The New (Pareto) Frontier of Cloud Routing

High Availability, Precise Control, or Configuration Stability — Choose Two

JIANGCHEN ZHU, Columbia University, United States
TOM KOCH, Columbia University, United States
ILGAR MAMMADOV, Columbia University, United States
SHUYUE YU, Columbia University, United States
KEVIN VERMEULEN, LIX, CNRS, Ecole Polytechnique, France
MATT CALDER, Meta, United States
ITALO CUNHA, Universidade Federal de Minas Gerais, Brazil
ETHAN KATZ-BASSETT, Columbia University, United States

Clouds provide latency-sensitive services to clients at geographically distributed sites. They need precise control of client-site mappings for performance and, if a site fails, fast failover to other sites without cascading failures due to failover-induced overload or risky routing reconfiguration. However, clouds today route clients to sites by anycast or unicast with DNS-based redirection—methods that compromise either control or availability. In fact, even beyond these existing techniques, we find that all general-purpose approaches for directing clients to sites face inevitable tradeoffs among control, availability, and routing stability. We then present new general-purpose routing techniques and demonstrate via Internet-scale experiments that they provide much better tradeoffs among these three goals than existing techniques. One of our techniques achieves anycast's advantages with much better control, and another speeds up unicast's failover while preserving its advantages. Our techniques establish a new Pareto frontier for cloud routing.

CCS Concepts: • **Networks** → **Network measurement**; **Network reliability**; **Network management**.

Additional Key Words and Phrases: Anycast, unicast, BGP, routing, performance, availability, cloud.

ACM Reference Format:
Jiangchen Zhu, Tom Koch, Ilgar Mammadov, Shuyue Yu, Kevin Vermeulen, Matt Calder, Italo Cunha, and Ethan Katz-Bassett. 2025. The New (Pareto) Frontier of Cloud Routing: High Availability, Precise Control, or Configuration Stability — Choose Two. *Proc. ACM Netw.* 3, CoNEXT4, Article 28 (December 2025), 25 pages. https://doi.org/10.1145/3768975

1 Introduction

A cloud (or CDN) serves its clients from many geographically distributed sites. To achieve low latency and balance load, it needs to *control* which site each client connects to. However, control is not the only goal. A site can fail due to upgrades, hardware failures (*e.g.,* router, line-card), misconfiguration (*e.g.,* BGP, DNS), facility outages [52], or sudden bursts of traffic (*e.g.,* DDoS).

Authors' Contact Information: Jiangchen Zhu, jz3268@columbia.edu, Columbia University, New York, United States; Tom Koch, tak2154@columbia.edu, Columbia University, New York, United States; Ilgar Mammadov, im2703@columbia.edu, Columbia University, New York, United States; Shuyue Yu, sy3011@columbia.edu, Columbia University, New York, United States; Kevin Vermeulen, kevin.vermeulen@polytechnique.edu, LIX, CNRS, Ecole Polytechnique, Palaiseau, France; Matt Calder, crawlder@gmail.com, Meta, Seattle, United States; Italo Cunha, cunha@dcc.ufmg.br, Universidade Federal de Minas Gerais, Belo Horizonte, Brazil; Ethan Katz-Bassett, ebk2141@columbia.edu, Columbia University, New York, United States.

ACM 2834-5509/2025/12-ART28
https://doi.org/10.1145/3768975

When a failure occurs, clouds want to maintain *availability* by rerouting clients quickly to sites with free capacity and want to maintain configuration *stability* by limiting reconfigurations at other sites. Reconfiguration can induce new outages, even during routine changes [11, 32, 33]. As a result, clouds invest heavily in systems that contain the blast radius of a failure [19, 38]. Finally, clouds support diverse third-party applications and so need to use universal techniques backwards-compatible with today's Internet, without modifying clients or other networks.

Unfortunately, simultaneously achieving these control, availability, and stability goals with universal techniques remains elusive. Two popular techniques used to direct clients to latency-sensitive cloud services are (i) anycast and (ii) unicast with DNS redirection. Anycast lacks control: many sites announce the same prefix, leaving site selection to BGP, which can inflate latency by 100 ms (§2.3.2). Failover with anycast is fast by withdrawing the prefix from the failed site, but the cloud cannot *control* which other sites failover traffic goes to which could cause failure if the site does not have enough capacity [17]. Unicast with DNS redirection offers precise site control but sacrifices availability—DNS resolution caching prevents clients from receiving new addresses quickly and delays failover, often by 60 seconds or more (§2.3.1). Solutions such as PAINTER [36] and others (§2.3.3) bypass the limitations of BGP and DNS, but such approaches cannot be used for all applications and all clients in all networks, a requirement for modern clouds.

Large clouds are impacted—Google [12, 71], Akamai [15, 58], AWS CloudFront [6], and Meta [57] rely on DNS-based redirection, while Cloudflare [20] and Microsoft [13, 22, 35] mostly use anycast.

Towards overcoming the limitations of current cloud routing techniques that severely compromise control or availability following failure, we make the following contributions:

First, we analyze possible techniques to reroute clients following a site failure and argue that *any* universal routing scheme has to compromise on at least one of control, availability, or stability (§3).

Second, we develop universal techniques that combine unicast with DNS redirection and anycast to address current techniques' limitations in control, availability and stability, achieving better tradeoffs than existing ones. We tailor BGP announcements to specific ASes from specific sites in ways that guide BGP path selection in other networks towards intended routes (§4). Based on our experience at large cloud providers, our techniques are applicable to any cloud and can be integrated into existing automated frameworks used by clouds. We term our best techniques ProactiveSuperprefix, SelectiveReactiveAnycast and SelectiveProactiveDeprefer.

Third, we develop load control that can be integrated into our techniques to prevent a site failure from overloading other sites, by carefully crafting BGP announcements to consider failover traffic volume, routing preferences of eyeball networks, and site capacities (§5). While load control is not part of the aforementioned tradeoffs, existing techniques like anycast also do not achieve this.

We deploy our techniques on 28 sites of Vultr [62], a cloud provider with thousands of peers worldwide. We evaluate control of client networks worldwide and emulate site failures. We show that SelectiveReactiveAnycast matches unicast in control and anycast in failover speed, while limiting the failure blast radius to a single backup site in most cases (§6.3). SelectiveProactiveDeprefer nearly matches the control of unicast and the failover speed of anycast without reconfiguring BGP following a failure, ensuring stability (§6.2, §6.3). While anycast failover overloads sites after >90% of failures (by >10% of the site's capacity after 85% of failures and over 300% in the worst case), our techniques keep overload below 1% of site capacity at all sites in 75% of failures (and below 15% in 90% of failures) (§6.4). Our techniques thus extend the Pareto Frontier of cloud routing.

2 Clouds Need Better Routing

2.1 Setting

We use the term "cloud" to refer to both clouds and content delivery networks (CDNs), since from the perspective of Internet routing they have similar footprints, share high-level goals (hosting tenant

applications/content, low latency, high reliability), and (generally, even if not for all applications) need to meet these goals for clients worldwide.

Clouds have *sites* that both contain sets of servers for their services and connect to the rest of the Internet. To connect to the Internet, clouds establish BGP sessions with peers/providers and announce one or more IP prefixes, enabling clients to reach the servers.

Client traffic is routed to the cloud using DNS and BGP. DNS resolves a hostname to an IP address in one of the prefixes, which may be cached by the client, its local resolver, or its router. Client traffic is steered by BGP to a site announcing the longest prefix that contains the address.

2.2 Key Cloud Routing Goals

We focus on key routing objectives that all clouds strive for. Although there may be additional objectives that certain clouds/applications may care about, publications from clouds have repeatedly stressed the importance of serving clients from desired sites (*e.g.,* low latency and not overloaded) [13, 15, 35, 37, 47, 65, 68] and maintaining high availability [22, 31, 36, 41, 43, 45, 66]. From these objectives, we derive the following three goals that are important to clouds. We define metrics that quantify these goals in Section 3.1.

Control. As network conditions, site health, client loads, and performance objectives change, the site a cloud wants to serve a client from may change over time [15]. To meet various performance objectives under dynamic conditions, a cloud ideally should be able to *route a client to any site it wants*. When a site fails, the cloud wants to fail the site's clients over to other sites with sufficient capacity to avoid overload, but other control objectives (*e.g.,* steering failover traffic to the lowest latency site) are of secondary importance during failover.

Availability. If a site fails, the cloud wants to *reroute affected clients to healthy sites quickly*.

Stability. A recent paper from Google noted that BGP has been "a source of historical outages at Google" and that, unlike other changes, there is no way to limit the blast radius of a BGP change, because by design BGP updates propagate globally and quickly into other domains [43]. Cloud BGP configuration is normally changed in a planned rollout and requires verification [8–10, 21, 60]. Making changes during failures either requires this slow rollout (delaying recovery) or risks causing more problems [11, 32, 33]. So, after failure, clouds should *limit the updating of BGP configurations at healthy sites* to avoid the risk of cascading failure. This caution does not apply to the same degree to changing DNS records (*i.e.,* changing client to site mappings) since DNS updates are part of normal operation and can be rolled out incrementally, query by query, and localized to specific domains with global consistency. Hence, we do not include updating DNS mappings in our definition of routing stability.

2.3 Current Techniques are Limited

Two widely used techniques for cloud routing are unicast with DNS redirection and anycast. In unicast with DNS redirection, each site announces a unique prefix, and a service's authoritative DNS resolvers answer queries to direct clients to particular sites. Clouds using anycast advertise a single prefix from all sites (or all sites in a region [69] or in an anycast ring [35, 68]) and rely on BGP to decide which clients route to which sites. Collectively, we have worked on Internet routing at two of the three biggest cloud providers and two of the three largest first-party content providers, as well as talked to engineers at other top clouds and CDNs, and it is our understanding that this represents the current state-of-practice.

2.3.1 Unicast with DNS redirection slows failover. Since unicast can direct clients to specific sites, unicast provides clouds with precise site control, but this control comes at cost. When a site fails, unicast clouds use DNS to direct clients to a different prefix (hence site). However, DNS results are

cached by clients' recursive resolvers, OSes, and applications, which can delay clients moving to a healthy site by minutes (§6.3, Table 1, Appendix A.2), significantly hurting availability.

DNS records include time-to-live (TTL) values that specify how long to cache the record. Small TTL values slow applications (by increasing cache misses) and still cannot guarantee up-to-date DNS records. Some DNS resolvers ignore TTL and use expired records to reduce network traffic [23, 51]. Also, many client applications rely on functions such as `getaddrinfo()` that only return IP addresses, making applications oblivious to TTL. Recent work shows that 20%–80% of the traffic destined to the cloud is sent at least 5 minutes after TTL expiration [36], and other recent work showed that 7% of all flows continued three minutes after DNS record expiration on a large residential network [67]. In addition, we test the behavior of the top 10 popular web services on MacOS and Windows when the service IP address is intentionally made unreachable by us. For some of the applications, their authoritative DNS resolvers return a single IP address (which we then block), whereas, for others, they return multiple (and we block only the first one connected to). We find that on average it takes 60 s for the applications to use a new address from DNS, but this value varies across different applications and systems, ranging from 0.1 s to 130 s (Appendix A.2).

