paths is a multivariate optimization problem over both configurations and traffic assignments. We tabulate important variables and expressions in **??**.

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 [115, 55]). Implementing this configuration (*i.e.,* advertising prefixes via BGP sessions) results in routes from UG*s* to the Service Provider. The assignment of traffic to resulting routes from this configuration is given by the nonnegative real-valued vector w, whose entries specify traffic allocation for each UG along each route.

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

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

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

The first constraint requires that traffic assignments be nonnegative and that configurations are binary. The second constraint requires that all user traffic $v(\texttt{UG})$ is assigned, and none is assigned to nonexistent paths. Capacity constraints depend on the choice of G.

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) performing traffic engineering over different traffic classes. The framework likely accommodates objectives that have been met in other networking settings [99, 95, 109, 55, 115, 13, 15, 48, 38, 66] but, like other work that has used gradient descent with success, we sacrifice the ability to provide a formal characterization of which objective functions are possible for an approach that lets us approximately optimize multiple criteria.

With our interdomain path model (§3.2.1), steady-state path latency for a UG to a prefix is uniquely determined by UG and the corresponding peering link over which the UG ingresses (but latency may change, which we evaluate in Section 5.2.2. Let the latency for a user UG via a link l be $\mathcal{L}(\texttt{UG}, l)$. G is then given by the following.

$$G(R(A), w) = \frac{1}{\sum_{\texttt{UG}} v(\texttt{UG})} \sum_{p,\texttt{UG}} \mathcal{L}(\texttt{UG}, R(A)(p,\texttt{UG}))w(p,\texttt{UG}) \; + \; \beta M$$
$$\text{s.t.} \; \frac{\sum_{R(A)(p,\texttt{UG})=l} w(p,\texttt{UG})}{c(l)} \leq M \quad \forall l \tag{2}$$

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

Unseen Scenarios To encourage good solutions to Equation (1) not only in steady-state but also in unseen conditions (failures, shifting traffic distributions), we add a term to G given by $\sum_l \alpha_l G(R(A * F_l), w)$. The vectors F_l are binary vectors defined so that multiplying advertisements, A, by these vectors simulates withdrawal/failure on a link/site (*i.e.,* zeroing out the corresponding components). This term (*i.e.,* regularizer) intuitively encourages advertisement solutions to provide UG*s* good primary and backup paths, thus preparing for dynamic conditions. In **??** we discuss why this regularizer helps us find a good global minimum by drawing analogies to the deep learning literature.

Different Traffic Classes To demonstrate that SCULPTOR works with other useful objectives, we consider a separate problem where we try to route traffic with different performance requirements. Service Providers already perform multi-class traffic engineering on their private WANs [38, 48] and would increasingly benefit from such capabilities in an interdomain setting due to varied service offerings (§2.1), but they currently have no capability to do so. We use a similar scenario to the private WAN setting where one traffic class (high-priority) should be routed with low latency, and the other (low-priority) should not congest the high-priority traffic. A key additional challenge in the interdomain setting is that we cannot use priority queueing to ensure that high-priority traffic is not congested since we do not control queue behavior on the path. The objective function is a weighted combination of the average latency of high-priority traffic and the amount of high-priority traffic that is congested. We also constrain the maximum low-priority overprovisioning on any link so that not all low-priority traffic lands on one link, as this solution would lead to poor low-priority goodput.

3.3 Predicting Interdomain Routes

Solving Equation (1) is challenging because we need to compute $G(R(A), w)$, but measuring $R(A)$ exactly requires advertising prefixes in the Internet which can only be done infrequently to avoid route-flap-dampening. The optimization problem (detailed below) requires evaluating $G(R(A), w)$ for millions of different A which could take a hundred years at a rate of advertising one strategy every 20 minutes. Hence we model, instead of measure, UG paths and improve this model over time through relatively few measurements.

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 randomly do/do not advertise those prefixes via random ingresses. We do not use a completely random initialization since anycast and unicast have their benefits (§2.2).

