AnyOpt: Predicting and Optimizing IP Anycast Performance

Xiao Zhang[★°], Tanmoy Sen[†], Zheyuan Zhang[†], Tim April[°], Balakrishnan Chandrasekaran[‡], David Choffnes[♣], Bruce M. Maggs[★◇], Haiying Shen[†], Ramesh K. Sitaraman[♠°], Xiaowei Yang[★]
[★]Duke University, [°]Akamai Technologies, [†]University of Virginia,
[‡]Vrije Universiteit Amsterdam, [♣]Northeastern University, [◇]Emerald Innovations, [♠]University of Massachusetts Amherst

ABSTRACT

The key to optimizing the performance of an anycast-based system (e.g., the root DNS or a CDN) is choosing the right set of sites to announce the anycast prefix. One challenge here is predicting catchments. A naïve approach is to advertise the prefix from all subsets of available sites and choose the best-performing subset, but this does not scale well. We demonstrate that by conducting pairwise experiments between sites peering with tier-1 networks, we can predict the catchments that would result if we announce to any subset of the sites. We prove our method in a simplified model of BGP, consistent with common BGP routing policies, and evaluate it in a real-world testbed. We then present AnyOpt, a system that predicts anycast catchments. Using AnyOpt, a network operator can find a subset of anycast sites that minimizes client latency without using the naïve approach. In an experiment using 15 sites, each peering with one of six transit providers, AnyOpt predicted site catchments of 15,300 clients with 94.7% accuracy and client RTTs with a mean error of 4.6%. AnyOpt identified a subset of 12 sites, announcing to which lowers the mean RTT to clients by 33ms than a greedy approach that enables the same number of sites with the lowest average unicast latency.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; **Network measurement**; **Network performance modeling**.

KEYWORDS

Anycast, Routing, BGP, Performance Optimization

ACM Reference Format:
Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, Xiaowei Yang. 2021. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *ACM SIGCOMM 2021 Conference (SIGCOMM '21), August 23–27, 2021, Virtual Event, USA*. ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3452296.3472935

1 INTRODUCTION

IP anycast [27, 30] is the practice of announcing the same IP address prefix from multiple network locations, and it is commonly used for load balancing and latency reduction. In part due to its inherent support in the routing system and potential for improving performance, anycast is used in large and popular services such as DNS (to distribute DNS query load [26, 34]), content delivery networks (CDNs) (to reduce latency between servers and clients [11]), and distributed denial of service (DDoS) mitigation services (to distribute and scrub attack traffic loads [28, 37]). A key challenge for deploying anycast services effectively is that mappings between client networks and anycast sites (i.e., anycast catchments) are determined by BGP's policy-based routing decisions rather than service providers' goals such as minimizing latency and balancing load. In fact, several measurement studies have revealed that some anycast catchments exhibit unexpectedly inflated latency [5], and increasing the number of anycast sites in a deployment (in an attempt to reduce the distance between clients and sites) counter-intuitively increases the average latency for clients and disrupts attempts to balance load [6, 24].

As a result, managing anycast deployments is a challenging task that requires expert knowledge and continuous intervention in response to BGP path changes, regular maintenance [11], or DDoS attacks [28]. Network operators lack tools that can accurately predict the system performance under different anycast configurations (i.e., the set of sites making announcements and the next-hop neighbors to whom the prefix is announced). Since BGP paths are determined by non-public policy information, such tools will require measurements or inferences for prediction. A naïve approach to measuring the impact of all potential announcements would require probing that scales exponentially with the number of sites under consideration. Using inferred topologies [35] to predict catchments can limit this cost, but it may introduce imprecision because of missing information in topology models and how BGP routers break ties among equally preferred paths.

In this paper, we address the above problems by using theoretical foundations to develop efficient measurement, prediction, and optimization techniques that allow an anycast operator to optimize a deployment for low latency while balancing load. This problem is important because latency is critical to the revenue generation of many Internet services [39]. Specifically, we present an experimental approach, *AnyOpt*, for predicting anycast catchments. A service operator can use AnyOpt to optimize an anycast network's deployment or dynamically reconfigure the network.

The key, empirically informed, insight that enables efficient catchment prediction is that most client networks, when given an option between any two (of potentially many) anycast sites, will consistently prefer one or the other. Further, we find that when considering *all pairs* of anycast sites, the set of pairwise preferences for a client network often forms a *total order*. This total order makes it straightforward to predict a site's catchment when we enable any subset of the anycast sites, as most client networks will consistently pick their

SIGCOMM '21, August 23–27, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8383-7/21/08…$15.00
https://doi.org/10.1145/3452296.3472935

most preferred sites in the subset. Furthermore, we observe that if a client network has a consistent total order among the anycast sites, we can map the anycast optimization problem to the Simple Plant Location with Preference Orderings [19] problem and solve it offline to find the subset of anycast sites that achieve the lowest overall latency.

Making AnyOpt accurate and efficient, however, requires addressing two key challenges. First, we find that not all client networks exhibit the consistent preferences that enable our approach. We use both theoretical analyses and measurements to understand why this problem occurs and whether consistent preference orders can predict a site's catchment. Our analyses reveal that a client network may not have a total order among preferences for anycast sites when autonomous systems (ASes) on the path of a BGP advertisement assign different local preferences to the advertisement. We prove sufficient conditions under which pairwise measurements yield a consistent total order and the total order is predictive of a client network's catchment. One example that meets these conditions is that we only announce an anycast prefix to tier-1 ISPs, and we adopt this setup for our real-world anycast testbed. Our experiments show that another cause of inconsistent preference orderings is a BGP implementation choice where ties between equally preferred paths are broken by the order in which a router receives BGP advertisements, which is not part of the BGP standard [32] but is implemented by most deployed routers (e.g., Cisco [1] and Jupiter [2]). Once we take into account both the policy-induced and implementation-induced inconsistent preference orders in AnyOpt, we show that we can expect consistent pairwise preferences. Then, we use the total orders constructed from those pairwise preferences to predict anycast catchments.

Second, we alleviate the issue of scaling AnyOpt measurements to large anycast deployments, e.g., those with hundreds or more sites [8]. For a deployment of this size, pairwise preference discovery experiments become impractical. For example, for an anycast network of 100 sites, if we space each pairwise experiment by two hours, which is necessary to avoid BGP instability, it would take years to finish all pairwise experiments. To address this challenge, we design AnyOpt to take a two-level approach to predict anycast catchments. The routing structure of the Internet makes the inter-AS and intra-AS anycast catchments two separate processes, where BGP determines the inter-AS catchments and the (interior) routing inside an AS determines the intra-AS catchments. Our experiments show that a site's catchment at the AS level remains stable when an anycast site is enabled or disabled within the same AS. Therefore, we can use pairwise experiments to discover client networks' AS-level preferences by choosing one representative site in each tier-1 AS that the anycast network connects to and run site-level pairwise experiments for sites within the same AS. If the latter is still prohibitive for a large network, we discuss a heuristic approach that might further reduce the number of BGP experiments needed for catchment prediction.

Our experiments on a real-world testbed show that AnyOpt can accurately predict anycast catchments and optimize client latency distribution, when announcing an anycast prefix to only tier-1 providers. We start with tier-1 network providers because they act as the backbone network that delivers the majority of the traffic for the testbed anycast network. To extend beyond the tier-1 providers, we adopt a heuristic to determine the impact of announcing via a peering link while simultaneously announcing to the tier-1 providers (§ 4.4).Our anycast testbed has 15 sites and connects with six tier-1 ASes. In the evaluation of transit-only configuration, we randomly choose an anycast configuration, predict its catchments and average latency to client networks, then deploy the configuration, and measure its actual catchments and average latency. Then we repeated this for 38 times. We find that AnyOpt predicts catchments with 94.7% accuracy and average RTTs with a mean error of 4.6%. In the offline configuration searching, AnyOpt identifies a 12-site lowest latency configuration that reduces the average client latency by more than 30ms compared to to configurations that greedily include sites with the lowest average unicast latency or randomly chosen sites. For the peering links, we also iterated through 104 peering links in the testbed and identified 47 peering links that can improve performance; more specifically, we find that including peering links in the 12-site lowest-latency configuration can further reduce the mean latency by 5ms to 7ms.

AnyOpt represents a crucial first step towards predicting and optimizing the performance of an anycast network. Specifically, this work makes the following contributions:

(1) We propose AnyOpt, an empirical approach that uses BGP measurements to reveal a client network's preferences between any two anycast sites, and then uses these to predict and optimize anycast network performance. We report for the first time how BGP announcement arrival orders impact anycast catchments and develop a technique to incorporate them into the catchment prediction. We use two-level prediction techniques to reduce the number of required experiments.
(2) We analyze the theoretical underpinnings for the heuristic approach and prove sufficient conditions for this approach to work.
(3) We use a real-world anycast testbed of a large content delivery network to evaluate AnyOpt. Our experiments show that AnyOpt can predict anycast performance accurately and can reduce the average latency to client networks by as much as 33ms (32%) compared to greedy approaches.

Ethical considerations. Active measurements such as issuing pings and BGP announcements can cause extra load on the Internet infrastructure. As discussed later in the paper, we mitigate these concerns by gathering our measurements at reasonably low rates, and target our measurements at routers (not end hosts). Our BGP announcements use only prefixes that we control and only our AS number in our announcements. The anycast prefixes we use do not serve any clients. This work raises no other ethical issues.

2 BACKGROUND

In this section, we briefly describe the architecture of an anycast network, define the terms we use, and use real-world anycast systems to motivate AnyOpt's design.

2.1 Architecture of an Anycast Network

Figure 1 illustrates the architecture of an anycast network. A service provider such as a CDN or a DDoS mitigation provider has servers that receive anycast traffic deployed at multiple locations. These servers offer services such as traffic scrubbing or caching. We refer to each location where these servers are deployed as an *anycast site*. A site has an onsite router that connects to one or more ASes. We

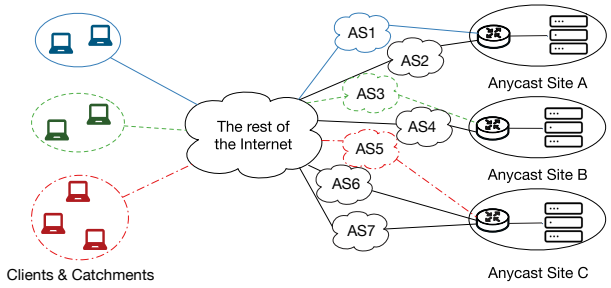


Figure 1: *The architecture of an anycast network.*

refer to each BGP connection to an outside AS as an *ingress point*. In Figure 1, the simple anycast network has three sites, each having two or three ingress points. We refer to the set of end systems reaching one site the *catchment* of the site. Figure 1 groups each site's catchment within an oval shape and marks the catchment's ingress AS/connection with the same line type.