2.3.2 Anycast compromises control and causes overload. Anycast achieves failover within seconds by routing to alternative sites without needing a new IP address from DNS ([7], §6.3), as a failed site can withdraw its announcements. However, anycast compromises *control.* Anycast relies on the BGP policy routing decisions made by other networks to route clients to a site, so the cloud cannot guarantee that clients are directed to appropriate sites (*e.g.,* based on latency or load). Anycast routes a substantial subset of clients to distant sites, inflating latency [13, 35, 42]. For the cloud we measure (and later use for evaluating our techniques), anycast inflates latency for half the clients, with 10% of clients experiencing 100ms extra latency (Appendix A.1). Compared to prior work that measured a different cloud [35], our fraction of clients with latency inflation is similar. After failure, failover traffic can overload other sites, affecting all clients served from these sites. Anycast overloads sites following more than 90% of site failure scenarios we tested (§6.4.4). In prior work, Microsoft also noted that this is an important but hard problem to solve [22, 45].

2.3.3 Clouds need approaches that (only) rely on DNS and BGP. Other systems sidestep some of the challenges of DNS and BGP in particular settings. However, such solutions require additional functionality from the application, the client's network, or the client, and thus are not backwards compatible. PAINTER improves performance over anycast and unicast but requires custom network functions in client networks to steer traffic. PAINTER notes that this requirement makes it less universal/deployable than using DNS [36]. Application-based redirection embeds the redirection in application layer protocols or the application itself (*e.g.,* a custom URL embedded in a Netflix page to direct the client to a particular video cache). This approach is not universal—for example, a cloud cannot apply this solution to hosted tenant applications or to landing pages of its own sites. These approaches are valuable, but any cloud provider that hosts legacy applications (such as websites!), enterprise tenants, or third-party tenant applications—that is, *all* major public cloud providers and CDNs—will also need to employ a universal approach that relies solely on BGP and DNS, and so we will limit our scope to such *universal* techniques.

3 Unavoidable Tradeoffs in Universal Cloud Routing

This section defines metrics to measure control, availability, and stability, and discusses how different choices in cloud routing lead to different tradeoffs. Quantifying these goals helps us compare different techniques. Other reasonable definitions of how exactly to quantify the goals would lead to numerically different results but the same fundamental tradeoffs and qualitative conclusions. Our

discussion of tradeoffs provides guidelines we use to design better routing techniques in Section 4 and applies to *any* universal technique for cloud routing, meaning one that relies on DNS and BGP.

3.1 Metrics for Quantifying Goals

Definition 1. *Control Metric*: Ideally, we would define this metric as the minimum fraction of clients correctly routed to a desired site over all possible client-to-site mappings, which would result in anycast having control of 0%. Evaluating all mappings is infeasible, however, as there are exponentially many to evaluate. Instead, we evaluate it relative to a single mapping, one that maps each client network to a site to which it has low latency. For the desired mapping, we first identify the set of clients that anycast maps to an incorrect site (and exclude those it maps correctly). We then define the control of a cloud routing approach as the percentage of clients with anycast errors that the approach maps to the desired site. Hence, we define anycast as having 0% control, which is reasonable since clouds using anycast rely exclusively on decisions made in other networks to route clients to their sites and so have no control. Both this work (Appendix A.1) and prior work [13, 35, 42] show that anycast suboptimally routes a substantial fraction of clients, and so our metric definition focuses on the clients that are difficult to route correctly. All techniques we consider can correctly route any clients that anycast routes correctly, and so excluding these clients and using anycast to "normalize" our metric does not obfuscate any tradeoffs among our techniques.

Definition 2. *Failover Load Control Metric*: For a cloud using routing technique T (*e.g.,* anycast) for sites S^T, we measure the failover load control of T by emulating the failure of sites one at a time and measuring the overload at the remaining sites in the immediate aftermath, when clients are using cached IP addresses from the failed site. For the failure of a site $f \in S^T$, we define the overloading rate of a site $s \in S^T \setminus \{f\}$ as $\text{overload}\left(s,f\right) = \max\left(0, \frac{\text{load}(s,f)}{\text{capacity}(s)} - 1\right)$, where $\text{load}\left(s,f\right)$ is the total load on s after f fails, and $\text{capacity}\left(s\right)$ is the total capacity of s. With this definition, a site at or below its capacity has an overload of 0, and a site with twice as much traffic as it can handle has an overload of 1. The aggregate overload when f fails is then $\sum_{s \in S^T \setminus \{f\}} \text{overload}\left(s,f\right)$, the aggregate overload across all single site failures is $\sum_{f \in S^T} \sum_{s \in S^T \setminus \{f\}} \text{overload}\left(s,f\right)$, and the aggregate overload across all single site failures across a set of different clouds $\mathcal{S}^T$ is $\text{overload}_{\mathcal{S}^T} = \sum_{S^T \in \mathcal{S}^T} \sum_{f \in S^T} \sum_{s \in S^T \setminus \{f\}} \text{overload}\left(s,f\right)$. Since anycast does not actively control load distribution at all, we define a technique T's failover load control as its normalized improvement over anycast: $\text{failoverControl}_{\mathcal{S}^T} = 100\% \times \left(1 - \frac{\text{overload}_{\mathcal{S}^T}}{\text{overload}_{\mathcal{S}^{\text{anycast}}}}\right)$, so anycast will score 0%, and a technique with no overload at any sites following every single-site failure will score 100%.

Definition 3. *Availability Metric*: Following a site failure, for clients that were using that site, the percentage of time those clients can be routed to a healthy cloud site over a period of x seconds after failure even if the application/OS is still using a cached IP address of the failed site.

The specific value of x will not impact the fundamental tradeoffs that we are about to discuss. However, it makes sense to choose a value of x based on the failover times of approaches under consideration, such as DNS caching TTL values and BGP route convergence time. We will use $x = 150$ since DNS redirection typically achieves failover within this time (§2.3.1). For example, if clients first connect to cloud sites 10 s after the failure *and* remain connected after, then over the period of $x = 150$ seconds, the availability is about $\frac{150-10}{150} \times 100\% = 93\%$. We do not just use the time of the first connection because clients can experience intermittent connectivity during route convergence. To measure this metric, the cloud can emulate a site failure by withdrawing prefixes and then test client reachability by issuing client-side measurements [14]. Section 6.3.1 details how we measure it in our evaluation.

Definition 4. *Stability Metric*: After a site failure, the percentage of sites (ignoring the failed one) that maintain the same BGP announcements as before. For example, if a cloud has 10 sites, and on average a failure of a site requires changing BGP announcements at 3 sites, then it has $\frac{10-3}{10} \times 100\% = 70\%$ stability.

3.2 Tradeoffs are Unavoidable

3.2.1 Failover mechanisms. To understand why at least one of the cloud routing goals must be compromised, we consider all possible approaches to redirect clients from a failed site to a healthy one. Let us suppose some clients are using address a to connect to a site s that announces a prefix p that includes a. When site s fails (and withdraws p), at least one of the following must happen so that the affected clients can connect to a healthy site and achieve failover:

(1) *DNS-failover.* A client queries the DNS, learns a different address a' that is within a prefix p' announced by some healthy site, and starts using it.
(2) Clients continue to use the address a (or another address within p). For the address to be reachable, a prefix that includes a must be announced from at least one healthy site. There are three possible announcements.
  (a) *same-prefix-failover.* A (healthy) site (not s) announces p.
  (b) *super-prefix-failover.* A site announces a superprefix p_{super} that includes p (and a).
  (c) *sub-prefix-failover.* A site announces a subprefix of p, p_{sub}, that includes a.

Since BGP routing works at a per-prefix basis, we categorize the announcements based on their prefixes' relationships to p. However, within each category, there exist many ways of making the BGP announcement (*e.g.,* BGP attributes, when and where to make the announcements), which we will discuss in Section 4.

3.2.2 Analyzing tradeoffs. We analyze the primary concern or tradeoff in each failover mechanism.

DNS-failover. As discussed in Section 2.3.1, it can take tens of seconds to minutes for applications to start using a new address after failure, resulting in low *availability* (§6.3, Table 1).

super-prefix-failover. Routers perform longest prefix matching to route packets. Even if a router has routes to p_{super}, they will not be used while the router has any route to the prefix p in its BGP table. When s fails and withdraws p, BGP will choose and exchange increasingly less preferred outdated routes toward the failed site, blackholing traffic, until all alternatives are exhausted and the withdrawal of prefix p has finally converged. But complete withdrawal of a prefix is known to converge slowly, taking a median of approximately 100 s [40] (also measured and validated in our preliminary work [70] and in Section 6.3). This leads to low availability (38%, §6.3, Table 1) in the 150 s window following the failure.

same/sub-prefix-failover. same-prefix-failover does not suffer from the same availability issues as *super-prefix-failover.* Alternate routes to p announced by healthy sites will compete with lingering outdated routes withdrawn by the failed site after failure. Instead of waiting for all routes towards the failed site to completely disappear as in *super-prefix-failover* (which takes much longer), routers in *same-prefix-failover* will short-circuit convergence by switching to a competing valid route announced by a healthy site. A prior study found that a CDN's anycast failover (which is *same-prefix-failover*) completes within 1 second for 90% of the vantage points [58], resulting in $\approx 100\%$ availability. We also validate this in Section 6.3. *sub-prefix-failover* achieves fast failover for a similar reason: routers will prefer p_{sub} over p, so as soon as they learn routes to p_{sub}, failover is achieved.

However, *same/sub-prefix-failover* will compromise either control or stability, depending on when the backup announcements of p or p_{sub} are made. These announcements from other sites can

either be made before the failure along with the announcements from site s (*i.e.,* proactively), or only after the failure of site s (*i.e.,* reactively).

If p or p_{sub} is announced by other sites before the failure, then *control* will be compromised. If other sites announce a more specific prefix p_{sub}, then longest-prefix matching will route all clients towards them, compromising control completely (in practice, no cloud would do this). When other sites announce the same prefix p, some routers will select routes towards the other sites, even if the sites use mechanisms such as AS path prepending, since routers are free to pick any route to a prefix in their BGP decisions. We validate in our preliminary work [70] and Section 6.2 that in practice 42% of clients are misrouted toward sites adding AS-path prepending even when non-prepended paths are available, and that increasing the prepending does not significantly improve control.

If p or p_{sub} is announced by other sites only after the failure, then their BGP configuration must be changed during an ongoing failure, compromising *stability*.

Unavoidable tradeoffs. Hence, some tradeoff has to be made in implementing a cloud routing technique. *DNS-failover* and *super-prefix-failover* can ensure full control and stability but have to compromise availability significantly, as with unicast with DNS redirection (§2.3.1). *same/sub-prefix-failover* achieve high availability but have to compromise some control or stability (*e.g.,* anycast compromises control (§2.3.2)). Our guidelines apply not just to our techniques but to all *universal* routing techniques and determine the central tradeoff for any possible cloud failover technique. Based on these guidelines, we introduce techniques that make announcements *selectively* to minimize these compromises (and achieve all goals almost fully).