During optimization we measure $R(A)$ by occasionally advertising the corresponding prefixes via peerings as specified by A and measuring routes taken by UG*s* to each prefix. We alternate between measuring advertisements we think are good (§3.4) and ones we think offer useful information (§3.3.3).

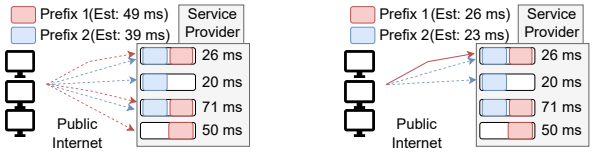
3.3.2 Probabilistic UG Paths

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

Our probabilistic model assumes a priori, for a given UG towards a given prefix, that all ingress link options that prefix is advertised to that are reachable from a UG are equally likely. (Providers can discern the set with high reliability based on which ingresses the UG is reachable from.) Upon learning that one ingress is preferred over the other, we exclude that less-preferred ingress as an option for that UG in all future calculations for all prefixes for which both ingresses are an option. Prior work used a similar routing model [115, 96, 55] but did not extend it to deal with general objectives nor did it treat routing probabilistically.

A useful property of many choices of objective function G is that we do not need to approximate the routing function perfectly to solve Equation (1). For example, most UG*s* have similar latency via their many paths to a single site [94], so when minimizing latency as in Equation (2) we effectively 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 4, where SCULPTOR refines a latency estimate towards an unmeasured



(a) (b)

Figure 4: SCULPTOR initially estimates latency from this UG to both the red and blue prefixes with the average over possible ingress latencies (4a). A priori, no prefix is advertised. SCULPTOR then advertises the red prefix and learn that the first ingress has higher preference than the third and fourth (4b). SCULPTOR uses this information to refine its latency estimate towards the blue prefix (since the third ingress is no longer a possible option) without advertising the blue prefix, saving time.

prefix (blue) using measurements towards other prefixes (red). Even though SCULPTOR has 50% confidence in which path the user takes (*i.e.,* in $R(A)$), SCULPTOR knows that the user's latency towards the blue prefix will be about 23 ms (*i.e.,* either 26 ms or 20 ms) and so SCULPTOR obtains a better estimate of the objective function.

3.3.3 Exploration to Refine $R(A)$

Probabilistic estimates of G could be inaccurate until we learn UG preferences. To refine its model, SCULPTOR periodically measures $R(A)$ on adjacent configurations to the configuration at the current optimization step (§3.4). By adjacent configurations, we mean ones that differ from the current configuration, A, by one entry, or configurations representing a single failure ($A \times F_l$). SCULPTOR measures adjacent configurations since it uses a descent-based optimization (§3.4) which benefits from knowing whether nearby strategies are better. Of the adjacent options, we select the one with the highest entropy. (One could use other measures to determine how to explore, but we found that entropy worked well).

Intuitively, a configuration has high entropy if SCULPTOR knows very little about what the resulting routes will be and if the different potential routes will lead to large differences in the objective function, leaving SCULPTOR unable to judge how useful the configuration will be. Specifically, for each adjacent configuration, SCULPTOR estimates the objective function's distribution at that configuration using Monte Carlo methods (as in Section 3.3.2) and computes the Shannon entropy **TBD: cite** of the estimated distribution.

3.4 A Two Pronged Approach

Even with our probabilistic model, we cannot solve Equation (1) directly since it is a mixed-integer program with millions of constraints which is too large for off-the-shelf solvers. Greedy [55] and random [115] approaches find good solutions in this setting for simple objectives such as steady-state latency, but, without more intelligent search through the large space, those approaches can converge to poor solutions on other objectives (§5.4).

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

3.4.1 Outer Loop: Gradient Descent

To apply gradient descent to our problem, we extend A to have real (instead of binary) entries between 0 and 1 and threshold its entries at 0.5 to determine if a prefix is advertised to a certain peer/provider. We approximate gradients between adjacent advertisements by computing the expected value of $G(R(A), w)$ at each advertisement and continuously interpolating G at intermediate configurations using sigmoids (like prior work in a different domain [61]). Since there are too many gradients to compute, we subsample gradient entries and track the largest ones (like prior work in a different domain [97]).