2.2 Motivating Examples

A key motivating application of our work is the configuration management of systems such as Akamai DNS [34] or an anycast-based CDN [6, 11]. Akamai DNS is one of the world's largest authoritative DNS systems, serving millions of queries per second from a few hundred sites that host 24 distinct anycast prefixes. Each anycast prefix is hosted from a subset of about 30 sites that form an "anycast cloud". Each domain name hosted on Akamai DNS is assigned to a delegation set of about 6 anycast prefixes. When a recursive DNS resolver (i.e., client) requests an authoritative translation of a domain name, it sends the request to an anycast prefix that is in the delegation set of that domain name. The request is then routed to a site within that prefix's anycast cloud by BGP. That site then responds with the answer.

The key challenge in configuring Akamai DNS is assigning each of the 24 anycast prefixes to a subset of sites such that the average query response latency experienced by the resolvers is minimized. As network conditions (e.g., routing policy or load) change, the subset of sites that host each anycast prefix must be recomputed to maintain optimal anycast performance. Since the number of ways to configure an anycast cloud is exponentially large, it is infeasible to predict the catchments of sites accurately, and, consequently, impractical to estimate the query latency achieved in each configuration. The state-of-the-art for configuring large anycast networks such as Akamai DNS is based on Monte Carlo simulations [34]. Our work is focused on improving the state-of-the-art in configuring anycast networks. AnyOpt assists the problem of optimally configuring an anycast cloud using a principled measure-model-and-optimize approach that is directly applicable to real-world systems such as Akamai DNS.

An anycast-based CDN faces a similar configuration challenge. For a CDN service provider, the latency between a client and an edge server can have a multiplicative effect on page-load times, given the many round-trips typically required to download various resources. Therefore, reducing latency by even tens of milliseconds can result in a substantial reduction of page-load times [21]. Simply adding more anycast sites, however, does not necessarily improve the mean latency between the clients and an anycast network [24]. Even if some sites offer poor performance for clients, BGP may prefer these sites to others for policy reasons. In such cases, AnyOpt can reliably identify which anycast sites improve performance, obviating manual interventions.

Table 1: *Locations of the 15 anycast sites along with the transit providers and counts of peers at each location.*

Site	*Location*	*Transit*	*#peers*
1	Atlanta	*Telia*	4
2	Amsterdam	*Telia*	1
3	Los Angeles	*Zayo*	6
4	Singapore	*TATA*	15
5	London	*GTT*	14
6	Tokyo	*NTT*	3
7	Osaka	*NTT*	4
8	Los Angeles	*Zayo*	4
9	Miami	*NTT*	7
10	London	*Sparkle*	2
11	Newark	*NTT*	7
12	Stockholm	*Telia*	14
13	Toronto	*TATA*	9
14	São Paulo	*Sparkle*	9
15	Chicago	*GTT*	5

2.3 Anycast Configuration

Although an anycast network cannot control the catchment of a site, it can "shape" the catchment with three control knobs: *(1)* the sites from which it announces an anycast prefix, *(2)* the ASes to which it announces the prefix at a particular site, and *(3)* the BGP path attributes it uses when announcing the prefix. Specifically, if we use S to denote the set of sites an anycast network has (or considers to open), the network can choose to announce an anycast prefix from any subset of S. For each anycast site s_i, let's denote the set of ASes it connects to as P_{s_i}. The service provider can choose any subset of P_{s_i} to announce the prefix. For each BGP announcement, the network can vary the parameters associated with the announcement, including the Multiple Exit Discriminator (MED) and the AS path length.

There are, hence, more than $2^{\sum|P_{s_i}|}$ possible ways to configure an anycast network. As a first step, in this work, we explore how an anycast network can optimize its performance by finding (a) from which subset of sites to announce the anycast prefix and (b) to which ASes at each site to announce the anycast prefix. We assume that an anycast network sets the path attributes to default values when it announces an anycast prefix. We call a site or an AS as "enabled," when it is chosen to announce an anycast prefix. We refer to the combination of the chosen subset of sites and the chosen ASes at each site as an anycast configuration.

3 OVERVIEW

In this section, we describe the anycast testbed used in this work, the experiments for discovering a client network's pairwise preferences, and the high-level idea of using the preference orders to predict and optimize the performance of an anycast network.

3.1 Anycast Testbed

Our testbed consists of an instance of GoBGP (version 2.14.0) [29], an open-source BGP implementation, running on an Ubuntu (18.04.3 LTS) server with 4 cores and 16GB of RAM. The GoBGP instance uses generic routing encapsulation (GRE) tunnels [13] to peer with

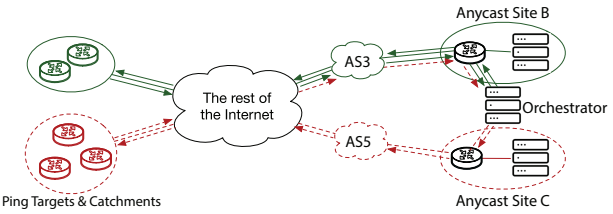


Figure 2: *This figure illustrates the testbed we use for RTT and catchment measurements.*

a large CDN's routers at different locations as described in Table 1. The routers at different locations serve as anycast sites. Each anycast site has a tier-1 transit provider to ensure global reachability, i.e., any client or end-user in the Internet can reach the anycast site. In addition to the transit provider, each site peers with a few other ASes (Table 1), including some under a settlement-free policy, i.e., where neither network pays the other for transit. We launch all active experiments and collect our measurements from the Ubuntu server, which we henceforth refer to as the *orchestrator*.

We deploy an anycast configuration as follows. First, we establish BGP sessions between the orchestrator and the routers at chosen anycast sites. Then, we program GoBGP [29] to announce an anycast prefix assigned to us via the BGP sessions between the orchestrator and the site routers. The router at an anycast site likely peers with multiple other neighbors. We use, hence, BGP's community attribute to control to which next-hop neighbors the router should advertise our anycast prefix. We can choose, therefore, to advertise to a site's transit provider or any chosen peer by appropriately setting the community attribute.

We develop a measurement tool that is similar to Verfploeter [12]. We run this tool at the orchestrator to measure the catchments. We also improve the tool to measure the RTT between any client and any anycast site as we soon describe.

Measuring catchments. For a given anycast configuration, we measure each site's catchment. We use these measurements in two ways. First, we use them to determine a client network's preference between two anycast sites, henceforth referred to as *pairwise preference*. Second, we compare the predicted catchments with the empirically observed catchments. To gather the measurements, we send ICMP requests [31] from the orchestrator to a large number of ping targets, which are routers in different client networks chosen by the CDN hosting our experiments to evaluate its network's global performance. We set the source address of each ICMP request to an IP anycast address that we advertise and its destination address to a target's IP address. When a target responds to this request, i.e., to the anycast address, it will be routed to an enabled anycast site. The router at the site will then forward that reply to the orchestrator, via a preconfigured GRE tunnel. The GRE tunnel carrying the reply, thus, identifies the target's catchment site.

Measuring RTTs. We can measure the RTT from the orchestrator to any target, but, for predicting and optimizing anycast performance, we must measure the RTT between the anycast site and the target. Rather than indirectly estimating the RTT between a site and a target from the physical distance between the two or approximating the RTT through appropriate proxies as in King [18], we use the following approach. First, we announce an anycast prefix from only one anycast site and send ICMP requests via the GRE tunnel connected to that site. We include a timestamp in the request for RTT measurements. Second, when the orchestrator receives a reply from a target, we subtract the echoed timestamp from the current time to calculate the RTT between the orchestrator and the target. Third, we periodically measure the "tunnel" RTT between the orchestrator and each anycast site. Finally, we subtract from the measured RTT between the orchestrator and a target, the corresponding tunnel RTT, i.e., the RTT between the orchestrator and the site through which the orchestrator received the target's ICMP responses. For each RTT measurement, we repeat the ICMP requests seven times and use the median value (to filter outliers) as the RTT between the concerned site and the target. If the link experience high packet loss rates, we can still sample a median RTT from at least three valid responses.

As an example, in Figure 2, suppose we only announce our prefix to AS3. Even though we send out the ICMP requests to the targets from both anycast site B and anycast site C, the ICMP replies will only return to anycast site B. Therefore, we can measure the RTT between any target and the orchestrator. By subtracting the tunnel RTT from the orchestrator to site B, we obtain the RTT between a target and site B.

3.2 Choosing Ping Targets

To understand the impact of anycast configurations on client networks' performances, we conduct active measurements (with ICMP probes). The targets of these ICMP measurements are routers in or near the targeted client networks.

To select our targets for measurements, we follow an approach used by the CDN hosting our testbed [22]. Specifically, we merge the network paths from the end-users to a CDN's edge server into a tree, rooted at the edge server. We then pick a common ancestor that is closest to the end-users. In our active measurements, we ping such targets from diverse networks to obtain a reasonable approximation of the global performance of end-users. Additionally, the targets also help us avoid sending ping probes to real end-users. Our target set contains 15,300 IP addresses from 12,143 /24 network prefixes or 5317 ASes. Each target is representative of one or more client networks.

3.3 Pairwise Preference Discovery

We conduct pairwise BGP experiments to elicit a client network's preference orders. For each experiment, we choose two sites s_i and s_j from the available anycast sites for comparison. We announce an IP anycast prefix from these two sites to *only* their corresponding transit providers, for reasons we soon describe in §4.1. If a ping target's response reaches site s_i instead of site s_j, we record that the client network prefers s_i to s_j. By pinging all targets in one experiment, we obtain all client networks' preferences between s_i and s_j.

3.4 Prediction and Optimization

With the RTT measurements and pairwise preference experiment results, we can predict an anycast configuration's performance and choose an optimal configuration. If a client network's set of pairwise preferences has no cycles, we can construct a total preference order for the network. For any subset of anycast sites enabled, we predict the client network's catchment site as its most preferred site within that subset. A client network's pairwise preferences may, however,

not form a total order, and we discuss why this situation may happen in §4.