4 Pushing The Limits of The Tradeoffs

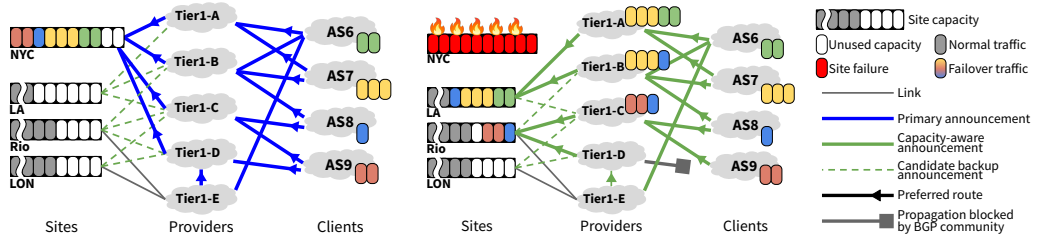
This section introduces new *universal* techniques that achieve *control*, *availability*, and *stability* close to optimal. Whereas existing systems choose unicast or anycast (with many clouds/CDNs depending on both across different systems), we take a hybrid approach. In normal operation, to achieve control, we assign each site a prefix and use DNS redirection to point clients to sites. After a site failure, we use announcements of the site's prefix from other sites as backup routes to achieve failover without being delayed by DNS caching, similar to anycast. One of our variants announces backup routes only after failure—*reactive* backup—while the other announces backup routes before failure, but in ways that cause most/all networks to choose the primary routes—*proactive* backup.

4.1 Preliminary Techniques and Their Limitations

We describe initial, simple ways of combining anycast and unicast that benefit from both strengths but also have significant drawbacks to overcome. We originally proposed these techniques in a short, preliminary version of this work [70], before we developed ways to overcome the drawbacks.

Reactive backup. Our preliminary work introduced ReactiveAnycast [70], which announces unicast prefixes from each site, and, when a site fails, its prefix is reactively announced from every other site. The technique preserves control in normal operation and availability after failure, but requires reconfiguring every site after a failure, compromising stability. Moreover, the anycast announcements overload healthy sites in 90% of failure scenarios (§6.4).

Proactive backup. Our preliminary work also proposed ProactiveSuperprefix, which announces covering (i.e., less-specific) prefixes at backup sites to avoid reconfiguration after failure, but, due to longest-prefix matching, failover is slow [70], as we validate in Section 6.3.3. Our preliminary work also introduced ProactivePrepending—backup sites proactively announce the *same* prefix as the primary site but with AS path prepending. For consistency with our naming conventions of new techniques we will introduce in this paper, we will refer to that technique as ProactiveDeprefer(prepend) rather than ProactivePrepending. We found that, of clients



(a) Announcement configuration for NYC during normal operation. Green (dashed) links are candidates to make backup announcements. Backup routes are only announced to providers shared with NYC (Tier1s A–D, but not Tier1 E) to maintain availability. Proactive backup announcements use AS path prepending or tag communities to lower `localpref` at the provider to achieve control (§4.2).

(b) Capacity-aware announcements during failover (§5) in solid green lines are a select subset of candidate announcements. The buckets demonstrate the bin packing problem where client loads (items) are routed (packed) into sites with capacities (bins) (§5.2). AS 6 consistently prefers Tier1 A (§5.2.1). The other client ASes have mixed preference among their providers (§5.2.1). AS 9 is assigned to Tier1 C by the BGP community. For AS 7, we group its providers (Tier1s A and B) at LA. For AS 8, we reserve duplicate headroom at LA and Rio where its traffic may arrive. LON makes no backup announcement to improve stability.

Fig. 1. Primary announcement configuration and candidate backup announcements before the failure (a), and capacity-aware backup announcements during failover (b). Most but not all providers are Tier1 networks. We label them as such to make it easy for the reader to tell when we mean a provider versus a client network.

not directed to a nearby site by anycast, an average of ≈46% per site are still misdirected by ProactiveDeprefer(prepend), compromising control (§6.2.2). Adding more prepending does not improve control significantly. In addition, following a site failure, while routes are reconverging, clients flip between multiple sites for ≈50 s (§6.3.3), which can break connections [64].

To summarize, the preliminary reactive and proactive techniques have significant limitations. The challenge in both is configuring announcements at backup sites to avoid significant compromises on any of the three goals, even though at least some compromise on at least one goal is inevitable (§3.2). Finding configurations with good tradeoffs involves selecting the right sites, providers, and BGP attributes for each backup announcement.

4.2 SelectiveProactiveDeprefer: Better Proactive Backup

Our preliminary technique ProactiveDeprefer(prepend) has much lower control than unicast in normal operation and has slower route convergence than anycast after failure. In this section, we will analyze the causes behind each problem and propose solutions through a combination of selective announcements and certain BGP communities.

Improving control by only announcing backup routes to providers of the primary site. We need to encourage all ASes to prefer the routes from the primary site. In the BGP decision process, AS path length is only considered when routes are tied according to a network's `localpref`, so, in ProactiveDeprefer(prepend), a longer prepended route may be selected when a network sets it a higher `localpref`. In Figure 1a, Tier1 E is a provider of backup sites Rio and LON but not the primary site (NYC). Tier1 E will prefer (customer) routes to Rio and LON over (peer) routes to NYC learned from Tier1 D, regardless of path length.

To avoid this outcome, we only use a site as backup if it shares provider(s) and/or peer(s) with the primary, and we announce at the backup sites *only* via those shared neighbors. In Figure 1a, to backup NYC, we only consider backup announcements to Tier1s A–D (NYC's providers), from sites that connect to those providers (dashed green lines). As networks usually assign the same `localpref` to all routes learned from the same AS [25], Tier1s A–D are likely to choose the primary route received from the cloud over a prepended backup route. Since they are the only networks receiving the backup routes and they prefer (and export) the primary route, no other networks (*e.g.*, Tier1 E) will learn or prefer the backup routes before failure. In particular, Tier1 E only learns routes through Tier1 D, and, since Tier1 D prefers the route from NYC, Tier1 E will be correctly routed to NYC. Section 6.2 validates this claim experimentally. If the primary site fails and withdraws

its route, the neighbors (Tier1s A–D) immediately switch to and propagate the prepended routes they are already learning from backup sites, enabling fast failover to restore availability. Clouds typically connect to a number of large providers and peers at multiple sites, and these neighbors account for a large volume of client traffic [59]. This ensures the existence of shared providers.

Speeding route convergence by using BGP communities to change `localpref`. When using BGP prepending to make backup routes less preferred, routes can converge slowly following a failure because ASes will attempt to select the outdated shorter routes towards the failed site before converging to the new longer backup routes. In Figure 1a, client AS 6 selects a route through Tier1 A to reach the primary site before failure. When the primary site withdraws its routes, Tier1 A updates its route to a prepended backup route (towards LA) and exports to AS 6. Meanwhile, Tier1 E still announces its stale, shorter route, so AS 6 prefers it. This route may fail to reach NYC or reach Rio, depending on whether Tier1 D has updated its route. Eventually, Tier1 D and Tier1 E will converge toward Rio and propagate this long prepended route to AS 6, which will switch back to the route through Tier1 A. In general, client ASes may bounce between sites during convergence.

Our idea is to announce backup routes that are of the same length as the primary route (*i.e.,* without prepending), so that the shared providers (Tier1s A–D) will fail over seamlessly and propagate the same backup routes as they do before failure. However, the risk is that the shared providers (Tier1s A–D) may pick the backup routes before failure, unless we can induce them to set lower `localpref` on the backup routes.

Although the `localpref` is set by the neighboring network importing the route and is not directly modifiable by the cloud coriginating a prefix, many large providers (including all of the Tier1 networks connected to our testbed (§6)) define BGP communities that customers (such as a cloud) can use to instruct the provider how to set `localpref` [18, 29, 48–50, 53, 61]. For example, NTT defines the community 2914:480 such that routes tagged with that community will be given a `localpref` equal to NTT's default customer value minus 10.

Therefore, backup sites can tag announcements with such communities, lowering a shared provider's `localpref` below its default customer value but above or equal to its default peer value. Since by default (without communities) routes from the cloud will be considered customer routes, routes from the primary site will be preferred by the shared providers (Tier1s A–D in Figure 1a). After failure, the route from the primary site is withdrawn, and the shared profiders will seamlessly switch to backup routes, which will be preferred over any peer routes (due to higher `localpref` or shorter AS path). Because primary and backup AS paths are identical, providers export the same routes after failure, avoiding route reselection in other networks (e.g., AS 6 in Figure 1a).

To sum up, our new technique SelectiveProactiveDeprefer announces routes at backup sites only to providers of the primary site, tagging them with appropriate communities. If a shared provider does not support such communities, the backup sites instead use AS path prepending to it. In practice, since large clouds connect to Tier1 networks that support `localpref` communities, SelectiveProactiveDeprefer can often achieve control through BGP communities. SelectiveProactiveDeprefer achieves 99% control (§6.2) and achieves availability similar to anycast (§6.3).

Multi-Exit Discriminators (MEDs) also indicate to a provider connected at multiple sites which ones to (de)prefer. We do not evaluate MEDs since the cloud we use for evaluation (§6) does not provide a way for us to control the MEDs announced to its providers. We expect that using MEDs would perform equivalently to using BGP communities that change providers' `localpref`, which we evaluate. In practice, a cloud can use `localpref` communities, MEDs, or prepending on backup announcements to a provider, depending on which ones are supported by that provider.

4.3 SelectiveReactiveAnycast: Better Reactive Backup

Our preliminary ReactiveAnycast technique requires reconfiguring all backup sites after failure, compromising stability. To mitigate this problem, we propose to use a smaller number of backup sites—we term this technique SelectiveReactiveAnycast. One might worry that announcing from fewer backup sites could lead to slower failover than anycast, if the reduced number of routes available for failover led to slower propagation. We find that, despite using fewer backup sites, SelectiveReactiveAnycast does not sacrifice the availability of anycast, if we select backup site(s) that connect to provider(s) or peer(s) of the failed site, because these providers quickly receive a backup route and avoid selecting failed routes during the convergence of the withdrawal (§6.3).

5 Load Control After Failures

While we propose using a small number of backup sites in SelectiveReactiveAnycast to improve stability, this desire to use as few backup sites as possible exposes another challenge—handling the failover load without overloading backup sites. If a backup site receives more failover traffic than it has free capacity, the overload impacts all its clients, not just the ones that came from the failed site. Using one backup site will not suffice if the failover traffic volume exceeds the backup's unused capacity, which will often happen (headroom is commonly $\leq 50\%$). Using multiple backup sites may not help either, regardless of aggregate spare capacity, as BGP may route clients to any site, and so the failover traffic load does not necessarily spread out evenly across backup sites.