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

Convergence: Gradient descent converges to a local minimum for bounded objectives [62]. Our evaluations show that SCULPTOR finds good solutions over a wide range of simulated topologies (§4.1), and converges quickly with thousands of UGs and ⟨peering, prefix⟩ pairs (§5). We comment on which problem properties will likely lead to faster convergence to a lower minimum in **??** by drawing analogies to the deep learning literature. Allocating higher prefix budgets and adding richer advertisement capabilities (*e.g.,* BGP community tagging) can lead to convergence to a better minimum, which is an area for future work.

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 Monte Carlo simulations and the scaling behavior of each traffic allocation, which depends on the objective.

Convergence depends on the objective, G. Our objectives (§3.2.3) and many others can be expressed as linear programs, so solvers can find globally optimal solutions. Non-convex objectives such as Cascara's sometimes have efficiently computable solutions [99].

4 Implementation

We prototype SCULPTOR on the PEERING testbed [93], 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 [55], and selected targets according to assumptions about Vultr cloud's client base.

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

In our evaluations, we limit our focus to clients who had a route through at least one of Vultr's direct peers (we exclude route server peers [12]). Vultr likely peers with networks with which it exchanges a significant amount of traffic [95], so clients with routes through those peers are more likely to be "important" to Vultr. We found 700k /24s with routes through 1086 of Vultr's direct ingresses. In an effort to focus on interesting 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 many 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 (§3.4.1) in Python [81]. We set $\alpha_l = 4.0$ and set the learning rate to 0.01 with decay over iterations. We solve traffic allocations (§3.4.2) with Gurobi [83].

Experiments in the Internet We assess how SCULPTOR performs on the Internet using RIPE Atlas probes [100], 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). We limit the scale of our deployment to 10 sites to avoid reaching RIPE Atlas daily probing limits (15k traceroutes/day). We choose a deployment with high geographic density rather than greater geographic coverage, as we believe the proximity creates a more interesting routing surface to solve (differences from unicast could be smaller, for example). These 10 sites were Miami, New York, Chicago, Dallas, Atlanta, Paris, London, Stockholm, Sao Paulo, and Madrid. Choosing RIPE Atlas probes to maximize network coverage and geographic diversity, we select probes from 972 networks in 38 countries which have paths to 484 unique ingresses. Each probe has paths via approximately 60 ingresses. We use 12 prefixes.

Simulations We also evaluate SCULPTOR by simulating user paths which allows us to conduct more extensive evaluations, as experiments take less time and use clients in more networks. We compute solutions over many random routing preferences, demands, and subsets of sites to demonstrate that SCULPTOR's benefits are not limited to a specific deployment. We evaluate SCULPTOR over deployments of size 3, 5, 10, 15, 20, 25, and 32 sites. Each size deployment is run at least 10 times with random subsets of UGs, UG demands, and routing preferences. The distribution of the number of /24s per peer is Pareto-like, so we consider random subsets of /24s through each ingress in a way that balances the number of unique /24s per ingress. Over all scenarios, we consider paths from clients in 52k prefixes (representing 31% of APNIC population [40]) to 873 ingresses. We use one tenth of the number of ingresses as the number of prefixes in our budget (10 prefixes for 3 site deployments, 60 prefixes for 32 site deployments). Prior work found that using tens of prefixes to improve performance was a reasonable cost [115, 55].

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 exists since we are not a Service Provider, but a Service Provider could measure this using `IPFIX`, for example.) We overcome this limitation using two methods corresponding to our two deployments in Section 4.2.

Experiments in the Internet For our first method of inferring client ingress links, we advertise prefixes into the Internet using the PEERING testbed [93], and measure actual ingress links to those prefixes using traceroutes from RIPE Atlas probes [100]. Specifically, we perform IP to AS mappings and identify the previous AS in the path to Vultr. This approach has limited evaluation coverage, as RIPE Atlas probes are only in a few thousand networks. In cases where we cannot infer the ingress link even from a traceroute, we use the closest-matching latency from the traceroute to the clients' (known) possible ingresses. For example, if an uninformative traceroute's latency was 40 ms to Vultr's Atlanta site and a client was known to have a 40 ms path through `AS1299` at that site, we would say the ingress link was `AS1299` at Atlanta.