Given the RTTs and preference predictions, we can map an anycast optimization problem to the Simple Plant Location Problem with clients' Preference Orderings (SPLPO) [19] for optimization. SPLPO is an extension to the well-known (uncapacitated) plant location problem [9]. It considers the problem of how to open facilities that have the overall lowest cost when each client has a preference order among the set of possible facility locations. If we consider a "facility location" as an anycast site, and use the *RTT* as the cost, then anycast performance optimization problem becomes exactly the SPLPO problem. The SPLPO problem is NP-hard [4], and we show in Appendix B that even approximating the minimum cost of SPLPO is NP-hard.

A network operator can, however, solve or approximate the optimization problem using offline simulations. When the number of anycast sites is small, he or she can solve it exhaustively. When the number of sites is large, he or she may not find the (theoretically) optimal configuration, but he or she can find a configuration that has the best performance among all configurations she simulates.

If an anycast network has a total of $|S|$ sites, each having one transit provider as in our testbed, then to optimize or predict an anycast network's performance, a network operator needs to run $O(|S|^2)$ pairwise experiments to obtain each client network's total preference order and $O(|S|)$ experiments to obtain a client network's RTT to each site. In contrast, if we do not employ the prediction or optimization technique, the operator needs to deploy $O(2^{|S|})$ anycast configurations to measure and compare the performance of each configuration. We formally describe the optimization model in Appendix B and show that the model can optimize for latency while meeting the load constraints of a site.

3.5 Practical Challenges

We have outlined the high-level idea behind AnyOpt's design. However, to make it useful, we must address the following challenges.

No total order. When we perform the pairwise preference discovery experiments, a client network may not exhibit consistent pairwise preferences that form a total order. Without a total order, we cannot predict a site's catchment for an arbitrary anycast configuration.

To address this challenge, we formally analyze the sufficient routing conditions under which a client network has a total order over a set of anycast sites and the total order predicts a site's catchment (§4.1). We modify the pairwise experiments described in this section to induce and discover a client network's total order.

Too many experiments. A naïve approach for pairwise preference discovery requires at least $O(|S|^2)$ BGP experiments. As it takes on the order of minutes for BGP to converge and routers implement route damping for frequently changing prefixes, conducting such experiments at scale may become impractical. Solving the optimization problem using an exhaustive search also becomes infeasible.

We reduce the number of experiments by separating AS-level catchment prediction from intra-AS catchment prediction (§ 4.3). This technique reduces the number of pairwise BGP experiments from $O(|S|^2)$ to $O(|I|^2)+O(avgSite^2 \times |I|)$, where I is the set of transit ISPs an anycast network connects to and $avgSite$ is the average number of sites connecting to a transit provider. For a large anycast network, this number of experiments may still be infeasible. We, hence, describe a heuristic method to approximate a client network's intra-AS site preferences. This heuristic can eliminate the need for intra-AS pairwise preference discovery experiments.

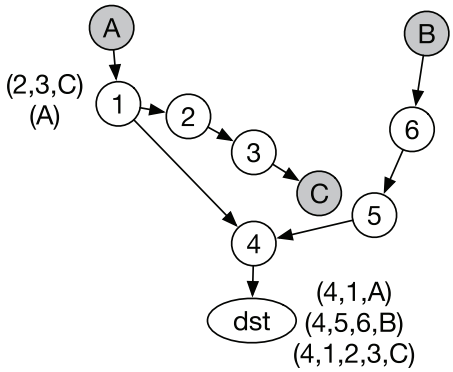


Figure 3: ***This example explains why a client network (dst) may not exhibit a total preference order among anycast sites A, B, and C. Arrows point from providers to customers. AS 1 prefers the path originated at site C, as it is a customer router, while AS 4 prefers the path from A, as it has a shorter AS path.***

4 DESIGN

Below, we discuss how we address the practical challenges that AnyOpt faces.

4.1 Sufficient Conditions for Total Orders

First, we investigate why client networks exhibit a total preference order and why this order can be used to predict anycast catchments. With this understanding, we can determine whether our experimental approach is generally applicable to other networks.

BGP routing model. We analyze anycast routing using the Gao-Rexford BGP routing model [15]. For simplicity, we consider two kinds of contractual relationships: provider-customer and peer-to-peer. In the former, a provider AS advertises a route received from a customer AS to all its other neighbors, while in the latter, a peer only advertises another peer's routes to its customers.

When a BGP router receives different route advertisements to the same prefix from its neighbors, it chooses the "best" path for reaching the prefix and advertises only the best path to its neighbors based on its export policies. The algorithm for determining the best path works as follows [32]. When a router compares two paths, it lexicographically compares two tuples, each consisting of an ordered list of attributes of the corresponding paths. The first element in the tuple of path attributes is local preference (`LOC_PREF`). An AS would generally prefer an economically profitable route. Therefore, under common BGP policies, an AS would prefer a customer route to a peer route and prefer a peer route to a provider route. When the routes have the same `LOC_PREF`, BGP breaks ties using the following path attributes, in the given order: AS path length, origin of prefix, `MED`, type of BGP session, interior cost, router Id, and neighbor address.

Why a total order might not exist? As a BGP path passes along from one AS to another, each AS may rank the paths differently. An

AS may prefer a path with a longer AS path length, while a downstream AS may prefer one with a shorter AS path length (all due to differences in LOC_PREF values at the two ASes). Suppose that A, B, and C are three anycast sites, and dst is a client network that receives anycast announcements (Figure 3). Each circle represents an AS and an arrowed line points from a provider AS to a customer AS. To elicit the preferences between the sites A and B, we use A and B to announce our anycast prefix. The client network dst will choose the path originated from site A, as both paths from A and B are provider routes, and the path from A has a shorter AS path. So we observe $A >_{dst} B$, where the operator $>_{dst}$ denotes "preferred by dst." When we compare the preferences between A and C, dst will choose C (i.e., $C >_{dst} A$), since AS 1 prefers a customer route to a provider route and will advertise only the path from C to AS 4. Finally, when we compare B and C, dst will choose B (i.e., $B >_{dst} C$), as it has the same LOC_PREF as the path from C but with a shorter AS path. This scenario leads to cyclic pairwise preferences—no total order.

Why do we observe total orders in practice? If a client network might not exhibit a total preference order under the common BGP model, why do our experiments observe so many instances of a consistent total order? To answer this question, we focus on the case where anycast sites peer only with tier-1 networks and make two assumptions: (a) any network that has settlement-free peering with a tier-1 network has settlement-free peering with all tier-1 networks; and (b) valley-free routing [15] holds. Then, if a network receives one or more announcements for an anycast prefix, then all of them should come from either peers (if the receiving network is a tier-1 network) or from providers (if the receiving network is not a tier-1 network). In selecting a path, a non-tier-1 network will, hence, only choose among paths advertised by providers. The available paths will, therefore, have the same LOC_PREF under the common BGP policy for any non-tier-1 network receiving the anycast prefix announcement, and the AS path length will be the most significant route selection criterion.

Furthermore, except for AS_PATH, the rest of BGP route attributes are all based on AS-local or router-local identifiers. We can view them collectively as one combined neighbor_ID. These local identifiers are numerical and therefore have a total order. Under these conditions, we prove in Appendix A that the following theorem holds.

Theorem 4.1. *If in a network a BGP speaker selects its best paths by comparing* (*AS_PATH*, *neighbor_ID*)*, then the paths from a client network to all available anycast sites form a total order. Pairwise preference comparison experiments are able to discover this total order, and this total order is predictive of a client network's catchment site when any subset of anycast sites are enabled.*

This sufficient condition suggests that if an anycast network announces an anycast prefix from only tier-1 transit providers, then under the common BGP routing policy, a client network will exhibit a total preference order among the anycast network's sites.

4.2 Practical BGP Implementation Issues

Below, we discuss two major challenges stemming from BGP implementations and how we address them.

Arrival orders of BGP advertisements. In our experiments, we observe that when we compare the same two sites, client networks may sometimes prefer different sites. This behavior is inconsistent with the BGP specification and introduces cyclic preferences in our experiments. Upon investigating this issue, we found that real-world BGP implementations use another attribute—the arrival time of a route advertisement—as a tie-breaker after the "interior cost" attribute. Both Cisco [1] and Juniper [2] describe this tie-breaking algorithm in their online documents, albeit the attribute is not part of the BGP specification [32]. Our empirical result shows that, after resolving the arrival order problem, the ratio of clients that have a consistent total order increases significantly. This result suggests that tie-breaking based on the arrival-order is a widespread implementation, and it is frequently triggered in a router's route selection process.

To cope with this implementation issue, we take the arrival orders of route advertisements into account in our experiment design. In our pairwise experiments, we explicitly discover the client networks that are affected by the arrival orders of a route advertisement and incorporate the arrival orders in anycast catchment prediction. Specifically, we space the route advertisements from two different sites by an interval T such that the first advertisement arrives earlier than the second at a client network globally. We measure each client network's catchment twice, with the route-advertisement order in the second experiment being the reverse of that in the first. If a network's preference stays the same across the two experiments, we conclude that the network has a strict preference order between these two sites; otherwise, we conclude that it has equivalent preferences.

Later, when predicting the catchments for an anycast configuration, we consider how the order of announcements would affect a site's catchment and use the corresponding pairwise comparison results to predict the catchments. For instance, if we choose a configuration of three sites A, B, and C, and we announce an anycast prefix in the order of first A, then B, and last C, we will use a client network's preference orders obtained from the measurements when A is announced before B, and B is announced before C for predicting the catchments.

Our experiments reveal that the order of BGP announcements primarily affects a network's preference at the AS-level. It does not have any effect on a network's preference orders when the prefix announcements are from different sites within the same AS.

Multi-path routing. Some routers may split traffic to the same destination prefix among multiple paths. A network's traffic may, as a consequence, reach different anycast sites, leading to inconsistent total orders. This practice of multi-path routing complicates the catchment prediction and could explain why the inconsistent total orders exhibited by some networks.

Most networks exhibit, however, consistent total orders after we take into account the arrival orders of route advertisements. We ignore the networks that continue to exhibit inconsistent total orders (i.e., even after taking route-announcement orders into account) from catchment prediction and optimization, but still include them when identifying catchments and measuring client RTTs under a given configuration.