Avoiding overload during failover is not just a challenge for SelectiveReactiveAnycast or even just for approaches that limit the number of backup sites, and in fact applies to all techniques that announce a failed site's addresses from other sites (proactively or reactively), including anycast. In fact, when we emulate site failures on the Internet using assumptions about site load and available capacity (headroom) informed by discussions with cloud and CDN operators, anycast overloads sites after 90% of failures, with sites receiving up to $3\times$ more traffic than they have total (not just available) capacity (§6.4.4)! In prior work, Microsoft also noted that this is an important but hard problem to solve [22, 45]. The root cause is that, as with anycast during normal operation, BGP does not allow operators to easily control (or predict) which sites clients will be rerouted to if multiple sites announce the same prefix.

5.1 Overview

To prevent overload, we introduce *capacity-aware* backup announcements that account for each site's available capacity. Since the backup routes are only needed until clients refresh DNS (which then restores cloud control), the priority is to quickly restore connectivity without overload, even if it means temporarily using distant sites with higher latency. Our capacity-aware announcements are compatible with all of our techniques.

We want to reroute the client networks from a failed site to as few backup sites as possible (*e.g.,* to boost stability of SelectiveReactiveAnycast) while not overloading any site. Cloud sites are overprovisioned even at peak time, to accommodate sudden traffic spikes and failures. Clouds monitor traffic volume from client networks and available capacity at each site in near real-time. Although load changes over time, a cloud can observe the load pattern per site to derive a capacity-aware configuration that will work at peak time (and also other times) or different capacity-aware configurations for different times of day. Our techniques do not specify the metrics for load. A cloud can use metrics such as requests or bytes per second.

5.2 Our Approach to Failover Load Control

We model this problem as a bin-packing problem, where the cloud can only manipulate which backup site (the bin) a client is routed to (packed in) indirectly through BGP announcements to the cloud's direct providers. A key challenge is that routing in client, provider, and other intermediate ASes is unpredictable [45], which leads to uncertainty about which backup site a client will route to (and thus whether a site will be overloaded). This section discusses our approach to the problem, and Appendix C formally articulates it as an optimization problem.

To overcome this uncertainty, we manipulate which routes are available so that we gain certainty about which provider a client will use and which site traffic through that provider will end up at. We can restrict which site each of the clouds' providers route traffic to by announcing to each provider only at the one site. Below we describe ways of further restricting the providers that clients reach the cloud through.

Figure 1b illustrates a capacity-aware configuration after site failure. After NYC fails, the cloud wants to redirect client ASes 6–9 to backup sites. Failover without overload is difficult since (i) the total traffic from clients (colored boxes) exceeds the unused capacity of any single backup site, so the cloud cannot just route all clients to the largest site, and (ii) predicting or controlling where clients ingress is hard [45]. We incorporate three key observations to overcome these challenges.

5.2.1 Making client to site mappings more predictable. First, most clients consistently prefer to reach the cloud through a single provider, regardless of which sites are announcing to that provider and which providers the cloud announces to [68]. Prior work demonstrates how to issue measurements to identify and continuously maintain these mappings of clients to providers, as well as how to identify the (small fraction of) clients do not have a single preferred provider [68]. In Figure 1b, AS6 consistently prefers Tier1 A over Tier1s B and E, so we assign AS6's load to Tier1 A.

Second, a cloud can sometimes restrict a client's route towards the cloud using BGP communities offered by many large ASes. In Figure 1b, Tier1 D has a BGP community for each customer AS to tell it not to export a route to that customer. The cloud can use this community to force traffic from client AS9 through Tier1 C (and so AS9's load is assigned to Tier1 C).

Third, the cloud may be unable to restrict a client's route to a single provider (ASes 7 and 8). In these cases, we assign the client's traffic to *multiple* providers. This assignment leads to two cases. (1) *Conservative headroom reservation.* A client may have mixed provider preferences, but as long as we retain enough headroom at all sites where the client's traffic can arrive, overload will not occur. As an example, client AS 8 has mixed preference between Tier1s B and C. If, as in the figure, the bin packing assigns Tier1s B and C to two different sites (LA and Rio), AS 8's traffic can potentially contribute to the load at either provider and hence either site. In the bin packing bins, we reserve room for AS 8's load twice (once per site) to account for the uncertainty. Incorporating uncertainty in this way translates to a conservative lower bound on a site's available capacity. Despite this conservative constraint, we are able to find solutions that accommodate all failover traffic in all scenarios we evaluate (§6.4). (2) *Grouping providers.* If the bin packing is able to assign all of a client network's possible providers to the same site, we can ensure the site's received load is predictable, regardless of which provider the client takes. Client AS 7 may route through either Tier1 A or Tier1 B, so the exact load from Tier1s A and B is uncertain. But both Tier1s A and B are assigned to LA (*i.e.,* only LA announces to Tier1s A and B). So, in the bin packing problem, we only need to reserve room for AS 7's traffic once at LA.

5.2.2 (Bin)Packing clients to providers to sites. Using these techniques, we assign client load to providers and run a bin packing solver that assigns providers to sites (which translates to BGP announcements from sites to providers). The decision of how/whether to (1) group providers or

(2) conservatively reserve capacity for a client AS at multiple possible sites trades off requiring more failover capacity at a single site versus requiring more global failover capacity. Grouping providers requires additional capacity at a single site to absorb failover traffic from all clients that use any of the providers in the group, which may be impossible if sites lack enough headroom. Conservatively reserving headroom at multiple sites requires additional global capacity for any clients that *might* use a site's assigned providers. Since in practice the number of providers and size of possible provider groups is relatively small (*e.g.,* if a site connects to ≤ 20 providers, which is typical for a cloud [1, 26], there will be ≤ 627 possible groups) and the problem at a cloud scale is typically solved in 20 minutes (Appendix C), we decide to run the bin packing for each possible grouping and choose the solution with the least number of backup sites used.

Occasionally, one provider may carry too much failover traffic for any healthy site to handle without overload. In this case, the cloud needs to assign the large site multiple prefixes (a priori, before failure) and split traffic across the prefixes ahead of failure by returning different DNS answers to clients. In this way, that provider's traffic to each prefix is smaller and can fail over to different sites. Since clouds need to limit prefix use [36], we use the smallest number of prefixes that enables a provider-to-site assignment (bin packing) for the site.

In Figure 1b, we reason through a solution that a bin packer might arrive at. Tier1 A's maximum possible load (5) can only be handled by LA (spare capacity 6). Since client 7's traffic is handled either by Tier1 A or Tier1 B, we can assign both to LA (grouping Tier1s A and B) without exceeding LA's unused capacity and avoid reserving duplicate headroom for AS7's traffic at different sites. After this assignment, LA is fully occupied by traffic from Tier1 A and Tier1 B. Tier1 C (max load 3) cannot be assigned to LA, so we assign it to Rio (capacity 4). This assignment allows the cloud to fail over by making announcements from just 2 sites. Client AS8's traffic may arrive at LA and Rio, so the bin packer has reserved unused capacity at both LA and Rio for AS8's traffic.

5.3 Limitations and Requirements

Our techniques do not consider the capacity of the links between backup sites and their providers. Future work may extend the optimization (Appendix C) with additional constraints on provider-to-site assignments based on link capacity.

Our techniques require some backup sites to connect to providers of the failed site. Most cloud providers (including the one we use in evaluation) have a high degree of shared providers across sites [1, 26], so we expect that our techniques work as expected on them. With limited or no shared providers/peers, our proactive backup technique may not work well (Appendix B.1), and our approach to load control cannot work.

If site capacities are not sufficiently overprovisioned, overload after failure may be unavoidable. For example, when the total spare capacity at backup sites is smaller than the failover load, or when many sites have insufficient capacity to handle any provider's traffic, there will be no solution to the bin packing problem. If no solution is found, the solver can be rerun with increased (virtual) site capacity to guide future provisioning.

Summary. This section discussed how to make an announcement from multiple sites so that clients are routed based on site capacity, a problem not solved in prior work. We solve it by jointly optimizing client-to-provider and provider-to-site assignments. Details can be found in Appendix C.

6 Evaluation

We evaluate the control, availability, and stability of our techniques using PEERING [56], deployed on 28 sites of Vultr, a public cloud with thousands of peers. Our evaluation shows that our techniques form a new Pareto Frontier for cloud routing over the three key routing goals (Table 1).

6.1 Evaluation Methodology

6.1.1 PEERING-based Prototype. We use PEERING's deployment on Vultr [56] to emulate a cloud and evaluate the control, availability, and stability of each technique. Every site has at least one transit provider (so is globally reachable), and most sites have 3–5 transit providers and hundreds of peers. This deployment provides a global footprint representative of the large cloud and content providers for which we intend our techniques. The 28 sites include 1 in Africa, 1 in South America, 2 in Oceania, 7 in Asia, 7 in Europe, and 10 in North America (Fig. 2). Similar to how 90% of Facebook traffic is to a user within 2500km [57], 80% of the world's Internet population is within 2500km of a Vultr site. The deployment density is between that of IBM and that of Microsoft/Google/Amazon [5]. We use a /23 prefix and the two /24s it contains for the experiments. We emulate a site failure by withdrawing our announcements from the site.

6.1.2 Emulating Cloud Clients. To test availability and control, we need a set of "clients" for each site. We use ISI's IPv4 Hitlist [39] to get a list of 3.6M clients that respond to `ping`. We sample targets differently from this superset for different evaluation purposes (explained later—§6.2.1, §6.3.1, §6.4.1). We issue and record responses for pings from Vultr sites to clients before and after emulated failures. We assess availability based on whether or not we receive the clients' responses after failure. We assess control based on whether clients' responses are routed to expected sites.

6.2 Achieving Greater Control

ProactiveSuperprefix and SelectiveReactiveAnycast always optimize control under normal operation, because the most specific prefix is only announced at one specific site, so the cloud can direct a client to a particular site using DNS. We evaluate the control of our technique SelectiveProactiveDeprefer, in which backup sites announce the prefix of the primary site under normal operation. We find that it routes more than 99% of clients to the intended site. In contrast, our preliminary approach ProactiveDeprefer(prepend) (§4.1) only routes 58% [70] .

6.2.1 Methodology. We measure whether our techniques can steer clients to 12 sites: Amsterdam, Atlanta, Bangalore, Chicago, Johannesburg, Paris, Sao Paulo, Seattle, Singapore, Sydney, Tokyo, and Warsaw (1 in Africa, 1 in South America, 1 in Oceania, 3 in Asia, 3 in Europe, and 3 in North America). They cover large global interconnection points and geographic diversity. Our results show consistency regardless of the site (§6.2.2) so we believe the study on 12 sites is sufficient.