Simulations The second method we use to infer ingress links is simulating user paths by assuming we know all user routing preference models (§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, though prior work found it held in 90% of the cases they studied. However, we found that our key evaluation results (§5) hold regardless of how we simulated routing conditions (we also tried random preference assignments).

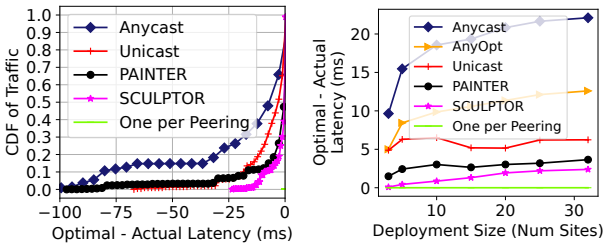
Given either method of inferring client ingress links (RIPE Atlas/simulations), we then measure paths to an `anycast` prefix and assign resource capacities as some overprovisioned percentage of this catchment. (Discussions with operators from Service Providers suggested that they overprovision using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

5 Evaluation

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, 16, 8, 107, 87, 113, 118].



(a) (b)

Figure 5: `SCULPTOR` provides the lowest steady-state latency both on the Internet (5a) and in simulation (5b).

`unicast`: A single prefix announcement to all peers/providers at each site (one per site). Another common strategy used in today's deployments [98, 57, 13].

`AnyOpt`/`PAINTER`: Two proposed strategies for reducing steady-state latency compared to `anycast` [115, 55]. 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 `AnyOpt` takes a long time to compute.

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

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

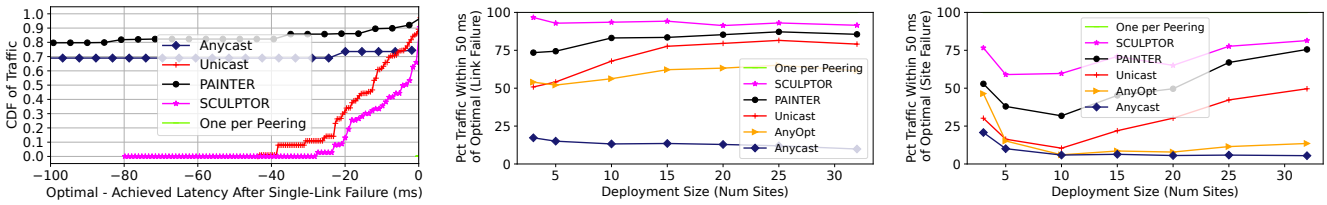
5.2 Handling Unseen Conditions

In minimizing user latency (§3.2.3), `SCULPTOR` achieves that objective both during steady-state and also during failures and conditions unseen during learning.

For context, at the scale of Service Providers that serve trillions of requests per day, improving a few percent of traffic by tens of milliseconds represents a significant improvement. Service Providers recently emphasized that small percentage gains are important [29, 110].

5.2.1 Lower Latency in Steady-State

Internet Deployment Figure 5a shows a CDF of the difference in latency between each solution and `One-per-Peering` for all `UG`*s*, weighted by traffic. `SCULPTOR` outperforms other solutions and is only 2.0 ms worse than (the unreasonably expensive) `One-per-Peering` solution on average, whereas the next best solution, `PAINTER`, is 5.5 ms worse. `SCULPTOR` also serves 91.8% of traffic within 10 ms of the latency with which `One-per-Peering` serves it, whereas the same is true for only 88% of traffic for `PAINTER`.



(a) (b) (c)

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

Simulations Figure 5b shows the average latency compared to the One-per-Peering solution over all simulated deployments at each deployment size. Average latency for SCULPTOR ranges from 0.1 to 2.4 ms worse than One-per-Peering. The next-best solution (PAINTER) is on average 1.1 ms and 2.2 ms worse than SCULPTOR. Interestingly, unicast (5.7 ms worse than One-per-Peering) performs better than AnyOpt (10.0 ms worse than One-per-Peering), which could be due to the different setting AnyOpt was designed for. AnyOpt was designed to minimize latency without capacity constraints over provider connections, which does not capture that many Service Providers have many peers and limited capacity [95].

