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

Paper #559, 12 pages body, 19 pages total

Abstract

Large cloud and content (service) providers help serve an expanding suite of applications that are increasingly integrated with our lives, but have to contend with a dynamic public Internet to route user traffic. To enhance reliability to dynamic events such as failure and DDoS attacks, large service providers overprovision to accommodate peak loads and reactively activate emergency systems for shifting excess traffic. We take a different approach with SCULPTOR, which proactively 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 17% on average, and enables service providers to achieve low latency objectives for up to 17% more traffic.

1 Introduction

Compiled on 2024/08/28at 11:23:36

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 [49, 46], whereas new applications such as virtual reality require $\leq$ 10 ms round trip latency [68] and $\leq$ 3 ms jitter [89]. 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 changing egress (out of the Service Provider's network) demands across links to meet performance objectives [82, 99, 36, 53, 86]. However, it is difficult to modify the part of the path from users to the Service Provider (*i.e.,* ingress traffic) with the same precision as in the egress case since that part 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, 65], DDoS attacks [71, 97, 78], flash crowds [41, 54, 57], new applications/business priorities (LLMs), and route changes [69, 63, 35, 64, 73, 65]. 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 [70, 76, 50, 46].

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 [90, 1, 59], 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 [100, 49], but those approaches have not been demonstrated to optimize other important objectives. FastRoute and TIPSY respond to changing loads to retain reliability [30, 65], 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 [86]).

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 useful objectives such as latency reduction, transit 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 ($> 2^{1,000}$ 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 (§5.5).

We prototype and evaluate our framework at Internet scale using the PEERING testbed [81] (§4), which is now deployed at 32 Vultr cloud locations [94]. 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 (we do not know the actual optimal). We found that, compared to other approaches, sparse increases the amount of traffic within 10 ms of the optimal by 3% in steady state (§5.2), 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 (§5.3).

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

Setting: Service Providers offer their services from tens to hundreds of geo-distributed sites. The sites for a particular service can serve any user, but users benefit from reaching a low latency site for performance. Sites consist of sets of servers which have an aggregate capacity. Sites can forward requests to other sites via private WANs, but such forwarding is undesirable since it uses valuable private WAN capacity [46]. 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, 17]. Resources can also fail completely due to, for example, physical failure and misconfiguration.

Upgrading capacity has fixed costs (*e.g.,* fiber, 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 [1, 17, 86]. We model deployment cost as correlated with the ingress peering capacity over all links/sites, even though the actual relationship may be complex. Hence, even though the peering capacity (*i.e.,* fiber) itself is not a significant cost, we model it as one since it is correlated with much more significant fixed/recurring costs throughout the deployment.

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 [49], which require very high reliability and are predicted to be a $60B industry by 2027 [40]. Gaming is a similarly important industry generating more revenue than the music and movie industries combined [6], but instead requires latency within 50 ms [68]. 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].

Despite Service Providers' efforts to meet variable service requirements [65, 45, 34, 52, 67, 49, 59], meeting require-

ments 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 [78, 70, 76, 50], despite the considerable effort in mitigating DDoS attack effects [71, 97]. 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 [57, 73, 17]. Operators regularly report these traffic surges on social media and blog posts [41, 54, 69, 63, 35, 64].

2.2 Routing Traffic to Service Providers

A way to protect against these challenges is to route traffic over different interdomain paths thus avoiding overwhelmed sites/links and balancing load. Existing systems balance traffic over egress interdomain paths [82, 99, 36, 53], 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 [9, 48, 104, 103], or `unicast` prefix advertisements to direct users to specific sites [51, 82, 14].

`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 [104]. Prior work shows that this availability comes with higher latency in some cases [58, 48]. `Unicast` can give clients lower latency than `anycast` by advertising a unique prefix at each site [14], but can suffer from reliability concerns potentially taking as much as an hour to shift traffic away from bad routes [49].

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, 100, 49] and egress traffic engineering [99, 36, 82, 53] also does not explicitly address this problem.

2.3 Limitations of Current Approaches

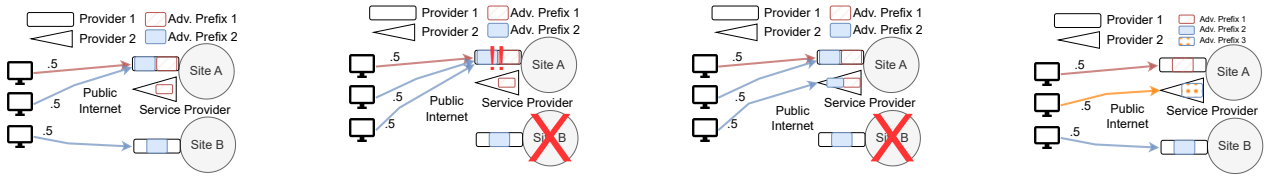
Too Specific: Some recent work has noted that specific objectives may be better met by offering better interdomain routes, achieving better latency and/or reliability [30, 9, 100, 103, 49]. These solutions advertise different prefixes to subsets of all peers/providers to offer users more paths. However, these solutions optimize specific objectives, mostly steady-state latency, and in Section 3.3 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 during a deployment failure. In normal operation, user traffic is split evenly across two prefixes and achieve 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` [100] and `PAINTER` [49], 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 [7, 43, 11, 59].