For each site we study, we select client targets that we desire to route to the site as follows:

- **Not routed by anycast.** We exclude targets already routed to the site by anycast (see control metric in §3.1)—ProactiveDeprefer(prepend) and SelectiveProactiveDeprefer can trivially route these targets to the site because routes to other sites are less preferred than under anycast.
- **Proximity and AS diversity.** For each site, we consider targets that are within 50 ms round-trip latency of the site because clouds normally want to redirect clients to nearby sites (80% of hitlist targets are within 50 ms of at least one site). We select 10,000 targets per site spread across ASes as evenly as possible, selecting randomly within an AS if it has multiple eligible targets.
  We measure 10,000 targets rather than probing all of them because measuring availability will require rapid probing for accuracy (§6.3.1). Probing a large number of targets quickly can introduce losses (at PEERING or other networks on the Internet) that could be mistaken for unavailability. Empirically we find that probing 10,000 targets achieves a good balance: it provides good coverage but rarely experiences packet loss once BGP has converged. Figure 2 shows that the sites and targets we use cover the most populous regions in the world. Our targets include 42K ASes out of 65K ASes with responsive targets in ISI's hitlist. They are located in 14K cities in 191 countries.

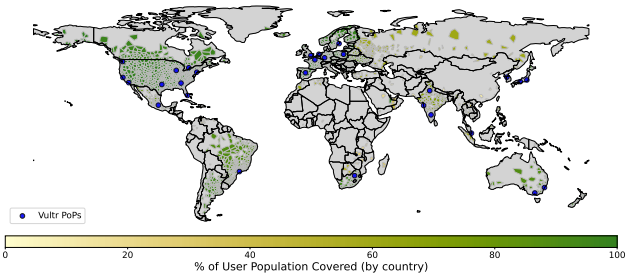


Fig. 2. Locations of Vultr PoPs and regions of population [3] in which we measured at least one ping target. We calculate the percent of each country's population that resides within regions with targets, then use that percent to color the country's regions in which we have targets. Our population coverage is good near PoPs.

Technique	Control[1,2]	Availability[1]	Stability [1]	Failover Load Control[1,2]
anycast	0%	99%	100%	0%
unicast	100%	0% [3]	100%	N/A [3]
ProactiveSuperprefix	100%	38%	100%	95%
ProactiveDeprefer(prepend)	58%	97%	100%	0%
SelectiveProactiveDeprefer	99%	97%	100%	95%
ReactiveAnycast	100%	99%	0%	0%
SelectiveReactiveAnycast	100%	99%	94%	95%

[1] See definitions in Section 3.1.

[2] Improvement over anycast (see Section 3.1).

[3] Availability and Failover Load Control while clients use cached IP addresses, so unicast does not fail over at all.

Table 1. Shaded techniques achieve the best tradeoffs (§3) among control, availability, and stability and are on the Pareto frontier of cloud routing.

We iterate through the 12 sites. For ProactiveDeprefer(prepend), we prepend three times and announce backup routes to all neighbors at the other 27 backup sites (§4.1). For SelectiveProactiveDeprefer, we announce backup routes only to providers that also connect to the failed site (§4.2). We test with two variants using prepending or BGP communities (§4.2). We wait 40 minutes to ensure convergence. We then ping every target and inspect whose response are received by the current primary site we are studying, which means they are *controlled* by the technique.

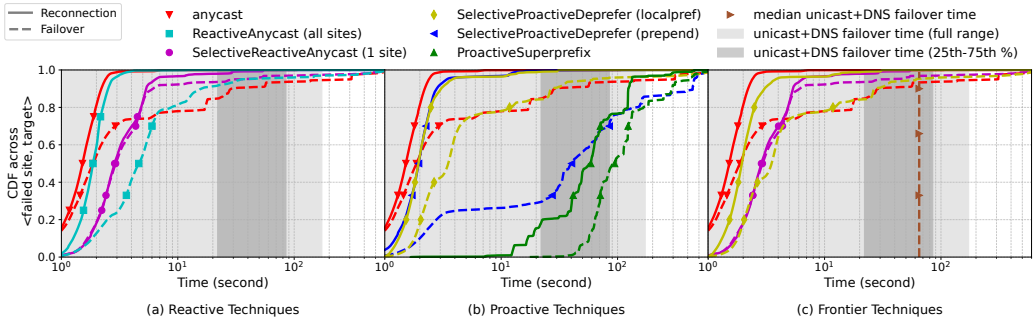
6.2.2 Results. The *Control* column in Table 1 shows ProactiveDeprefer(prepend) only controls ≈58% of targets on average. Its control ranges from 34% to 82% across sites (52% median). In contrast, SelectiveProactiveDeprefer (both with prepending or communities) achieves at least 99% control on every site we measure, validating our claim in Section 4.2.

6.3 Achieving High Availability and Configuration Stability

We now show that SelectiveReactiveAnycast achieves the same availability as ReactiveAnycast despite using a much smaller number of backup sites, providing it with much better stability. Both have availability close to anycast. SelectiveProactiveDeprefer also achieves similar failover speed and hence availability to anycast. For SelectiveProactiveDeprefer, we find that using BGP communities is better than AS path prepending—the use of non-prepended backup routes enables its routes to converge faster following failure (§4.2).

6.3.1 Methodology. We use 28 Vultr sites to emulate a cloud and study the availability after the failure of the same 12 sites as in Section 6.2, one at a time. For each specific site we study, in addition to using 10,000 targets that are not routed to the site by anycast (§6.2), we select another 10,000 targets that are routed to it by anycast because anycast directs roughly half of all clients to optimal sites (§2.3.2).

When studying a technique, we fail each site one at a time and measure failover properties. Before failure, we announce a prefix from the primary site. For this availability evaluation, we do not use capacity-aware backup configurations, which we instead evaluate separately in Section 6.4. For



(a) Reactive Techniques (b) Proactive Techniques (c) Frontier Techniques

Fig. 3. (a) While requiring reconfiguration at only 1 site, SelectiveReactiveAnycast achieves similar availability to ReactiveAnycast with much more stable configuration. (b) SelectiveProactiveDeprefer(localpref) achieves faster convergence (failover) than SelectiveProactiveDeprefer(prepend). ProactiveSuperprefix has much slower reconnection time than the other proactive backup techniques. (c) SelectiveProactiveDeprefer-(localpref) and SelectiveReactiveAnycast achieve similar reconnection and failover time to anycast. Unicast+DNS has a wide range of failover times across applications and OSes, with median slower than others.

ProactiveSuperprefix (§4.1), we announce a covering prefix at all other backup sites. For SelectiveProactiveDeprefer, we test with two variants: (1) prepending or (2) adding "set `localpref`" BGP communities. We announce (depreferred) backup routes from all sites but only to providers of the primary site. With these proactive backup announcements in place, we emulate a failure at the primary site by withdrawing its announcement. After the failure, for SelectiveReactiveAnycast, we select a single backup site that shares the most providers with the failed site and announce backup routes to the shared providers (§4.3). (We conduct experiments using different sets of backup sites and find similar results in Appendix B.1.) We then probe all targets every ≈1 s for ≈600 s. We measure from the current site's prefix, and responses are either routed to some backup sites or lost as a result of BGP convergence.

6.3.2 Reconnection and failover time. In addition to the availability metric in Definition 3, we define two more metrics for each target (client). *Reconnection time* is the delay between the onset of failure until the first response is received at any site. After the first response, a target may continue to experience periods of unreachability and/or be routed to one or multiple other sites before its route converges and settles on a site. Hence we also define *failover time* as the delay between the onset of failure and the first response from the target after which the target routes to a single fixed site. The reconnection time serves as the lower bound to restore the service, and failover time serves as a conservative upper bound for the time to restore connectivity.

6.3.3 Results. Reactive failover. Figure 3(a) shows the reconnection time and failover time for SelectiveReactiveAnycast using only one backup site, ReactiveAnycast using all backup sites, and anycast using all sites. Using one backup site yields nearly the same reconnection time as using all sites—only 1s longer at the median and 2s longer at the tail—and slightly reduces the failover time by 2s at the median (measurements taken at 1 probe per second). The slight increase in reconnection time occurs because the backup route is only propagating from a single source, so takes longer to reach some networks. However, with only one backup route, routing decisions rarely "bounce" once the route is learned, and so reconnection time and failover time are usually the same, leading to the failover improvement. Overall SelectiveReactiveAnycast offers similar availability (99%, Table 1) to ReactiveAnycast, with much smaller blast radius and better stability.

Proactive failover. Figure 3(b) shows the reconnection time and failover time for three proactive failover techniques and anycast, using all backup sites. First, ProactiveSuperprefix has much longer reconnection and failover times than anycast (§4.1), confirming that *super-prefix-failover* has availability issues (§3.2). Second, both SelectiveProactiveDeprefer variants achieve reconnection

time similar to anycast (≈2s at median), showing that limiting announcements to shared BGP neighbors preserves availability and improves control significantly (§6.2). SelectiveProactiveDeprefer(localpref) also improves failover time compared to SelectiveProactiveDeprefer(prepend) and achieves a failover time similar to anycast, as it achieves faster route convergence by offering non-prepended backup routes (like in anycast) (see discussions in Section 4.2). In practice, the availability of SelectiveProactiveDeprefer likely matches SelectiveProactiveDeprefer(localpref) in most cases, since cloud sites often connect to tier-1 networks that support communities. Table 1 shows the availability of each technique. The availability of SelectiveProactiveDeprefer is 97%, but the availability of ProactiveSuperprefix is much lower, at 38%. Appendix B.1 shows that high-availability failover can be provided by any of the top three backup sites that share the most providers and peers with the primary site, enabling flexible backup configuration.

Comparison to existing techniques. Figure 3(c) compares the techniques introduced in this work (SelectiveProactiveDeprefer(localpref) and SelectiveReactiveAnycast) to today's dominant techniques (anycast and unicast with DNS redirection). We estimate the reconnection time for unicast based on applications' DNS behavior during failure, showing the range across applications and OSes (see Appendix A.2). Both techniques achieve similar availability to anycast. SelectiveProactiveDeprefer(localpref) retains full stability and achieves almost the same control of unicast. SelectiveReactiveAnycast retains the full control of unicast at the cost of reconfiguring a small number of sites after failure.

6.4 Balancing Load During Failure

Our evaluation shows that anycast failover overloads sites following 90% of site failures, with 0.4× overload rate at median and 1× overload at 90th percentile, while our bin-packing approach leads to less than 0.01× overload in 75% site failures and ≈0.1× overload at the 90th percentile (§6.4.4).

6.4.1 Assigning load and capacity of sites. To meaningfully evaluate load control after failure, we need to assign reasonable site capacities and network traffic volumes that reflect interconnectivity and routing. Since we lack production traces, we assign client traffic and site capacities based on the intuition that traffic, capacity, and interconnectivity tend to increase with the number of users. We assign each network traffic proportional to its user population (estimated by APNIC [4]). We split the population evenly across all responsive ISI hitlist targets within the network and use all targets in our evaluation to maximize coverage (since the measurements can be run at a slow rate, loss is no longer a concern). We run all failure scenarios twice, using different approaches to assigning site capacity. In one, we assign targets to the site they are routed to by anycast. In the other, we assign them to their lowest latency site. For both approaches, we compute each site's *site load* as the sum of the APNIC user population contributed by its assigned targets.