These average latency improvements translate into quantifiable routing inefficiency for different fractions of traffic (we include full results in **??**). SCULPTOR has on average 94.9% traffic within 10 ms of the One-per-Peering solution, 99.2% within 50 ms, and 99.9% within 100 ms. These percentages compare favorably to the next-best solution, PAINTER, which has on average 92.3% traffic within 10 ms of the One-per-Peering solution, 97.7% within 50 ms, and 99.2% within 100 ms.

5.2.2 Better Resilience to Failure

We next assess SCULPTOR's ability to gracefully handle a type of *unseen* condition—ingress failures and site failures. Examples include excessive DDoS traffic on the link/site (thus using the link/site as a sink for the bogus traffic), physical failure, resource draining, changes in latency on a path, and planned maintenance. Figure 6 demonstrates that SCULPTOR shifts traffic without overloading alternate links/sites more effectively than any other solution, *without reactive BGP changes* which could cause further failure (§2.3).

Here, 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 that use that component in steady-state. For example, if the Tokyo site fails, we report on the post-failure latency of UGs that were served from Tokyo before the failure and not of other UGs.

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

Internet Deployment Figure 6a demonstrates that SCULPTOR offers lower latency paths for more UGs during single link failure in realistic routing conditions. On average, SCULPTOR is 7.9 ms higher latency than One-per-Peering, compared to unicast which is 14.2 ms higher than One-per-Peering. PAINTER struggles to find sufficient capacity for UGs, overloading 69.6% of traffic.

Site failures (shown in **??**) show similar results. On average, SCULPTOR is 13.1 ms higher latency than One-per-Peering, while the next best solution, unicast, is 25.1 ms higher. PAINTER again performs poorly, with 95% of traffic overloaded during site failure.

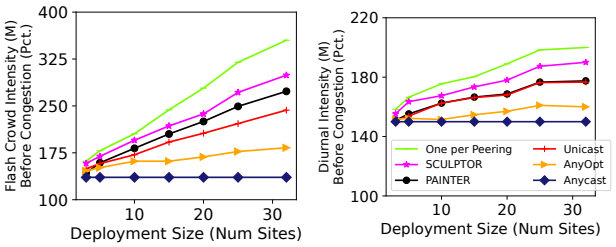
Simulations We show the fraction of traffic within 50 ms of the One-per-Peering solution for link and site failures in Figure 6b and Figure 6c (further results are in **??**).

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

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

5.2.3 Efficient Infrastructure Utilization

Figure 7 shows that installing more capacity to handle peak loads during unseen scenarios is not always necessary with better routing — SCULPTOR finds ways to distribute load over existing infrastructure to accommodate increased demand.



(a) (b)

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

We quantify this improved infrastructure utilization under two realistic traffic patterns that SCULPTOR *did not explicitly consider in its learning process* — flash crowds and diurnal effects.

Methodology We define a flash crowd as a transient traffic increase for users in a region. Examples include content releases that spur downloads in a particular region, and localized DDoS attacks. Since increased demand is localized, we can spread excess demand to other sites, which is a cheaper option than installing more capacity (see Section 3.2.1 for our cost model). Links are still provisioned 30% higher than anycast traffic volume as in the rest of Section 5.2.

To generate Figure 7, we identify each client with a single "region" corresponding to the site at which they have the lowest possible latency ingress link. For each region individually, we then scale each client's traffic in that region by M% and compute traffic to path allocations. If there are S sites in the deployment, we thus compute S separate allocations per M value where each allocation assumes inflated traffic in exactly one region, but all the allocations are across a single set of routes calculated based on the original (non-flash) traffic. For a target region, we increase M until any link experiences overloading, then find the lowest such M value across regions. For example, if a 60% increase in traffic for Atlanta clients creates overload (and no values < 60% did for any region), we call M = 160% the critical value of M.