4.3 Two-level Preference Discovery

A real-world anycast network, such as Akamai DNS [34], may have a few hundred sites. It is impractical to run pairwise measurements for

a network of this size. To reduce the number of preference measurements, we exploit the two-level structure—inter-AS and intra-AS—of routing in the Internet.

When one or more sites that connect to the same AS advertise an anycast prefix, the site-level differences disappear (i.e., cannot be observed) once a neighboring AS re-advertises the prefix to its neighbors. Suppose that a client network is not directly connected to an anycast site. Then, if we discover the client network's total preference order among the ASes that interconnect it to the anycast network, we can predict which ingress AS the network will use. Within that AS, the interior routing metrics determine which site the network will use. This routing structure allows us to separate the discovery of a client network's preference order at the AS-level from that at the site-level. More concretely, our two-level approach first predicts the AS-level (or provider-level) preferences of a client network, and then proceeds to discover the client network's site-level preferences, across available sites within an AS.

Provider-level preference discovery. To elicit a client network's pairwise preferences at the AS (or provider) level, we choose two transit providers and use one representative site from each provider for announcing the anycast prefix, as described in §3; we repeat the exercise across all pairs of transit providers. Recall that I denotes the set of transit providers of an anycast network. We need $O(|I|^2)$ pairwise experiments to discover a client network's total order. The experiments in our testbed show that when we vary the representative site or the number of representative sites for each transit provider, 94.2% of the client networks on average do not change their pairwise preferences.

Site-level preference discovery. To discover a client network's site-level preferences among anycast sites within each transit provider, we proceed as follows. We choose two sites for each transit provider, announce the anycast prefix and measure the client network's preference order; as before, we repeat this experiment with other site pairs. We found that the announcement order has no impact on the client network's site-level preferences.

Site-level preference discovery might still be prohibitively costly for a large anycast network. We could, however, use the following heuristic to eliminate this step. Once a client network's traffic enters an AS that hosts multiple anycast sites, that AS's interior routing protocol determines the network's catchment site, typically based on shortest path routing metrics. We can, therefore, approximate a client network's site-level preference order in a given AS using the shortest path distances from the client network's ingress point to the anycast sites inside the AS. Per our experiments in some tier-1 networks, the shortest-path distance is closely correlated with a client network's RTT to an anycast site. We use, therefore, the RTT from a client network to an anycast site to predict the site-level preference: the shorter the RTT, the more preferable the site.

Once both the provider-level and site-level preference-discovery steps are completed, a network operator can determine a network's preference order among its transit providers as well as across the sites within each transit provider. Armed with the preference orders and the RTT measurements, they can predict the catchments of a given anycast configuration for a specific announcement order, thereby predicting the latency and load distribution. They can furthermore simulate the performance of various anycast configurations and deploy the ones that best fit their performance requirements.

4.4 Incorporating Peers

With the steps above, AnyOpt can generate an optimal transit-only anycast configuration. However, an anycast network may include peering connections. As peering connections can be settlement-free and deliver traffic to client networks via shorter AS paths, incorporating them in an anycast configuration may improve performance and reduce transit cost. However, it is not straightforward how to incorporate peers in an anycast configuration, as previous work [7, 17] suggests that peer connections can worsen the performance of a transit-only anycast configuration.

While it is our future work to study how to incorporate peering connections in an anycast configuration in more depth, we develop a heuristic technique to conservatively include only the beneficial peers in an anycast configuration. We refer to this heuristic as the "one-pass" method. Again, it is based on measurements and offline optimization.

In the one-pass method, we first measure whether enabling a peer will reduce the average latency of the baseline configuration where only transit providers are enabled. If so, we consider the peer as a beneficial peer. From the optimal transit-only configuration found by AnyOpt, we enable one peer at a time, measure the peer's catchment after the peer is enabled, and measure how the average latency has changed. If the average latency is reduced, we mark this peer as a beneficial peer. We then disable this peer, measure another peer, and so on until we measure all peering connections of an anycast network. If an anycast network has a total of M peers, this step requires M BGP measurements.

After we identify the beneficial peers, we use a greedy offline algorithm to choose the set of peers to add to the transit-only configuration. We rank the beneficial peers by the size of their catchments measured during the one-pass experiments. We start with including the beneficial peer with the largest catchment, and then examine the beneficial peer with the second largest catchment, and so on. For each peer we consider, we estimate whether including this peer will reduce the average latency. If it will reduce the average latency, we include it in the configuration. Otherwise, we skip it. We note that the one-pass experiments we conduct only include one peer at a time. Thus we cannot predict the catchment of a peer and hence, the average latency, accurately when multiple peers are enabled simultaneously. To overcome this challenge, we conservatively assume that when we add a peer with a smaller catchment size, all client networks in the peer's catchment discovered in the one-pass experiments will switch to that peer. Only if the average latency is reduced in this case, we will include the peer. Otherwise, we will skip the peer.

Experimentally, we find that the one-pass method can further reduce the average latency of a transit-only configuration, but not by much. It is around 5ms for our testbed. It is our future study to investigate why the reduction is this small, but one plausible explanation is that the beneficial peers in total only attract a small fraction of the overall traffic in our testbed, as we show in § 5.4. This result may not hold for other anycast networks where peering connections attract larger amounts of traffic.

4.5 Putting it Together

Finally, we summarize the steps it takes to predict an anycast network's catchments and to optimize its performance.

(1) For each site, we announce a test anycast prefix to a transit provider the site connects to. We use this experiment to measure the RTT from a client network to this site (refer §3).
(2) We run pairwise preference measurements among all transit providers of the anycast network. We consider the impact of the arrival orders of BGP advertisements by announcing each pair twice with a reversed order in the second measurement to get enough information for simulating all possible announcement orders. We choose one representative site from each transit provider to perform these experiments.
For each transit provider, we use pairwise experiments to discover a client network's preferences among the sites within this transit provider. For a large anycast network with many sites where this approach is infeasible, we approximate a client network's preferences by its RTTs to the sites within the transit provider.
(3) Using the above experiments, we compute offline the total preference order of each client network for the announcement order that maximizes the number client networks with a consistent total order. We exclude a client network in this computation if its pairwise preferences do not exhibit a total order. We use the total order to predict the catchment site of a client network given a site-level anycast configuration. We can enumerate through as many configurations as required offline and choose the best ones to deploy. After the deployment, we can include the beneficial peering links discovered using the one-pass method described in §4.4.

Analysis. We now estimate the number of BGP measurements needed as well as the time it takes to finish them for optimizing the Akamai DNS anycast network [34] with a transit-only configuration. Akamai DNS has a few hundred sites. We use 500 sites and 20 transit providers to approximate the Akamai DNS network. For a network of this size, site-level pair-wise experiments are infeasible. We instead use a client network's RTTs to the anycast sites to approximate their intra-AS site-level preferences. In total, we require 500 singleton experiments for measuring a client network's RTT to each site and 380 pair-wise measurements for discovering the network's pair-wise preferences between any two transit providers. We can, however, run the BGP measurements in parallel with different anycast prefixes. Suppose that we use four test anycast prefixes (the number of prefixes we use in our testbed), and we separate each BGP experiment by two hours. Then the 500 singleton measurements will take $500*2/4=250$ hours or about 10 days to finish. The 380 pair-wise experiments will take $380*2/4=190$ hours or around eight days to finish. So for an anycast network of size as large as the Akamai DNS system, a network operator can perform these measurements once a month and use the results to adjust their network configurations. If the topological features of the Internet such as a client network's average RTT to a site remains stable over the course of a month, then AnyOpt is suitable for such large networks.

5 PERFORMANCE EVALUATION

In this section, we use real-world experiments on the anycast testbed described in §3.1 to evaluate AnyOpt. In particular, we answer the following questions:

(1) How does the order in which we announce an anycast prefix from different sites affects the catchment of each site?
(2) How effective are the pairwise preference elicitation experiments in discovering the total ordering of AS-level and site-level preferences of client networks in a provider-only anycast configuration?
(3) How accurately can we predict the catchments in a provider-only anycast configuration?
(4) Can AnyOpt's catchment prediction help in optimizing anycast deployment for performance (e.g., in terms of latency reduction)?

5.1 Pairwise Preference Discovery

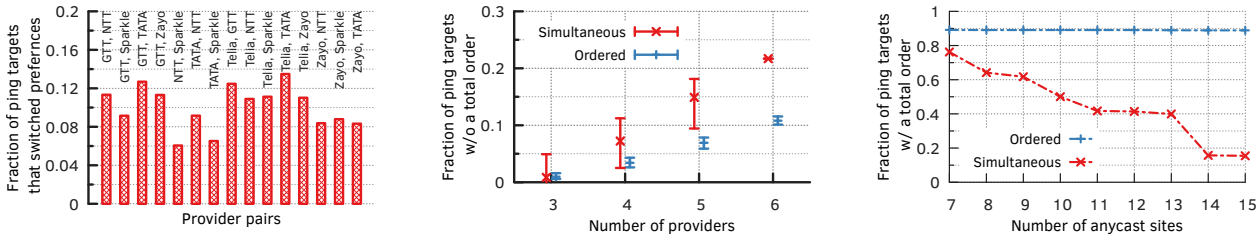
In this section, we answer the first two questions regarding the impact of BGP announcement order and our ability to observe a total preference order. We begin with experiments to assess inter-AS preferences and then repeat the same for intra-AS preferences.

Inter-AS experiments & impact of announcement order. AnyOpt uses pairwise experiments to discover a client network's preference order between two anycast sites. If a network has a total ordering among all sites, AnyOpt uses it to predict its catchment for a given anycast configuration. We take the two-level approach described in §4.3 to discover a network's preference orders for anycast sites on our testbed. For the AS-level preference discovery, we pick two transit providers and run two pairwise comparison experiments. In each experiment, we announce an anycast prefix to a representative site from each provider AS respectively. After BGP stabilizes, we measure each site's catchment and the RTT from a target network to each catchment site as described in §3. We separate the two announcements by six minutes in each experiment, and in the second experiment, we reverse the order of the announcements in the first experiment. As we describe in §4.2, BGP implementations break ties using the arrival order of route advertisements. Therefore, a network's preference may differ across the two experiments.