Our approaches assign higher load to sites with higher connectivity in more populous regions. For example, with the first approach, Tokyo attracts 14% of the global load and Melbourne attracts 1% of the global load. Sites have large differences in their load assignments due to Vultr's small presence in APAC relative to the Internet population there. We apply other ways of assigning traffic volumes to users and capacities to sites in Appendix B.2 and find that our key takeaways still hold.

We assign each site headroom capacity of 40% of its normal load, which is similar to the headroom we were informed of by operators at multiple clouds. For example, Tokyo attracts 14% of global load and thus has headroom for an additional 14% × 40% = 5.6% of global load. Appendix B.2 varies the site's headroom with diurnal patterns and shows that our technique still works well.

6.4.2 Simulating Different Clouds. To test our techniques with varying routing dynamics, we subsample our sites to create 21 additional "sub-clouds", each including all sites except 2. For each sub-cloud, we remove two of the seven largest capacity sites from the set of all sites. Removing any

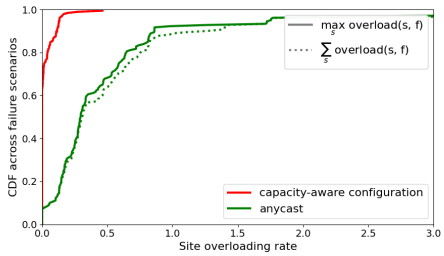


Fig. 4. Anycast causes overload after >90% site failures (often severely), while our capacity-aware configuration only leads to overload after <40% of failures. When overload occurs, capacity-aware configurations achieve a much lower overload (0.01 at P75 and 0.1 at P90).

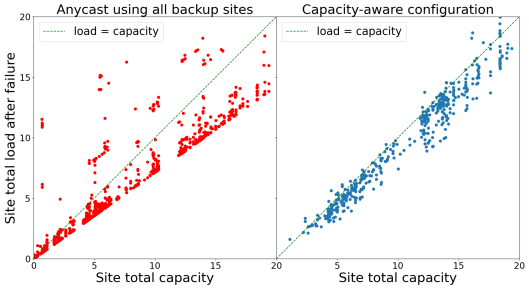


Fig. 5. After a site fails, load on backups using anycast (left) vs. our capacity-aware configurations (right). Capacity and load are shown as % of global load. For most failures, anycast failover overloads some backup sites, while our configurations cause little overload.

of them causes large changes in routing and hence large changes in the capacity we assign to other sites, making the scenarios we evaluate more diverse. We exclude a few scenarios, when (1) the failover load of a single large site exceeds the total spare capacity of all others, or (2) there exists a provider with so much anticipated load that, even if the load is split evenly across 16 prefixes, backup sites do not have enough headroom to partition the load from the prefixes (§5.3).

6.4.3 Simulating Failures. For a sub-cloud, we compute capacity-aware configurations for sites whose load cannot be absorbed by a single site. When our approach needs to assign multiple prefixes (§5), we assume traffic is split evenly across them before failure for simplicity. We then announce prefix(es) at backup sites to specific providers according to the capacity-aware configuration (§5), probe clients, and measure failover load on each site.

6.4.4 Results. We compare the load control of our techniques to anycast failover by emulating more than 150 site failures in the original cloud and 21 sub-clouds. 4For each failed site f, we compute two metrics: the maximum overloading rate among all backup sites ($\max_s \text{overload}(s, f)$) and the sum of overloading rates among all backup sites ($\sum_s \text{overload}(s, f)$) (see Definition 2 in Section 3.1). Figure 4 plots the CDF of these two metrics across all site failures. The two metrics are closely overlapped because, in most site failures, at most one site is overloaded. The capacity-aware configuration causes less than 0.01× overload at the 75th percentile and ≈ 0.1× at the 90th percentile. Comparatively, anycast causes 0.4× overload at median and 1× overload at the 90th percentile. Over all failure scenarios, our technique achieves 95% failoverControl (see Definition 2).

Figure 5 visualizes the total site load (including its normal and failover load) on the y-axis and site capacity on the x-axis for each backup site used in our evaluation. Following the capacity-aware configuration, most backup sites receive load below or slightly above their capacities, while anycast causes much more severe overloads. To further reduce overload, operators can leave extra headroom when computing the capacity-aware configuration. If a site has x% unused capacity, the operator can feed $(x - \delta)$% unused capacity to our technique's input. This may use more sites but adds a buffer to absorb routing uncertainty. We validate its efficacy in Appendix B.2.

6.5 Evaluation Summary: Establishing the Pareto Frontier

Table 1 compares our techniques and existing universal techniques. Compared to unicast, by sacrificing a small amount of control (<1%) with SelectiveProactiveDeprefer or a small amount of configuration stability (6%) with SelectiveReactiveAnycast, one can achieve much faster failover, similar to anycast's but without anycast's load control problems. If those sacrifices are unacceptable, ProactiveSuperprefix guarantees full control and stability, while achieving faster

failover than unicast (but slower than anycast and our other techniques). In combination, our techniques push what is achievable close to the (impossible) ideal (§3).

7 Related Work

Cloud redirection. Much prior work studied the performance and pitfalls of clouds through the lens of DNS and anycast. DNS TTL violations and non-compliant resolvers have long been a challenge for DNS-based clouds [2, 34, 46] to fail-over quickly. Anycast lacks this problem and performs well most of the time [13, 35] but lack of control can lead to performance and load management problems. Work from Akamai showed how DNS can map clients to a nearby site [15].

Our preliminary work proposed initial hybrid techniques combining the strengths of anycast and DNS redirection [70]. We extend that work in this paper via our demonstration that tradeoffs are inevitable (§3), our techniques that achieve better tradeoffs (§4.2, §4.3) and control failover load (§5), and our cloud-scale comparison between our techniques and prior ones (§6).

Stability. Google's study of failures and their impact argues for low risk operations [28]. Azure has published guidelines for safe deployment practices [54].

Routing and BGP. Prior work found that withdrawals tend to take longer than announcements to converge, with a median of 170 seconds [40]. We used this insight to refine our improvements over existing work. Prior work observed that anycast failover completes within 20 seconds for many clients [7], and our techniques leverage this fast failover. Other work assesses loss during route changes [63], which we also consider in our evaluation.

Load balancing. FastRoute uses DNS to steer traffic from overloaded anycast sites to datacenters [22], sacrificing valuable WAN capacity to make up for anycast's lack of control. The approach relies on DNS to shed load, which does not apply to our problem of enabling safe failover even before DNS is re-queried. Regional IP anycast, an approach to advertise a unique IP anycast prefix for sites in the same geographic region, has been deployed by several clouds, allowing them to assign client requests to a region [14, 30, 44, 69], but each region suffers the same problems intrinsic to anycast. Prior work also identified that anycast's loss of control causes overload and proposed a non-universal cloud architecture based on ISP-CDN collaboration [24].

TIPSY mitigates ingress congestion on specific peer links by predicting traffic shifts if specific routes are withdrawn and then withdrawing routes that will not lead to congestion on other links to "shift" traffic off the congested link [45]. Our problem requires fast rerouting after *site* failure, while minimizing BGP reconfiguration to preserve stability (§5). It is unclear how well TIPSY can predict where traffic will go when an entire site goes offline, how many BGP reconfigurations will be needed to restore availability, and how long that will take.

8 Conclusion

A New Pareto Frontier: We demonstrated the tension between availability, control, and BGP configuration stability—any universal Internet routing technique has to compromise on at least one goal. We introduced new techniques that achieve better tradeoffs than existing techniques. Depending on which goal a cloud is willing to slightly sacrifice, it can use one of our techniques to do well on that goal while optimizing the other two goals, allowing it to offer reliable, performant services with safer day-to-day operation.

Acknowledgements. Columbia's participation in this project was funded in part by NSF grants CNS-2344761, CNS-2323307, and OAC-2029295. Italo Cunha's work is partially funded by FAPESP grant 2023/00812-7, FAPEMIG grant APQ-02793-23, CNPq and CAPES. We thank Loqman Salamatian for his assistance in data analysis and visualization. We thank the anonymous reviewers for their insightful comments.

References

[1] Alibaba. 2021. A cooperative approach to content delivery. https://www.alibabacloud.com/help/en/express-connect/getting-started/locations-of-access-points

[2] Mark Allman. 2020. Putting DNS in Context. In *ACM IMC 2020*.

[3] Scott Anderson, Loqman Salamatian, Zachary S Bischof, Alberto Dainotti, and Paul Barford. 2022. iGDB: Connecting the Physical and Logical Layers of the Internet. In *ACM IMC*.

[4] APNIC. 2023. https://stats.labs.apnic.net/aspop/. https://stats.labs.apnic.net/aspop/

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

[6] Amazon AWS. 2023. How CloudFront delivers content to your users. https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/HowCloudFrontWorks.html#HowCloudFrontWorksContentDelivery

[7] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. 2006. A Measurement-Based Deployment Proposal for IP Anycast. In *ACM IMC*.

[8] Ryan Beckett and Aarti Gupta. 2022. Katra: Realtime verification for multilayer networks. In *USENIX NSDI*.

[9] Ryan Beckett, Aarti Gupta, Ratul Mahajan, and David Walker. 2017. A general approach to network configuration verification. In *ACM SIGCOMM*.

[10] Ryan Beckett and Ratul Mahajan. 2019. Putting network verification to good use. In *ACM HotNets*.

[11] Ann Bednarz. 2023. Global Microsoft cloud-service outage traced to rapid BGP router updates. https://www.networkworld.com/article/3686531/global-microsoft-cloud-service-outage-traced-to-rapid-bgp-router-updates.html

[12] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. 2013. Mapping the expansion of Google's serving infrastructure. In *ACM IMC*.

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

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

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

[16] David Chou, Tianyin Xu, Kaushik Veeraraghavan, Andrew Newell, Sonia Margulis, Lin Xiao, Pol Mauri Ruiz, Justin Meza, Kiryong Ha, Shruti Padmanabha, Kevin Cole, and Dmitri Perelman. 2019. Taiji: Managing Global User Traffic for Large-Scale Internet Services at the Edge. In *ACM SOSP*.

[17] Cloudflare. 2024. How Cloudflare's systems dynamically route traffic across the globe. https://blog.cloudflare.com/meet-traffic-manager/

[18] Cogent. 2024. Cogent Customer User Guide. https://cogentco.com/files/docs/customer_service/guide/global_cogent_customer_user_guide.pdf

[19] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, Sandeep Hebbani, Gaya Nagarajan, Omar Baldonado, Lixin Gao, and Ying Zhang. 2023. EBB: Reliable and Evolvable Express Backbone Network in Meta. In *ACM SIGCOMM*.