Our diurnal analysis in Figure 7b uses a similar methodology. We define a diurnal effect as a traffic pattern that changes volume according to the time of day. Diurnal effects might be different for different Service Providers, but a prior study from a Service Provider demonstrates that these effects can cause large differences between peak and mean site volume [15]. 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 **??**. 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 7 plots the average over simulations of critical M values that cause overloading for each deployment size under flash crowds and diurnal effects computed using simulated deployments.

Figure 7a shows that SCULPTOR finds ways to route more traffic during a flash crowd without overloading than other solutions. For deployments with 32 sites, SCULPTOR can handle flash crowds at 2× more than the expected volume, creating a 3× savings in provisioning costs compared to anycast, and 26% more savings than PAINTER. Figure 7b shows that SCULPTOR also handles more intense diurnal traffic swings, allocating traffic to paths without overloading for 24% more intense swings than both PAINTER and unicast with 32 sites. Hence, instead of scaling deployment capacity to accommodate peak time-of-day traffic, Service Providers can re-distribute traffic to sites in off-peak time zones.

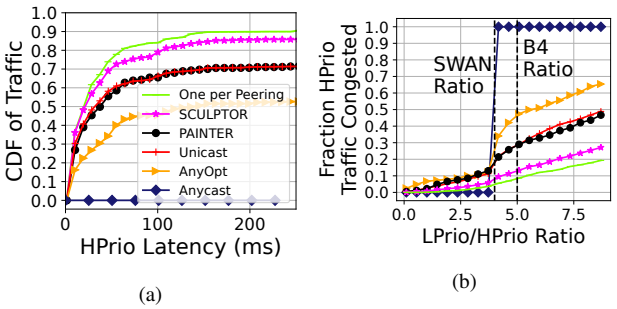
Using backup paths does not imply reduced performance as Service Providers can move less latency-sensitive traffic onto these backup paths so as to not impact high-level applications. We explore this idea in Section 5.3.

5.3 Handling Multiple Traffic Classes

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

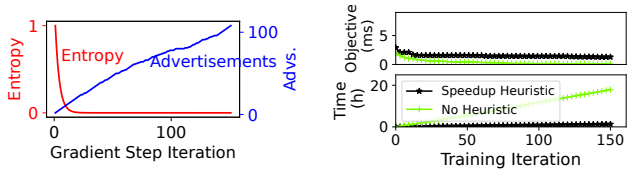
In Figure 8b we vary the amount of low-priority traffic as a multiple of high-priority traffic volume and compute the fraction of high-priority traffic congested. Links are provisioned to 5× the high-priority traffic volume (different from Section 5.2), as that is roughly the LPrio/HPrio ratio reported in prior work [38, 48]. There is insufficient global capacity to route all traffic for all LPrio/HPrio over 4.0, thus the jump in anycast congestion in Figure 8b. SCULPTOR finds strategies that allow us to route more low-priority traffic with less congestion than all other approaches. For example, when LPrio/HPrio = 5, SCULPTOR achieves half as much congestion as PAINTER and unicast.

Figure 8a also shows that SCULPTOR routes traffic with lower latency than other solutions (intercepts at the right of the graph show fractions of high-priority traffic not congested). Figure 8a uses LPrio/HPrio = 4, a midpoint between the ratios seen in SWAN [38] and B4 [48].



(a) (b)

Figure 8: SCULPTOR routes high-priority traffic with low latency (8a) and minimal congestion from low-priority traffic (8b). We inferred LPrio/HPrio ratios for SWAN [38] and B4 [48] from those papers.



(a) (b)

Figure 9: SCULPTOR's model entropy decreases exponentially with few path measurements (9a). Heuristic speedups can also speed computation, but may sacrifice convergence (9b).

5.4 Why SCULPTOR Works

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

Conducting Fewer Advertisements SCULPTOR only conducts advertisements for exploration (§3.3.3) or for strategies that SCULPTOR thinks will yield good performance (§3.4). Since the key limited resource in finding good advertisements is time, it is essential that SCULPTOR is confident that its gradient descent leads it to good performing advertisements (or else it would waste time exploring bad ones).