Naturally, we first investigate how catchments change when we switch the prefix announcement order between two transit providers. Figure 4a shows that around 6% to 14% of ping targets change their catchment sites, suggesting that the arrival order of the BGP announcements breaks the tie between two equally preferred paths. Note that the change of a client network's preference is not due to transient path changes, as 1) we wait long enough for BGP to converge to measure the catchments, and 2) we separate the first and second experiments by two hours and withdraw the prefix announced in the first experiment before we announce it in the second experiment.

Next, we check whether a client network's pairwise preferences can form a total order among the set of transit providers. As a comparison, we also run pairwise experiments without considering the order of BGP announcements. That is, we announce an anycast prefix simultaneously from a representative site in each of the two providers. Our testbed has a total of six transit providers. We use it to emulate an anycast network with three to six providers respectively. For each emulation, we choose a random $X \in [3,6]$ number of transit providers and run pairwise preference discovery experiments among those providers.

Figure 4b shows that as the number of providers increases the fraction of networks that do not have a total order increases. The



(a) (b) (c)

Figure 4: ***(a) A significant fraction of ping targets switch their preferences based on the order in which they receive the anycast prefix announcements. (b) Announcing an anycast prefix simultaneously (in red) from two transit provider ASes leads to more clients without a total preference ordering compared to separating the announcements from the two ASes by six minutes (in blue). (c) Fraction of ping targets with a total ordering remains steady at 85% as the number of sites increases and the announcement order is controlled, but falls drastically otherwise.***

error bars in Figure 4b show the variance among different measurements. Incorporating the order of BGP announcements reduces the fraction of networks without a total ordering by half. When there are six transit providers, if we do not consider the announcement order, 21.7% of networks do not have a total preference order among the providers. In contrast, if we consider the announcement order, this number decreases to 10.8%.

Intra-AS experiments. After AS-level preference discovery, we determine each network's preference orders among the anycast sites within the same transit provider AS. To do so, we choose a transit provider and announce an anycast prefix from any two sites within the transit provider. The site-level catchment is determined by an AS's interior routing mechanism. Therefore, the BGP announcement order should not affect site-level catchment. For a transit provider with N sites, we run $N*(N-1)/2$ pairwise site-level experiments for discovering a network's preference order. For our testbed, each provider has two to four sites. So it takes one to six pairwise experiments to discover a network's preferences per provider.

After the two-level preference measurements, we calculate a network's total preference order among all sites by first ranking the transit providers based on the network's preference for a specific announcement order and then ranking the sites within each provider. We then calculate the fraction of networks that have a total order. Similarly, as a comparison, we also run pairwise site-level preference discovery experiments without considering the BGP announcement order. To do so, we pick two random sites and announce an anycast prefix from these two sites simultaneously. We measure a network's pairwise preferences and compute its total preference order. We vary the number of sites in the experiments to emulate an anycast network of varying sizes.

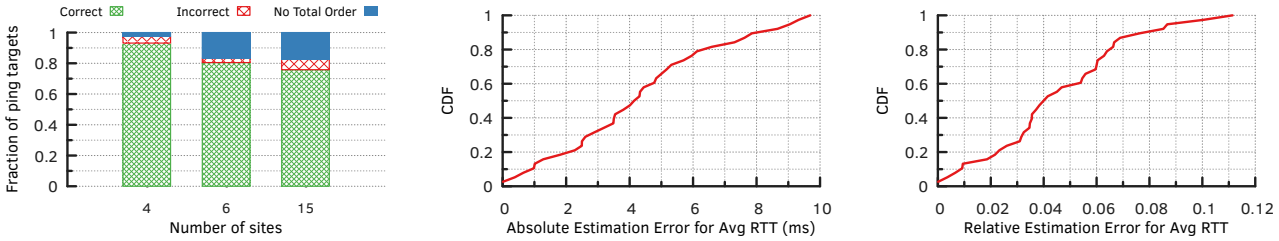
We start with an anycast network with one site in each transit provider connected to our testbed. When we add more sites, the fraction of networks that have a total preference order sharply decreases (as shown in Figure 4c) if we do not consider the impact of BGP announcement orders on route selection. When the number of sites reaches 15, only 15.5% of networks have a total preference order. In contrast, when we consider BGP announcement orders and use the two-level pairwise preference discovery mechanism, 88.9% of networks still exhibit a total preference order, which enables AnyOpt to accurately predict the catchments of an anycast configuration.

5.2 Catchment Prediction

Next, we evaluate whether AnyOpt can accurately predict catchments and the overall latency of an anycast configuration, which answers question (3). We first choose a random subset R from all sites in our anycast testbed. We then use each client network's total preference order (among this subset) under a BGP announcement order for predicting the client network's most preferred site among R and its RTT to its catchment site. We do not predict catchments for client networks that do not exhibit a total order. We then deploy the configuration R under the same BGP announcement order and measure the resulting catchment of each site in R and each target's RTT to its catchment site. We compare the predicted catchments and RTTs with the measured ones to gauge the accuracy of AnyOpt's predictions. We then vary the subset R and repeat the above steps.

Figure 5 summarizes the results. In Figure 5a, we show the results from three experiments. In the first two experiments, we choose four and six transit providers in our anycast testbed and enable a representative site in each provider. In the third experiment, we enable all 15 sites. We measure the fraction of client networks for which we correctly/incorrectly predict their catchment sites and the fraction of networks that do not have a total order. The figure shows that when the number of sites increases, the number of networks that have a total preference order decreases. However, within those networks that have a total order, we can correctly predict the catchment sites more than 93% of the time. There are several reasons that might lead to no total orders, such as multipath routing, uncommon BGP policies, and routing configurations that violate the sufficient conditions for a total order (§4.1).

We plot the CDF of the absolute values of the differences between the predicted average RTT (of all targets) of an anycast configuration and the measured average RTT of the same anycast configuration in Figure 5b. We compute the CDF from 38 random anycast configurations with the number of sites ranging from 1 to 14. Per this figure, the predicted average RTT is within 6ms of the measured RTT for more than 80% of anycast configurations.



(a) *Catchment prediction accuracy* **(b)** *Absolute RTT estimation error* **(c)** *Relative RTT estimation error*

Figure 5: ***Evaluation of catchment prediction accuracy. (a) AnyOpt predicts most catchments correctly when client networks exhibit a total ordering; (b,c) the vast majority of errors in RTT estimates are small.***

Figure 5c shows the relative errors of the predicted average RTT when compared with the measured average RTT for each anycast configuration we choose. For all configurations we tested, the mean predicted average RTT error is less than 4.6%.

Takeaways. These results are encouraging. They suggest that once we obtain a client network's total preference order and the RTTs from each site to each target, we can accurately predict the catchments and the overall RTT of an anycast configuration for our testbed.

5.3 Performance Optimization

In this section, we answer question (4). In addition to predicting the catchments of an anycast configuration, AnyOpt assists in finding a configuration (i.e., the set of sites enabled to announce an anycast prefix) that results in the lowest average RTTs between the anycast sites and the targets. To estimate the extent of potential RTT reductions, we conduct the following experiments. We use offline computations to iterate over as many anycast configurations for our testbed as we could possibly compute within a time bound, which we currently set to six hours. For each configuration, we choose a prefix announcement order that yields the largest fraction of client networks with a total preference order. The computation returns a 12-site configuration out of 15-site testbed. We deploy this AnyOpt-optimized configuration and measure the catchments of each site and the average client latency. We then compare the average RTT of the AnyOpt configuration with two other types of configurations and the default configuration of enabling all 15 sites.

N-Greedy. In these configurations, we enable N sites using a greedy algorithm. We enable the sites in a configuration according to their average unicast RTTs to all client networks. Recall that we measured those RTTs by announcing an anycast prefix from only one site. We choose the top N sites with the lowest average RTTs to the measurement targets, deploy the configuration, and measure its catchments and RTTs. The 12-Greedy configuration has the same number of sites as in the AnyOpt-optimized configuration.

4-Random. To simplify management, network operators may choose to use only a small number of providers and sites. For this scenario, we assume a network chooses two providers and two sites in each provider. We randomly generate three such configurations, deploy them, and measure their catchments and RTTs.

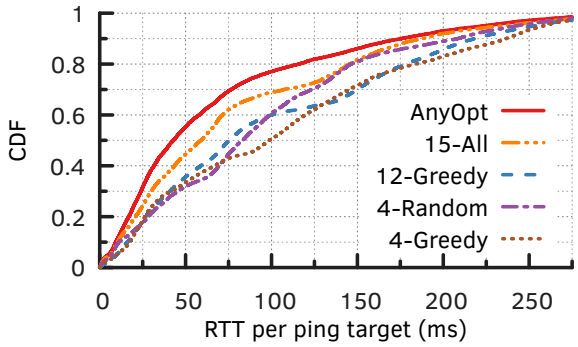


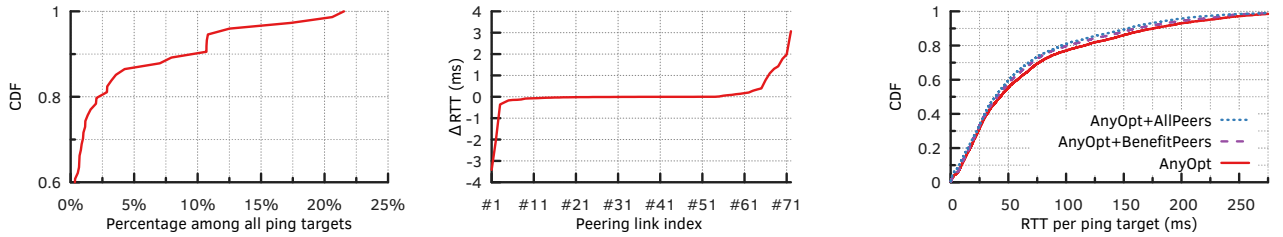
Figure 6: ***AnyOpt-optimized configuration substantially outperforms other approaches in terms of RTT.***

We show CDF of the resulting RTTs to each target network under each scenario in Figure 6. The 4-Random line is the result from the best random four-site configurations we generate. The median RTT for the AnyOpt-optimized configuration (the "AnyOpt" line) is 43ms, while that of the greedy configuration of the same number of sites (the "12-Greedy" line) is 76ms. Put another way, AnyOpt improves the median RTT by 43.4% for the same number of sites. Compared with other configurations, AnyOpt improves the median RTT by 27-59.8%. Although not shown in the figure, the AnyOpt configuration also has a 33ms lower average RTT compared to the greedy configuration with the same number of sites. It has 14–35ms lower average RTT than the other configurations.