[20] Marwan Fayed, Lorenz Bauer, Vasileios Giotsas, Sami Kerola, Marek Majkowski, Pavel Odinstov, Jakub Sitnicki, Taejoong Chung, Dave Levin, Alan Mislove, Christopher A. Wood, and Nick Sullivan. 2021. The Ties that un-Bind: Decoupling IP from Web Services and Sockets for Robust Addressing Agility at CDN-Scale. In *ACM SIGCOMM*.

[21] Nick Feamster. 2004. Practical verification techniques for wide-area routing. *ACM SIGCOMM CCR*.

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

[23] Ashley Flavel, Pradeepkumar Mani, and David A Maltz. 2014. Re-evaluating the responsiveness of DNS-based network control. In *IEEE LANMAN*.

[24] Qiang Fu, Bradley Rutter, Hao Li, Peng Zhang, Chengchen Hu, Tian Pan, Zhangqin Huang, and Yibin Hou. 2018. Taming the wild: A scalable anycast-based CDN architecture (T-SAC). *IEEE Journal on Selected Areas in Communications* (2018).

[25] Phillipa Gill, Michael Schapira, and Sharon Goldberg. 2014. A Survey of Interdomain Routing Policies. *ACM SIGCOMM CCR*.

[26] Google. 2021. A cooperative approach to content delivery. https://cloud.google.com/network-connectivity/docs/interconnect/concepts/service-providers

[27] Google. 2023. Google OR-Tools. https://developers.google.com/optimization

[28] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or die: High-availability Design Principles Drawn from Googles Network Infrastructure. In *ACM SIGCOMM*.

[29] GTT. 2024. BGP Communities for IP Transit | AS3257 - GTT. https://www.gtt.net/us-en/services/managed-networking/internet/ip-transit/bgp-communities/
[30] Shuai Hao, Yubao Zhang, Haining Wang, and Angelos Stavrou. 2018. End-Users get maneuvered: Empirical analysis of redirection hijacking in content delivery networks. In *USENIX Security*.
[31] Tamás Hauer, Philipp Hoffmann, John Lunney, Dan Ardelean, and Amer Diwan. 2020. Meaningful Availability. In *USENUX NSDI*.
[32] Rebecca Hersher. 2017. Amazon and the $150 Million Typo. https://www.npr.org/sections/thetwo-way/2017/03/03/518322734/amazon-and-the-150-million-typo
[33] Santosh Janardhan. 2021. Facebook: More details about the October 4 outage. https://engineering.fb.com/2021/10/05/networking-traffic/outage-details/
[34] Jaeyeon Jung, E. Sit, H. Balakrishnan, and R. Morris. 2001. DNS Performance and the Effectiveness of Caching. In *ACM SIGCOMM IMW*.
[35] Thomas Koch, Ethan Katz-Bassett, John Heidemann, Matt Calder, Calvin Ardi, and Ke Li. 2021. Anycast In Context: A Tale of Two Systems. In *ACM SIGCOMM*.
[36] Thomas Koch, Shuyue Yu, Sharad Agarwal, Ethan Katz-Bassett, and Ryan Beckett. 2023. PAINTER: Ingress Traffic Engineering and Routing for Enterprise Cloud Networks. In *ACM SIGCOMM*.
[37] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. 2009. Moving beyond end-to-end path information to optimize CDN performance. In *ACM IMC*.
[38] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. 2022. Decentralized cloud wide-area network traffic engineering with BLASTSHIELD. In *USENIX NSDI*.
[39] ISI ANT Lab. 2022. IPv4 Hitlists. https://ant.isi.edu/datasets/ip_hitlists/
[40] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. 2000. Delayed Internet Routing Convergence. In *ACM SIGCOMM*.
[41] Raul Landa, Lorenzo Saino, Lennert Buytenhek, and João Taveira Araújo. 2021. Staying Alive: Connection Path Reselection at the Edge. In *USENIX NSDI*.
[42] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2018. Internet Anycast: Performance, Problems, & Potential. In *ACM SIGCOMM*.
[43] Bingzhe Liu, Colin Scott, Mukarram Tariq, Andrew Ferguson, Phillipa Gill, Richard Alimi, Omid Alipourfard, Deepak Arulkannan, Virginia Jean Beauregard, Patrick Conner, P. Brighten Godfrey, Xander Lin, Joon Ong, Mayur Patel, Amr Sabaa, Arjun Singh, Alex Smirnov, Manish Verma, Prerepa V Viswanadham, and Amin Vahdat. 2024. CAPA: An Architecture For Operating Cluster Networks With High Availability. In *USENIX NSDI*.
[44] Ritesh Maheshwari. 2015. TCP over IP Anycast—Pipe dream or Reality? https://engineering.linkedin.com/network-performance/tcp-over-ip-anycast-pipe-dream-or-reality
[45] Michael Markovitch, Sharad Agarwal, Rodrigo Fonseca, Ryan Beckett, Chuanji Zhang, Irena Atov, and Somesh Chaturmohta. 2022. TIPSY: Predicting Where Traffic Will Ingress a WAN. In *ACM SIGCOMM*.
[46] Giovane C. M. Moura, John Heidemann, Ricardo de O. Schmidt, and Wes Hardaker. 2019. Cache Me If You Can: Effects of DNS Time-to-Live. In *ACM IMC*.
[47] Netflix. 2021. A cooperative approach to content delivery. https://openconnect.netflix.com/Open-Connect-Briefing-Paper.pdf
[48] NTT. 2024. Routing policies. https://www.gin.ntt.net/support-center/policies-procedures/routing/
[49] Onestep. 2024. as3356. https://onestep.net/communities/as3356/
[50] Onestep. 2024. BGP Community Guides. https://onestep.net/communities/
[51] Jeffrey Pang, Aditya Akella, Anees Shaikh, Balachander Krishnamurthy, and Srinivasan Seshan. 2004. On the responsiveness of DNS-based network control. In *ACM IMC*.
[52] Matthew Prince. 2023. Post mortem on the Cloudflare Control Plane and Analytics Outage. https://blog.cloudflare.com/post-mortem-on-cloudflare-control-plane-and-analytics-outage/
[53] RADB. 2024. RADB. https://www.radb.net/query?advanced_query=&keywords=as3491
[54] Mark Russinovich. 2020. Advancing safe deployment practices. https://azure.microsoft.com/en-us/blog/advancing-safe-deployment-practices/
[55] Sandvine. 2023. Global Internet Phenomena Report 2023. https://www.sandvine.com/global-internet-phenomena-report-2023
[56] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. 2019. PEERING: Virtualizing BGP at the Edge for Research. In *ACM CoNEXT*.
[57] Brandon Schlinker, Italo Cunha, Yi-Ching Chiu, Srikanth Sundaresan, and Ethan Katz-Bassett. 2019. Internet Performance from Facebook's Edge. In *ACM IMC*.
[58] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K Sitaraman. 2020. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *ACM SIGCOMM*.

[59] Rachee Singh, Sharad Agarwal, Matt Calder, and Paramvir Bahl. 2021. Cost-effective Cloud Edge Traffic Engineering with Cascara. In *USENIX NSDI*.
[60] Alan Tang, Ryan Beckett, Steven Benaloh, Karthick Jayaraman, Tejas Patil, Todd Millstein, and George Varghese. 2023. Lightyear: Using modularity to scale BGP control plane verification. In *ACM SIGCOMM*.
[61] Twelve99. 2024. Twelve99/AS1299 BGP Communities. https://www.arelion.com/our-network/bgp-routing/bgp-communities
[62] Vultr. 2023. Announce your IP Space with BGP and Vultr. https://www.vultr.com/features/bgp/
[63] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. 2006. A Measurement Study on the Impact of Routing Events on End-to-End Internet Path Performance. In *ACM SIGCOMM*.
[64] Lan Wei and John Heidemann. 2017. Does Anycast Hang Up on You? In *TMA*.
[65] Jing'an Xue, Weizhen Dang, Haibo Wang, Jilong Wang, and Hui Wang. 2019. Evaluating performance and inefficient routing of an anycast CDN. In *IWQoS*.
[66] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. 2017. Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering. In *ACM SIGCOMM*.
[67] Shuyue Yu, Thomas Koch, Ilgar Mammadov, Hangpu Cao, Gil Zussman, and Ethan Katz-Bassett. 2025. Internet Service Usage and Delivery As Seen From a Residential Network. In *ACM SIGMETRICS*.
[68] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, and Xiaowei Yang. 2021. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *ACM SIGCOMM*.
[69] Minyuan Zhou, Xiao Zhang, Shuai Hao, Xiaowei Yang, Jiaqi Zheng, Guihai Chen, and Wanchun Dou. 2023. Regional IP Anycast: Deployments, Performance, and Potentials. In *ACM SIGCOMM*.
[70] Jiangchen Zhu, Kevin Vermeulen, Italo Cunha, Ethan Katz-Bassett, and Matt Calder. 2022. The best of both worlds: high availability CDN routing without compromising control. In *ACM IMC*.
[71] Yaping Zhu, Benjamin Helsley, Jennifer Rexford, Aspi Siganporia, and Sridhar Srinivasan. 2012. LatLong: Diagnosing Wide-Area Latency Changes for CDNs. *IEEE Transactions on Network and Service Management*.

A Additional Analysis of Existing Techniques

A.1 Anycast Control

In Section 2.3 and Section 3.1, we point out that anycast has low control. To demonstrate this and motivate the need for new techniques we present (§4), we measure anycast's loss of control (and performance) for Vultr, the cloud provider we use for experiments throughout this paper, relative to a common goal of routing clients to the lowest latency site. We use 28 Vultr sites globally, each connected to 3-5 transit providers and dozens to hundreds of peers. In ISI's hitlist [39], we identify ≈3.6M responsive addresses in 65K ASes. For each, we measure unicast and anycast round-trip latency by announcing from sites and pinging the addresses. Figure 6 presents the CDF of anycast inflation across all responsive addresses. Anycast inflation is computed as the latency difference between the anycast site and the lowest-latency unicast site. We find that for half of the targets anycast performs worse than unicast, with more than 100ms additional delay at the 90th percentile.

A.2 DNS-based failover is slow, hurting availability

To evaluate how quickly DNS can redirect client traffic after a site failure, we analyze DNS behaviour for several globally popular applications (according to Sandvine [55]), blocking IP addresses to emulate failures. We categorize applications into those returning single versus multiple IP addresses in DNS A responses. For single-IP applications (*e.g.,* Facebook, Twitch), we measure the delay before the application sends a new DNS query after that IP address is blocked. For multi-IP applications (*e.g.,* Netflix, YouTube), we measure the delay before the application switches to a different IP address after the active one is blocked. We test with various browsers, applications and OSes.

We find variability in DNS behaviours across OSes and applications. For single-IP applications, MacOS waits ~75 seconds before it issues a new DNS query, and Windows waits 22-100 seconds. In addition, issuing a new DNS query does not guarantee new responses as resolvers may return

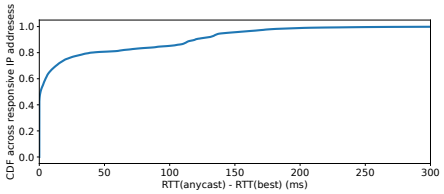


Fig. 6. Anycast inflation in an anycast deployment in Vultr. Anycast inflates the latency on half of the clients. At the 90th percentile, latency is inflated by 100ms.