PAINTER measures advertisements that it thinks might be good, but its greedy approach means that it only considers a single scenario per advertisement iteration. AnyOpt spends time measuring potentially uninformative advertisements. Hence, both AnyOpt and PAINTER conduct a large number of advertisements on the Internet relative to the configurations they estimate, limiting the number of configurations they can explore. Figure 9a shows how the maximum entropy of the distribution of G on unmeasured advertisements (§3.3.3) exponentially decreases as SCULPTOR makes these advertisements on the Internet, and so it quickly grows confident that it does not need to issue more advertisements to find good strategies. This quick convergence manifests since we only have to predict the objective, G, not the paths (§3.3.2).

Horizontal Scaling SCULPTOR converges faster given more compute since gradient computations are parallelizable. PAINTER does not scale horizontally since it uses greedy search [55]. Some objectives additionally admit heuristics for finding approximately optimal traffic allocation solutions (*e.g.,* **??**). Other work similarly uses heuristics to quickly solve challenging problems [99, 48, 84]. Figure 9b shows that our heuristic (**??**) marginally hurts SCULPTOR's convergence (2 ms average latency difference) but requires drastically less compute (50× speedup).

6 Related Work

Egress Traffic Engineering Prior work noted that traffic from Service Providers to users sometimes experienced suboptimal performance due to BGP's limitations [114, 95, 59]. SCULPTOR works in tandem with these systems, similarly working with BGP, but to engineer ingress traffic. Other work also shifts traffic to other links/sites to lower cost [116, 99, 15]. SCULPTOR adapts this idea in a new way but instead uses the public Internet to carry traffic to lower costs.

Ingress Traffic Engineering We compare SCULPTOR to other work on ingress traffic engineering [8, 67, 115, 119, 118, 71, 55] both in evaluation (§5) and discussion (§2.3). FastRoute coarsely balances users across anycast rings to respond to changing conditions [30], giving Service Providers much less control over the specific path users take. PECAN [106] 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.

Other prior work built a BGP playbook to mitigate DDoS attacks [91], but it is unclear if those strategies would scale to larger Service Providers (they tested on a few sites/providers). SCULPTOR deals with arbitrary unseen conditions over thousands of peering links. Other work and companies create overlay networks and balance load through paths in these overlay networks to satisfy latency requirements [104, 45, 2, 18, 63]. SCULPTOR can work alongside these overlay networks by advertising the external reachability of these nodes in different ways to create better paths through the overlay structure.

Intradomain Failure Planning Service Providers have shown interest in reducing the frequency/impact of failures in their global networks [26, 36, 58, 65]. SCULPTOR works alongside these systems by, for example, enabling Service Providers to shift traffic away from failed components/regions while still retaining good performance. Prior work also shifted traffic during peak times to limit cost/congestion [15], but did so using a private backbone. SCULPTOR's benefits are orthogonal to this prior work and useful for Service Providers without private backbones, as they use the public Internet to realize the same result. Other prior work tried to plan intradomain routes to minimize the impact of k-component failures [109, 66, 11, 50, 15]. SCULPTOR solves different challenges

that arise due to the lack of visibility/control into potential paths and their capacities in the interdomain setting.

7 Discussion & Conclusions

Unilateral control over intradomain traffic has enabled technology that improves performance, reliability, and flexibility, including fast reroute for quick recovery from failure [42], virtual routing and forwarding for flexible traffic engineering [43], and virtual output queueing for differentiated service [82]. Researchers have proposed making similar functionality available in interdomain settings, but, after decades of little adoption, it seems unlikely that such technologies will be widespread enough for Service Provider use. For example, intserv/diffserv [41] and L4S [44] are proposed frameworks for achieving differentiated service but show no signs of deployment. MIRO is a flexible interdomain multipath routing protocol that similarly shows no signs of deployment [112].