Consistent with observations from prior work, we find that the configuration with all 15 sites enabled (the "15-all" line in Figure 6) exhibits worse performance than a smaller AnyOpt configuration with 12 sites. However, the 12-site AnyOpt configuration substantially outperforms the other configurations with fewer or the same number of sites. *This result shows that* ***more sites*** *can lead to* ***substantially better*** *performance when using the right measurements and optimization approach.*

5.4 Incorporating Peering Links

Next, we show how incorporating peering links to the transit-only AnyOpt configuration can impact the average client latency. The AnyOpt testbed includes 104 non-transit peering links. Among them, only 72 peering links can reach some of our ping targets, which could



(a) **(b)** **(c)**

Figure 7: ***Impact of non-transit peers on AnyOpt performance. (a) CDF of a peer AS's catchment when adding a peering link to AnyOpt's transit-only configuration. (b) Mean RTT changes when adding a peering link to AnyOpt's transit-only configuration. (c) CDF of client RTTs after incorporating peering links.***

be due to routing configurations, e.g., a peer may filter our testbed traffic, or a peer's catchment is too small to include any ping target.

To estimate a peer's impact on the average client latency, we enable each peer separately on the AnyOpt-optimized transit-only configuration (as described in §4.4). We then measure the peer's catchment size under this configuration and how the average client RTT has changed. If enabling the peer reduces the average RTT, we deem it a beneficial peer. Figure 7a shows the CDF of a peer's catchment size distribution, and Figure 7b shows how enabling a peer changes the RTT averaged over all ping targets. We rank the peers by the value of average RTT changes they introduce. As can be seen, more than 80% of the peer links on our testbed have a catchment size consisting of fewer than 2.5% ping targets. Only a few peers have noticeable impact on the average RTTs.

We then use the one-pass heuristic to enable the beneficial peers that are likely to reduce the average RTT of the AnyOpt-optimized configuration. We refer to this configuration as AnyOpt+BenefitPeers. We measure the RTT distribution of each ping target under this configuration, and compare it with the configuration of enabling all peers (AnyOpt+AllPeers) and AnyOpt. Per Figure 7c, AnyOpt+AllPeers and AnyOpt+BenefitPeers have similar performance. Both perform slightly better than AnyOpt, but not significantly. Specifically, AnyOpt+BenefitPeers reduces AnyOpt's average RTT from 68 ms to 63ms, while AnyOpt+AllPeers reduces it to 61ms. We note that in our testbed, enabling all peers leads to a configuration with a slightly lower mean RTT than the configuration identified by the one-pass heuristic. This result may not be generally applicable to other anycast networks, and a conservative approach such as the one-pass heuristic, which includes beneficial peers one-by-one, will be useful in situations where enabling all peers worsens the performance of a transit-only configuration.

6 LIMITATIONS AND FUTURE WORK

This work has a few key limitations and leaves open a number of interesting directions for future work. We discuss them below.

Testbed We obtain all experimental results on the anycast testbed we use. Although from our theoretical analysis, we expect that other networks would obtain similar results, this hypothesis is yet to be validated by real-world experiments on other anycast networks.

Large anycast networks. Although we outlined a heuristic approach in §4.3 that uses a client network's RTTs to anycast sites of a large anycast network to discover the network's intra-AS site preferences, we have not yet to test the effectiveness of this approach on other anycast networks.

Settlement-free peers. We have performed experiments to fine-tune a baseline anycast configuration consisting of only advertisements to tier-1 transit providers. We also used a one-pass method to evaluate the peering links we had in our infrastructure. Our evaluation showed that the RTT reduction brought by these peerings is small. While interesting, our findings might be impacted by limited connectivity in our testbed (15 sites and 104 peering links). An open question is how much performance might change if we advertised to settlement-free peers for other larger anycast networks.

Stability analysis. Deploying AnyOpt as a production system would require a longitudinal study to determine how often client total orders change, and by how much. This information would then govern the frequency of the pairwise experiments that would be necessary to keep the total orders up to date. We have only conducted a few experiments in January 2021 to gauge whether the performance of an anycast configuration remains stable. That is, if we deploy the optimal configuration given by AnyOpt, will it remain optimal? We deployed a configuration and measured it weekly in the first three weeks of January 2021. The results are promising, more than 90% of the catchments remain unchanged and the average RTT is also very stable in the three-week duration. It is our future work to study the stability of an optimal configuration in detail.

Other control knobs. A network operator can modify the BGP attributes of an anycast prefix advertisement (e.g., prepend its own AS numbers) to influence catchments. She can also use the BGP poisoning technique [20] to avoid a specific AS hop along the path. It is in our future work to explore how to use these "knobs" for catchment prediction and performance optimization.

Reducing the number of experiments. When the number of transit networks that an anycast network connects to grows larger (e.g., 20 or more), performing a quadratic number of experiments becomes burdensome. It is natural to ask whether the total orders could be learned, or learned approximately, using fewer experiments. One possible future direction to reducing the number of experiments would be to rely on publicly available BGP routing tables to infer

as much about catchments as possible, and then to supplement the information gleaned from these tables with active measurements.

7 RELATED WORK

Measuring IP anycast performance. IP anycast has long been used by Internet services to provide automatic load balancing and latency reduction among service replicas [5, 6, 10]. Most related work focuses on measuring the performance of deployed IP anycast systems, including DNS root servers [16, 21, 23, 25, 26, 28] and CDNs [6, 11, 21]. The main results from these studies are consistent: Global IP anycast fails to route clients to the replicas that provide the lowest latency or to evenly distribute the workload among the replicas [33]. As an example, Li et al. [24] showed that for the D-root name server, only one third of its clients' queries were routed to their geographically closest anycast sites.

Explaining and improving poor performance. Sarat et al. [33] proposed to limit the radius of an anycast prefix announcement to prevent a client from reaching a topologically distant site. However, as a global ISP often has a network spanning a large geographic area, limiting the radius of a BGP announcement cannot prevent a client from reaching a distant site. Ballani et al. [5] hypothesized that the sub-optimal performance of IP anycast is due to BGP's routing behavior. BGP is performance oblivious, and ASes configure their BGP routers to find the "cheapest" rather than the best performing routes. They proposed to host all anycast sites through one tier-1 provider. Li et al. [24] proposed to embed the origin router's geographic location in a BGP announcement to make BGP latency aware. Alzoubi et al. [3] proposed to use a central route controller and MPLS tunnels to direct anycast traffic to specific anycast sites within one ISP, but it is difficult to generalize this approach to large anycast networks that span multiple ISPs, as across-domain MPLS engineering is not well supported by ISPs. Fastroute [14] describes an anycast architecture that combines DNS redirection and anycast routing to manage the workload of a large CDN.

Different from this body of work, AnyOpt aims to predict anycast catchment and enable a service provider to choose an optimal anycast configuration. It can reduce the overall client latency without modifying BGP announcements to embed geographic information. Although we do not explicitly address load-balancing in this work, as we explain in §3 and Appendix B, a network operator can add a load constraint to the optimization problem or predict how load will change by accurately predicting anycast catchment.

Measuring and inferring anycast catchment. Cloudflare's Verfploeter [11, 12] measures the catchment of an anycast site without using a large number of active probes. Verfploeter sends out ICMP requests to hosts with the source addresses of the requests set to an anycast address. A host's ICMP reply will reach its corresponding catchment site. Vries [12] et al. have shown that Verfploeter can accurately map out the catchment of an anycast site, overcoming the limitation of previous work that uses RIPE Atlas [36] to measure anycast catchment. Another body of work [10, 24, 38] uses RIPE Atlas to send active probes to an anycast address, but RIPE Atlas has a skewed geographic distribution. AnyOpt borrows Verfploeter's architecture to map an anycast site's catchment, but enhances the architecture to measure a client's RTT to an anycast site (§3).

Sermpezis and Kotronis [35] proposed to use the inferred AS-level Internet topology for predicting anycast catchment. Their approach cannot, however, accurately predict how ASes break ties among equally preferred routes. In addition, any incorrect inference in the AS topology will exacerbate the inaccuracy of its prediction. As shown in their simulations, when the number of anycast sites increases from two to four sites, the number of nodes with certain inference decreases from 15000 to 6000 and will keep decreasing as the number of anycast sites increases.

In contrast, AnyOpt takes a measure-model-and-optimize approach. It uses carefully designed BGP experiments to discover how ASes choose paths and combine the experimental results with offline computation for anycast performance prediction and optimization. In the future, we plan to investigate whether we can combine AnyOpt with an inference-based method to further reduce the number of BGP experiments required for making accurate predictions.

8 CONCLUSION

In this paper, we introduced AnyOpt and showed how it can be used to minimize the latency and balance the load of an anycast network. The key idea is that, in certain circumstances, the site preferences of each client network exhibit a total order and we can discover the total orders of all client networks using pairwise preference-elicitation experiments that announce an anycast prefix from any two available sites. We prove the sufficient conditions under which a client network exhibits a total order and use a two-level approach consisting of inter-AS experiments followed by intra-AS experiments to reduce the total number of experiments. With the total order in hand for each client network, we can predict the catchment of each anycast site for any particular subset of sites that might advertise. Then by formalizing the problem as an instance of the SPLPO problem, we can find a set of anycast sites that minimize latency while balancing load (i.e., satisfying capacity constraints). Our evaluation using a testbed that has 15 global sites demonstrates the feasibility of our system. AnyOpt can predict catchment areas with small errors using only a quadratic number of experiments, and solving the resulting optimization problem yields tangible reductions in latency. To the best of our knowledge, AnyOpt is the first work that systematically tackles the anycast performance prediction and optimization problem.

9 ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd Jennifer Rexford for their helpful comments, and Haoyu Wang and Shen Zhu for helping with an early draft of this paper. We sincerely thank the network engineering team at Akamai Technologies, especially Aaron Block and Aaron Atac, whose help made this work possible. We thank Kamesh Munagala for help in proving that even approximating the minimum cost of SPLPO is NP-hard. This work was supported in part by the National Science Foundation under awards 1910867, 1763617, 1763742, 1822965, and 1827674, and in part by subcontracts from Akamai Technologies in support of DARPA prime contract HR0011-17-C-0030. Additional support was provided by Microsoft Research Faculty Fellowship 8300751 and AWS Machine Learning Research awards.