Too Conservative: Another common solution to satisfying diverse objectives is to overprovision resources to handle peak loads [90, 1, 59], but Figure 2 demonstrates that doing so can incur excessive costs. Figure 2 computes differences in peak utilization on links between different successive time periods, using longitudinal link utilization data from OVH cloud [74].

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



(a) Normal operation. (b) Overloaded link during failure. (c) Good operation during failure. (d) Costly (3 prefix) operation.

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

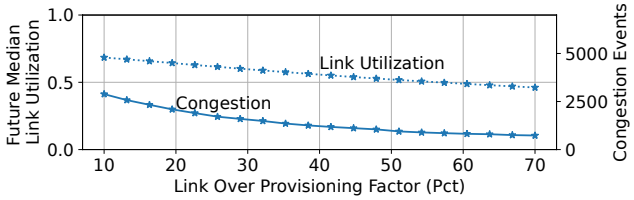


Figure 2: Planning for peak loads to avoid overloading requires inefficient overprovisioning.

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, 82].

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.) IPv6 is not a good alternative as there are fewer IPv6 paths since IPv6 peering is less common, IPv6 is not universally supported, and IPv6 routing entries take 8× the amount of memory to store in a router so would pollute global routing tables even more. Prior work noted the same but found that advertising around 50 prefixes was acceptable [100, 49], and most Service Providers advertise fewer than 50 today according to RIPE RIS BGP data [88]. 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 $> 2^{1,000}$. 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, and since there are too many possible changing conditions (failures, attacks) over which to assess these predictions.

3 Methodology

3.1 SCULPTOR Overview

SCULPTOR's high level goal is to find an advertisement strategy that optimizes an objective function to capacity constraints, even in unseen conditions. Considering capacity constraints when optimizing latency means that some traffic placement decisions become correlated since different users share the same capacity. Hence, considering capacity constraints makes computing optimized latencies a large optimization problem (10M constraints, 1M decision variables) (**??**).

Minimizing this objective function requires evaluating it with several different inputs, but performing such measurements (*i.e.,* advertising prefixes) takes time, and so is not scalable. It would take at least 20 minutes per test to avoid route flap dampening which translates to years of optimization given the problem sizes that we consider (millions of evaluations). Instead, we estimate latency by predicting paths. Predicting interdomain paths is hard, however, so we model paths probabilistically (§3.3.1), and update this model over time using a small number of measurements in the Internet (§3.5).

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

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

3.2 Problem Setup and Definitions

3.2.1 Setting and Goal

We aim to advertise relatively few prefixes to connected networks to meet objectives under changing conditions from user networks to the Service Provider subject to capacity constraints on links. Adding capacity constraints on sites (*e.g.,* servers) in addition to links would be a straightforward extension.

We assume that the Service Provider has some technology for directing traffic towards prefixes. Examples include DNS [14, 9, 51], multipath transport [77, 23], or control-points at/near user networks [80, 49]. DNS offers slow redirection due to caching [49] 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 [4] and Ubuntu 22 [31], and MPQUIC can be installed on any device in user space with applications [102]. SCULPTOR assumes that all users can be precisely directed to each prefix.

Service Providers connect to peers/providers at sites via physical connections we call peering links. Users route to the deployment through the public Internet to a prefix, over one of the peering links via which that prefix is advertised. The path (and therefore peering link) is chosen via BGP. We consider users at the granularity of user groups (UG*s*), which generally refer to user networks that route to the Service Provider similarly, but could mean different things to different Service Providers (*e.g.,*/24 IPv4 prefixes, metros). UG*s* generate known steady state traffic volumes, $v_{\texttt{UG}}$, and the Service Provider provisions capacity at links/sites to accommodate this load. We assume a system run by the Service Provider measures latency from UG*s* to Service Provider peering links l enumerated as $L_{\texttt{UG},l}$. (Having these measurements is a reasonable assumption [10, 14, 24**?** , 100, 49]). For a given UG, paths towards prefixes may be different and so may have different latencies.

3.2.2 Optimization Problem

We wish to find an advertisement configuration that admits good assignments of user traffic to paths which we cast as a multivariate optimization over both configurations and traffic assignments.

The advertisement configuration A can be represented by a binary vector. Each entry indicates whether we advertise/do not advertise a particular prefix to a particular peer/provider (similar to prior work [100, 49]). 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. The entries of the vector w 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 (*i.e.,* R is a mapping that creates mappings). For example, say $R(A) = f$, and $f(p, \texttt{UG}) = l$ — this notation means that the output of an advertisement configuration A under routing R is a function f that maps prefixes p and 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). The joint minimization over configurations and traffic assignments can then be expressed as the following.