Rather than require any changes to the Internet, `SCULPTOR` uses a simple, black-box model of interdomain routing, BGP's flexibility, and the unused capacity of a Service Provider's global resources to give operators the benefits of those aforementioned technologies in the interdomain setting. For example, `SCULPTOR` can configure multiple paths per user to enable traffic failover and can set up different routes for different traffic classes to enable virtual output queueing despite lack of control on the interdomain path. `SCULPTOR` is a step towards providing Service Providers with programmable interdomain networking primitives so that they can give us the performance that our services increasingly need.

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, Sophia Yoo, Benjamin Herber, Jennifer Rexford, and Maria Apostolaki. [n.d.]. {TANGO}: Secure Collaborative Route Control across the Public Internet. In *NSDI 2024*.

[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] 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*.

[15] 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*.

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

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

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

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

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

[21] 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

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

[23] Quentin De Coninck and Olivier Bonaventure. 2021. Multiflow QUIC: A generic multipath transport protocol. *IEEE Communications Magazine* (2021).

[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] Google. 2024. Google IPv6. https://google.com/intl/en/ipv6/statistics.html

[36] 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*.

[37] 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

[38] 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*.

[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] Geoff Huston. 2014. How Big is that Network? labs.apnic.net/?p=526

[41] IETF. 2000. A Framework for Integrated Services Operation over Diffserv Networks. https://datatracker.ietf.org/doc/html/rfc2998

[42] IETF. 2005. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. https://datatracker.ietf.org/doc/html/rfc4090

[43] IETF. 2006. BGP/MPLS IP Virtual Private Networks (VPNs). https://datatracker.ietf.org/doc/html/rfc4364

[44] IETF. 2023. Low Latency, Low Loss, and Scalable Throughput (L4S) Internet Service: Architecture. https://datatracker.ietf.org/doc/rfc9330/

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

[46] 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

[47] 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

[48] 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*.

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

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

[51] 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*.

[52] Bhaskar Kataria, Palak LNU, Rahul Bothra, Rohan Gandhi, Debopam Bhattacherjee, Venkata N Padmanabhan, Irena Atov, Sriraam Ramakrishnan, Somesh Chaturmohta, Chakri Kotipalli, et al. [n.d.]. Saving Private WAN: Using Internet Paths to Offload WAN Traffic in Conferencing Services. In *CoNEXT 2024*.

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

[54] 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*.

[55] 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*.

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

[57] 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*.

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

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

[60] 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

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

[62] 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).

[63] 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*.

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

[65] 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*.

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

[67] 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*.

[68] 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.

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

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

[71] 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*.

[72] 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).

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

[74] 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

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

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

[77] 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

[78] 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*.

[79] 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*.

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

[81] Yurii Nesterov et al. 2018. *Lectures on convex optimization*. Vol. 137. Springer.

[82] Juniper Networks. 2023. Understanding CoS Virtual Output Queues (VOQs). https://juniper.net/documentation/us/en/software/junos/traffic-mgmt-qfx/topics/concept/cos-qfx-series-voq-understanding.html

[83] Gurobi Optimization. 2024. Gurobi Optimizer. https://www.gurobi.com/downloads/

[84] 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*.

[85] 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*.

[86] 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*.

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

[88] 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

[89] 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*.

[90] 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

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

[92] 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*.

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

[94] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. [n.d.]. Internet Performance from Facebook's Edge. In *IMC 2019*.

[95] 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*.

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

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

[98] Patrick Shuff. [n.d.]. Building a Billion User Load Balancer. https://usenix.org/conference/lisa16/conference-program/presentation/shuff

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

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

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

[102] 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)*.

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

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

[105] Ticiane Takami. 2021. Project Myriagon: Cloudflare Passes 10,000 Connected Networks. https://blog.cloudflare.com/10000-networks-and-beyond/

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

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

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

[109] 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*.

[110] 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*.

[111] 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.

[112] Wen Xu and Jennifer Rexford. [n.d.]. MIRO: Multi-Path Interdomain Routing. In *SIGCOMM 2006*.

[113] 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*.

[114] 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*.

[115] 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*.

[116] 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*.

[117] 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*.

[118] 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*.

[119] 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*.