REFERENCES

[1] 2016. *BGP Best Path Selection Algorithm.* Retrieved Jun 28, 2021 from https://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

[2] 2020. *Understanding BGP Path Selection.* Retrieved Jun 28, 2021 from https://www.juniper.net/documentation/en_US/junos/topics/reference/general/routing-protocols-address-representation.html

[3] Hussein A. Alzoubi, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van Der Merwe. 2011. A Practical Architecture for an Anycast CDN. *ACM Trans. Web* 5, 4, Article 17 (Oct. 2011), 29 pages. https://doi.org/10.1145/2019643.2019644

[4] Giorgio Ausiello, Pierluigi Crescenzi, Giorgio Gambosi, Viggo Kann, Alberto Marchetti-Spaccamela, and Marco Protasi. 2012. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer Science & Business Media. https://www.springer.com/gp/book/9783540654315

[5] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. 2006. A Measurement-Based Deployment Proposal for IP Anycast. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement* (Rio de Janeriro, Brazil) *(IMC '06)*. Association for Computing Machinery, New York, NY, USA, 231–244. https://doi.org/10.1145/1177080.1177109

[6] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proceedings of the 2015 Internet Measurement Conference* (Tokyo, Japan) *(IMC '15)*. Association for Computing Machinery, New York, NY, USA, 531–537. https://doi.org/10.1145/2815675.2815717

[7] Ignacio Castro, Juan Camilo Cardona, Sergey Gorinsky, and Pierre Francois. 2014. Remote Peering: More Peering without Internet Flattening. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies* (Sydney, Australia) *(CoNEXT '14)*. Association for Computing Machinery, New York, NY, USA, 185–198. https://doi.org/10.1145/2674005.2675013

[8] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. 2015. Characterizing IPv4 Anycast Adoption and Deployment. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies* (Heidelberg, Germany) *(CoNEXT '15)*. Association for Computing Machinery, New York, NY, USA, Article 16, 13 pages. https://doi.org/10.1145/2716281.2836101

[9] Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. 1983. *The uncapicitated facility location problem.* Technical Report. Cornell University Operations Research and Industrial Engineering. https://hdl.handle.net/1813/8491

[10] Ricardo de Oliveira Schmidt, John Heidemann, and Jan Harm Kuipers. 2017. Anycast Latency: How Many Sites Are Enough?. In *Passive and Active Measurement*, Mohamed Ali Kaafar, Steve Uhlig, and Johanna Amann (Eds.). Springer International Publishing, Cham, 188–200. https://link.springer.com/chapter/10.1007/978-3-319-54328-4_14

[11] Wouter B. de Vries, Salmān Aljammāz, and Roland van Rijswijk-Deij. 2020. Global-Scale Anycast Network Management with Verfploeter. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 1–9. https://doi.org/10.1109/NOMS47738.2020.9110449

[12] Wouter B. de Vries, Ricardo de O. Schmidt, Wes Hardaker, John Heidemann, Pieter-Tjerk de Boer, and Aiko Pras. 2017. Broad and Load-Aware Anycast Mapping with Verfploeter. In *Proceedings of the 2017 Internet Measurement Conference* (London, United Kingdom) *(IMC '17)*. Association for Computing Machinery, New York, NY, USA, 477–488. https://doi.org/10.1145/3131365.3131371

[13] Dino Farinacci, Tony Li, Stanley P. Hanks, David Meyer, and Paul S. Traina. 2000. *Generic Routing Encapsulation (GRE).* RFC 2784. RFC Editor. 1–9 pages. https://www.rfc-editor.org/rfc/rfc2784.txt

[14] 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 *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 381–394. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/flavel

[15] Lixin Gao and Jennifer Rexford. 2001. Stable Internet Routing without Global Coordination. *IEEE/ACM Trans. Netw.* 9, 6 (Dec. 2001), 681–692. https://doi.org/10.1109/90.974523

[16] Danilo Giordano, Danilo Cicalese, A. Finamore, M. Mellia, M. Munafo, Diana Joumblatt, and D. Rossi. 2016. A First Characterization of Anycast Traffic from Passive Traces. In *IFIP workshop on Traffic Monitoring and Analysis (TMA)*. ouvain La Neuve, Belgium, 30–38. https://hal-imt.archives-ouvertes.fr/hal-01383092

[17] Vasileios Giotsas, George Nomikos, Vasileios Kotronis, Pavlos Sermpezis, Petros Gigis, Lefteris Manassakis, Christoph Dietzel, Stavros Konstantaras, and Xenofontas Dimitropoulos. 2021. O Peer, Where Art Thou? Uncovering Remote Peering Interconnections at IXPs. *IEEE/ACM Transactions on Networking* 29, 1 (2021), 1–16. https://doi.org/10.1109/TNET.2020.3025945

[18] Krishna P. Gummadi, Stefan Saroiu, and Steven D. Gribble. 2002. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurment* (Marseille, France) *(IMW '02)*. Association for Computing Machinery, New York, NY, USA, 5–18. https://doi.org/10.1145/637201.637203

[19] Pierre Hanjoul and Dominique Peeters. 1987. A facility location problem with clients' preference orderings. *Regional Science and Urban Economics* 17, 3 (1987), 451–473. https://doi.org/10.1016/0166-0462(87)90011-1

[20] Ethan Katz-Bassett, Colin Scott, David R. Choffnes, Ítalo Cunha, Vytautas Valancius, Nick Feamster, Harsha V. Madhyastha, Thomas Anderson, and Arvind Krishnamurthy. 2012. LIFEGUARD: Practical Repair of Persistent Route Failures. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (Helsinki, Finland) *(SIGCOMM '12)*. Association for Computing Machinery, New York, NY, USA, 395–406. https://doi.org/10.1145/2342356.2342435

[21] Thomas Koch, Ke Li, Calvin Ardi, Ethan Katz-Bassett, Matt Calder, and John Heidemann. 2021. Anycast in Context: A Tale of Two Systems. In *Proceedings of the 2021 Conference of the ACM Special Interest Group on Data Communication* (Virtual Event) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 20. https://doi.org/10.1145/3452296.3472891

[22] F Thomson Leighton, Ravi Sundaram, Matthew Levine, and Adrian Soviani. 2007. Method for generating a network map. US Patent 7,251,688.

[23] Matthew Lentz, Dave Levin, Jason Castonguay, Neil Spring, and Bobby Bhattacharjee. 2013. D-Mystifying the D-Root Address Change. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (Barcelona, Spain) *(IMC '13)*. Association for Computing Machinery, New York, NY, USA, 57–62. https://doi.org/10.1145/2504730.2504772

[24] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. 2018. Internet Anycast: Performance, Problems, & Potential. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication* (Budapest, Hungary) *(SIGCOMM '18)*. Association for Computing Machinery, New York, NY, USA, 59–73. https://doi.org/10.1145/3230543.3230547

[25] Jinjin Liang, Jian Jiang, Haixin Duan, Kang Li, and Jianping Wu. 2013. Measuring Query Latency of Top Level DNS Servers. In *Passive and Active Measurement*, Matthew Roughan and Rocky Chang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 145–154. https://link.springer.com/chapter/10.1007/978-3-642-36516-4_15

[26] Ziqian Liu, Bradley Huffaker, Marina Fomenkov, Nevil Brownlee, and kc claffy. 2007. Two Days in the Life of the DNS Anycast Root Servers. In *Passive and Active Network Measurement*, Steve Uhlig, Konstantina Papagiannaki, and Olivier Bonaventure (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 125–134. https://link.springer.com/chapter/10.1007/978-3-540-71617-4_13

[27] Chris Metz. 2002. IP anycast point-to-(any) point communication. *IEEE Internet Computing* 6, 2 (2002), 94–98. https://doi.org/10.1109/4236.991450

[28] Giovane C.M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. 2016. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *Proceedings of the 2016 Internet Measurement Conference* (Santa Monica, California, USA) *(IMC '16)*. Association for Computing Machinery, New York, NY, USA, 255–270. https://doi.org/10.1145/2987443.2987446

[29] NTT Labs. 2020. BGP implemented in the Go Programming Language. https://github.com/osrg/gobgp.

[30] Craig Partridge, Trevor Mendez, and Walter Milliken. 1993. *Host Anycasting Service.* RFC 1546. RFC Editor. https://www.rfc-editor.org/rfc/rfc1546.txt

[31] Jon Postel. 1981. *Internet Control Message Protocol.* RFC 777. RFC Editor. 1–14 pages. https://www.rfc-editor.org/rfc/rfc777.txt

[32] Yakov Rekhter, Tony Li, and Susan Hares. 2006. *A Border Gateway Protocol 4 (BGP-4).* RFC 4271. RFC Editor. 1–103 pages. https://www.rfc-editor.org/rfc/rfc4271.txt

[33] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. 2005. On the Use of Anycast in DNS. In *Proceedings of the 2005 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems* (Banff, Alberta, Canada) *(SIGMETRICS '05)*. Association for Computing Machinery, New York, NY, USA, 394–395. https://doi.org/10.1145/1064212.1064271

[34] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. 2020. Akamai DNS: Providing Authoritative Answers to the World's Queries. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication* (Virtual Event, USA) *(SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 465–478. https://doi.org/10.1145/3387514.3405881

[35] Pavlos Sermpezis and Vasileios Kotronis. 2019. Inferring Catchment in Internet Routing. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 2, Article 30 (June 2019), 31 pages. https://doi.org/10.1145/3341617.3326145

[36] RIPE NCC Staff. 2015. RIPE Atlas: A global internet measurement network. *Internet Protocol Journal* 18, 3 (2015).

[37] Akamai Technologies. 2020. *Prolexic Routed.* Retrieved Jan 28, 2021 from https://www.akamai.com/us/en/multimedia/documents/product-brief/prolexic-routed-product-brief.pdf

[38] Lan Wei and John Heidemann. 2017. Does anycast hang up on you?. In *2017 Network Traffic Measurement and Analysis Conference (TMA)*. 1–9. https://doi.org/10.23919/TMA.2017.8002905
[39] Einav Yoav. 2019. *Amazon found every 100ms of latency cost them 1% in sales.* Retrieved Jan 28, 2021 from https://www.gigaspaces.com/blog/amazon-found-every-100ms-of-latency-cost-them-1-in-sales/

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A USING PAIRWISE MEASUREMENTS TO PREDICT CLIENT PREFERENCE

AnyOpt relies on pairwise measurements to predict which site is chosen by a client from among the set of sites that announce a given prefix. We now theoretically explore the following questions. Under what conditions are pairwise measurements guaranteed to be *consistent* with a total ordering of the sites? That is, the outcomes from the pairwise measurements are guaranteed to not to contain a cycle. If a consistent total ordering exists, when is that total ordering guaranteed to be *predictive*? That is, the total ordering accurately predicts the site chosen by the client for *every* subset of possible sites announcing the prefix. Finally, are there situations when pairwise measurements are not consistent and/or not predictive?