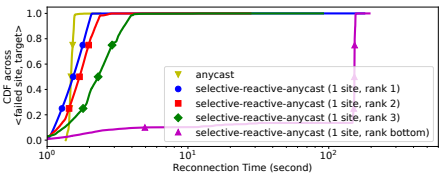


Fig. 7. Reconnection time of SelectiveReactiveAnycast using different single backup sites for the Bangalore failed site. Backup sites sharing a high number of providers and peers with Bangalore achieve high availability comparable to anycast, whereas sites sharing fewer providers and peers can lead to lower availability.

cached results, further delaying failover. For multi-IP applications, macOS and iOS clients retry after about 100 ms, while Windows clients typically wait around 22 seconds, although some applications, like Google Chrome, may delay retries for up to 130 seconds. We take the delay between failure and an application's use of new IP addresses, assume equal users across the applications, and use this data to generate our estimate of reconnection and failover time for DNS-based failover in Figure 3.

Though limited in scale, these results likely generalize to many applications and users of a cloud, as some popular applications and major operating systems consistently take more than a minute to either resend DNS queries or switch to backup IP addresses. Changing this behaviour requires widespread updates to client OSes, browsers, and applications, and is thus unlikely to happen in the near future. To summarize, DNS-based failover can take minutes, causing low availability (§2.2).

B Additional Details for Evaluation

B.1 Additional evaluation of availability

Section 6.3 evaluates the availability of our techniques, using either all backup sites or one backup site that shares the most number of providers and peers. We conduct additional experiments using different backup sites. We find that the top three backup sites, ranked by their number of shared providers and peers, all provide high-availability single-site backup options, providing a cloud with flexibility in configuring backups.

Similar to Section 6.3.1, we evaluate availability with the same targets under 6 different site failure scenarios (located in Singapore, Tokyo, Bangalore, Amsterdam, Paris, and Seattle), one at a time. For each of the top three backup sites per failed site, we evaluate both SelectiveReactiveAnycast and SelectiveProactiveDeprefer using that one backup site. The top three sites always provide high availability comparable to anycast when they serve as the backup site individually. On the other hand, low-ranked sites, with fewer or no providers and peers in common with the failed site, can sometimes lead to inflated reconnection and failover times.

We present a case study of a site failure at Bangalore. Figure 7 presents the reconnection time of SelectiveReactiveAnycast when using each of the top three sites ranked by their numbers of shared providers and peers as backup sites individually. It also presents the reconnection time when using a bottom-ranked site (which shares no common provider and few peers with the Bangalore site) as the backup site and anycast. We observe that using sites with shared providers and peers achieves short reconnection time ($\approx$2 s at the median), but using sites with few shared providers and peers causes a very long reconnection time (over 100 s at the median).

This validates that using backup sites with sufficient shared providers preserves availability, which gives flexibility in configuring the backup routes.

B.2 Additional evaluation of load control

In Section 6.4.1, we evaluate with load assignments assuming that each site has a fixed load in normal operation. In practice, backup sites' load can be dynamic. Their loads at the time of a site failure depends on factors including their timezones relative to the failed site. To begin understanding how dynamic load impacts our ability to recover from a failure without overloading other sites, we assign site load in a way that takes diurnal load patterns into account because, at the time of failure, many sites may operate in off-peak times and have most of their capacity unused. We still assume each site is overprovisioned by 40% based on its *peak* traffic, and the peak traffic is assigned based on its site population. But a site has higher fractions of capacity unused at non-peak hours in a day, based on diurnal patterns observed in a real CDN [16]. When a specific site fails, we assume it is at its peak time (so the amount of its failover traffic is maximum) and compute the free capacity at other backup sites based on their time zones. For example, when the Tokyo site fails, we assume it is at the peak time, so another site nearby, such as Seoul, is also at peak time so has 40/(100+40) of its total capacity left to absorb failover traffic, while a site in a very distant timezone, such as Los Angeles, may have 90% of its capacity free. We also experiment by assuming that the failed site is at off-peak times and find that the capacity-aware configuration for its peak time can always work.

The left graph in Figure 8 presents the results. In most failure scenarios, our technique achieves little or no overloading on sites, similar to our evaluation with fixed site load (Fig. 5). However, we note that with diurnal-pattern load assignment, backup sites tend to receive loads that are much lower than their capacity, because they are very likely to operate in their off-peak times.

In Section 6.4.1, we observe that sites have large differences in their load assignments (e.g., 14% vs. 1%) due to Vultr's small presence in APAC relative to the Internet population there. The load difference is an effect of our generation of load numbers based on population and our lack of knowledge about the cloud's real load. However, a cloud may have more even load distribution across its sites. So, to simulate another cloud with smaller differences between sites' load, we apply a sublinear function (square root) to the site population (§6.4.1) and assign load of each site in proportion to the function value. The right graph in Figure 8 shows the results. Similar to our evaluation with load assignment in Section 6.4.1 (Fig. 5), our technique reduces overload. The sites' capacities are much closer to each other due to the use of a sublinear function.

We summarize that our load control technique works for varied or dynamic load assignments, not just one assignment (§6.4.1). This suggests that it will likely work for real cloud deployments.

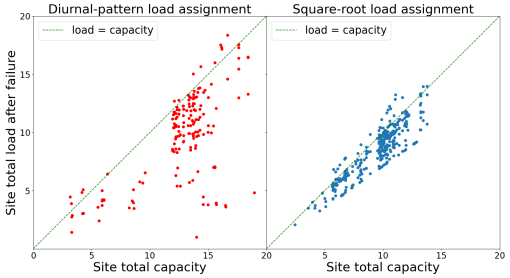


Fig. 8. Our load control technique achieves similar overload rates despite using a different load assignment than Section 6.4.1.

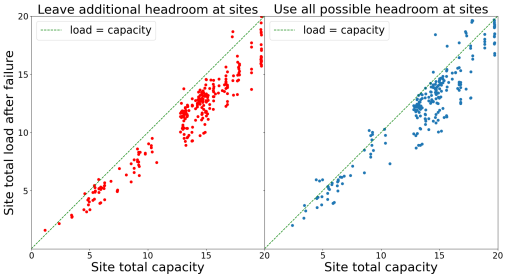


Fig. 9. Load control is improved with additional headroom reserved. Fewer sites are overloaded.

Although our approach avoids overloading most sites, sometimes clients arrive via a different provider than we predict and hence arrive at a different site than we assign—in other words, our model of routing preferences was incorrect, and so our bin packing solution was based on incorrect input. To guard against this type of overloading, an operator can reserve additional headroom

on backup sites when computing the capacity-aware configuration. We conduct experiments by setting a 50% overprovisioning rate on sites but only allowing the bin packing to use up to 80% of the free capacity of each site, reserving the rest to accommodate for inaccuracy. Figure 9 shows the result. The left one shows the sites' loads when the bin packing solver leaves additional capacity, and the right one shows the sites' loads when fully utilizing their capacity. We find that leaving some extra headroom at each site reduces chances of overload while slightly increasing the median number of backup sites from 3 to 4.

C Optimization problem for capacity-aware configuration

Section 5 introduces our approach to load control after failure at a high level. Below we detail how we formulate it as a bin packing optimization problem with constraints to reflect the underlying BGP routing. For simplicity, here we assume that there is only one prefix for the failover and that cloud can always assign a client network to a provider with certainty. We will address later how to modify the inputs to allow optimization for multiple failover prefixes and clients with uncertainty in their provider preference.

Formulation of the optimization problem. Table 2 shows the inputs and outputs for the optimization problem. The optimization objective is to minimize $\sum_s y_s$ (the number of backup sites), subject to the following constraints.

(1) $\forall c \in C, \sum_p \sum_s x_{cp} \cdot t_{ps} = 1$.
If $x_{cp} \cdot t_{ps} = 1$, it implies that the client network c fails over to site s through provider p. The constraint means that each client network is assigned to one provider, and each provider is assigned to one site.

(2) $\forall s \in S, \sum_c \sum_p x_{cp} \cdot t_{ps} \cdot v_c \leq b_s \cdot y_s$
The constraint means that each site receives traffic volume that does not exceed its free capacity. It also implies that if site s receives non-zero traffic, then y_s must be 1 (otherwise the right hand side is 0 while the left hand side is greater than 0).

(3) $\forall p \in P, \sum_s t_{ps} \leq 1$
The constraint means that we will assign a provider's traffic to at most one site.

(4) $\forall c \in C, \forall p \notin P_c, x_{cp} = 0$
The constraint means that a client network c should only fail over through a provider in its preferred provider set P_c.

(5) $\forall p \in P, \forall s \notin S_p, t_{ps} = 0$
The constraint means that we could only assign a provider's traffic to a site if the provider is connected to that site.

Inputs

c, C	c: a client network, C: the set of client networks
v_c	The traffic volume from client network c
s, S	s: a backup site, S: the set of backup sites.
b_s	The free capacity of site s
p, P	p: a provider, P: the set of providers on the failed site
S_p	The set of sites that provider p connects to
P_c	The set of providers preferred by client network c Cloud can assign c to one of them.

Outputs

x_{cp}	Binary $x_{cp} = 1$ iff client network c is assigned to ingress through provider p
t_{ps}	Binary $t_{ps} = 1$ iff provider p is assigned to send failover traffic to site s
y_s	Binary $y_s = 1$ iff site s receives failover traffic

Table 2. Notations for optimization problem

Procedure of finding capacity-aware configuration. We start by using one prefix, and increase if no provider-to-site assignment can be found. When using multiple prefixes (say k), we modify the inputs to the algorithm in the following way. For each provider network p, we create k (virtual) provider networks, where each connects to the same sites that p connects to (S_p). For each client network c, we create k (virtual) client networks with the same preferred providers as c (P_c) but replace them with the corresponding virtual providers. For example, if a client network prefers provider NTT, then its ith virtual client network prefers the corresponding ith virtual provider NTT. Finally, each virtual client network shares $1k$ of the original traffic volume v_c. This modification essentially allows the algorithm to potentially split a client network across different providers as well as assign a provider to different sites on a per-prefix basis. We then iterate through all possible provider groupings (§5). In each iteration, we apply conservative headroom reservation (§5) for those clients that we are uncertain which group of providers they will reach the cloud through by creating (virtual) duplicates of such clients, each having a fixed preference towards one group of providers. We then solve the optimization problem using Google's OR-Tools [27] to assign a client to a provider (group). At the end of iterations, we pick the provider grouping that minimizes the number of backup sites.

D Ethics

Our probing (§6) follows well-established ethical measurement practices. Our probe packets included a web link to opt out, but no one opted out. We probed at a low rate (<100B/s per target). Our work raises no other ethical concerns.

Received June 2025; accepted September 2025