Whether or not a set of pairwise measurements yield a predictive total order depends on the policies executed by each router. When a router receives multiple route announcements for a prefix in its incoming links, it uses a *selection policy* to choose one of these announcements and further uses an *export policy* to decide the outgoing links on which the selected announcement must be sent. We will assume that the selection and export policies are compliant with the BGP model of Gao-Rexford criterion[15]. Gao-Rexford states that a router selects route announcements from a customer first, peer next, and provider last. For its export policy, the model states that a route learned from a customer is sent to all outgoing links, a route learned from a peer is sent to its customers, and a route learned from a provider is sent to its customers.

A link is said to be empty if no announcement is transmitted over that link. The following hold.

Lemma 1. *Let S and S' be two subsets of the incoming links of a router R such that $S \subset S'$. Let E(S) (resp. E(S')) be the event that announcements for the prefix are received by R in exactly the links in S (resp., S'). For any Gao-Rexford compliant policy, the following statements hold.*

(1) If an outgoing link l of R is non-empty in E(S), then l is non-empty in E(S'). This statement says that if a router R receives route announcements from more incoming links, it cannot shrink the set of outgoing links it exports to.
(2) If an announcement received via an incoming link $i \in S$ was not sent over an outgoing link l in E(S), the announcement received via i is also not sent over that link l in E(S'). This statement says that any additional announcement received from a neighbor cannot increase a router R's preference to an announcement received from a link i.

Proof. The first statement is true since if the link l has an announcement sent in event $E(S)$, the same or a more preferred announcement will be sent in event $E(S')$. The second statement is true since if the announcement from $i \in S$ is not sent in $E(S)$, then it will not be sent in event $E(S')$ when strictly more choices are presented to the router. □

A.1 Local Preference Model

We now present a simple model called the local preference model for inter-domain routing that obeys the Gao-Rexford criterion. The selection policy of a router in this model uses a total preference order of its incoming links to select the announcement in the most preferred link for re-transmission. The total preference order must be determined "locally" in that it is oblivious to the source or path taken by the announcement prior to reaching the router. Note that the Gao-Rexford criterion of choosing route announcements in the preference order of customer, peer, and producer is consistent with a local preference model as long as ties (say, between two customers) are broken using a source-oblivous metric (say, link id). BGP in practice is not source-oblivous, and we extend this model in § A.2.

Lemma 2. *In the local preference model, let A and B be two anycast sites. When A and B are compared pairwise, let A be the "winner" and B be the "loser". When additional sites are turned on, B will continue to be a "loser".*

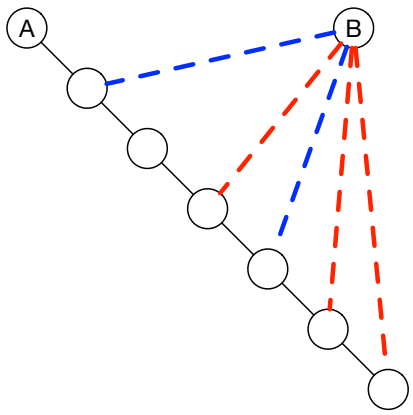


Figure 8: In the pairwise comparison of A and B, the winning path π from site A (in black) is compared to paths π' (in blue) from B, but not to additional paths π'' (in red) from B, which don't reach pi.

Proof. Suppose $S = \{A,B\}$ are the only two sites that are turned on. Figure 8 shows the winning path π of A and the paths π' where the announcements from B meet that winning path π and lose to A. The figure also shows other "potential" paths π'' where an announcement from B could have met the winning path π, but did not, because B was not propagated beyond a certain router on that path.

Now, suppose we turn on more sites, i.e., we turn on all sites in S' such that $S \subset S'$. We claim that B cannot win in event $E(S')$ when all sites in S' are turned on. Note that all links in the winning path π of A are non-empty in $E(S)$, since (by definition) those links carry the announcement from A. Using the first statement of Lemma 1, we conclude that that all links in π are non-empty in $E(S')$. Consider a path π' of B that meets π at some router R. Since the announcement in π won at router R over the announcement in π', the same must happen in $E(S')$, since we assume that the selection policy at R is oblivous to the actual source of the announcements on either path. Thus, B cannot win using path π'. Likewise, using the second

statement of Lemma 1, B cannot be propagated on a potential path in $E(S')$ since it was not propogated on that path in $E(S)$. Thus, B cannot win using a potential path π'', Since B cannot win on a path or a potential path in $E(S')$, we conclude that B cannot win in $E(S')$ and is hence a loser. □

Theorem A.1. *In the local preference model, the following hold. (i) Pairwise site comparisons are always consistent with a total ordering, i.e., no cycles are formed. (ii) The total ordering predicts the winner for any subset of sites.*

Proof. First, we show the existence of a compatible total ordering by showing that a cycle cannot exist. For contradiction, suppose a cycle exists such that $A_1 < A_2 < \ldots < A_k$ and $A_k < A_1$. Using the pairwise comparison $A_1 < A_2$ and invoking Lemma 2, we know that A_2 is a loser when all sites in $S' = \{A_1, A_2, \ldots, A_k\}$ are turned on. Likewise, we can show that $A_i, 1 \le i \le k$, are losers in $E(S')$. But this is not possible as there has to be a winner in $E(S')$. Hence there can be no cycle.

To show that the total ordering always predicts the winner for any set of sites, consider any set $S' = (A_1, A_2, \ldots, A_k)$ such that $A_1 < A_2 < \ldots < A_k$ according to the total ordering. Note that we know that $A_i < A_{i+1}$ through pairwise comparisons, since all pairwise comparisons were performed. Using Lemma 1, we can show that when $S' = (A_1, A_2, \ldots, A_k)$ are turned on, $A_2, \ldots, A_k$ must be losers. So A_1 must be the winner in $E(S')$, as one winner should exist. □

A.2 Shortest-Path Model

The local preference model of Section A.1 assumes a total preference order among incoming links, but in practice some incoming links may be equally preferred. Furthermore, the local preference model does not capture the use of other information such as AS path length to differentiate among paths. In an attempt to capture the impact of AS path length in the route selection process, we define the shortest path model. In this model, each router considers path length first and then breaks ties based on neighbor id. The model is consistent with Gao-Rexford policy routing when, e.g., all of the paths received at a node fall into the same class (e.g., all are from providers), which occurs if all sites peer only with tier-1 networks.

Lemma 3. *In the shortest path model, let A and B be two anycast sites. When A and B are compared pairwise, let A be the "winner" and B be the "loser". When additional sites are turned on, B will continue to be a "loser".*

Proof. The proof is similar to Lemma 2, but we make a slightly different argument. First, we observe that Lemma 1 still holds, since the shortest-path model is Gao-Rexford compliant. Statement 1 of this lemma states that if a link is non-empty in event $E(S)$, then it is non-empty in event $E(S')$, when $S \subset S'$. In the shortest path model, we can make an additional claim that the path length of the announcement sent over the link in $E(S')$ is at most the path length of the announcement sent over that link in $E(S)$, because the routers choose the announcement with the shortest path length.

Now, consider a pairwise comparison of sites $S = \{A, B\}$ with a winning path π from A that was compared to paths π' of B during the pairwise comparison, and additional paths π'' from B that were not compared to π as shown in Figure 8. Suppose we now turn on more sites S' such that $S \subset S'$. Let R be a router where path π and a path π' meet. B cannot win using a path π' in event $E(S')$ since it lost to A on path π in event $E(S)$ and the announcement entering R in π in event $E(S')$ has path length smaller or equal to the path length of the announcement on that link in $E(S)$. Note that if both paths have the same length, tie breaking according to router id will continue to prefer π. Thus, B cannot win using a path π'. Likewise, B cannot win using an additional path π'' since, by statement 2 of Lemma 1, these paths cannot appear at π when more sites are turned on. Thus, B cannot win in $E(S')$ and is hence a loser. □

Theorem A.2. *In the shortest path model, the following hold. (i) Pairwise site comparisons are always consistent with a total ordering, i.e., no cycles are formed. (ii) The total ordering predicts the winner for any subset of sites.*

Proof. The proof invokes Lemma 3 and is identical to the proof of Theorem A.1. □

B THE OPTIMIZATION MODEL

We formally introduce the SPLPO optimization model and show how we can extend it to meet different practical constraints. Let S denote the set of available anycast sites and s_i denote a site in this set. We denote the set of hosts as H and use h_k to denote a host in H. We define a preference order operator for each host h_k over the set of anycast sites S. If in the pairwise experiment, the host h_k prefers the site s_i to s_j, we denote it as $s_i \succ_{h_k} s_j$, where $\succ_{h_k}$ is the preference order operator. Let $rtt_(h_k, s_i)$ denote the round trip latency from a host h_k to a site s_i. We use the boolean variable x_{h_k,s_i} to denote a client's catchment. When a client's catchment site is s_i, $x_{h_k,s_i} = 1$ and vice versa. Similarly, we use a boolean variable y_i to denote whether a network operator enables a site s_i to announce an anycast prefix: $y_{s_i} = 1$ means site s_i is enabled and vice versa.

We now can formulate the anycast performance optimization problem as the following problem:

$$min \sum_{k=1}^{|H|} \sum_{i=1}^{|S|} rtt_{h_k,s_i} \cdot x_{h_k,s_i} \tag{1}$$

$$\sum_{i=1}^{|S|} x_{h_k,s_i} = 1, k = 1,2,\ldots,|H| \tag{2}$$

$$0 \le x_{h_k,s_i} \le y_{s_i}, k = 1,2,\ldots,|H|, i = 1,2,\ldots,|S| \tag{3}$$

$$x_{h_k,s_i} = 0 \text{ or } 1, h_k = 1,2,\ldots,|H|, \; i = 1,2,\ldots,|S| \tag{4}$$

$$y_{s_i} = 0 \text{ or } 1, i = 1,2,\ldots,|S| \tag{5}$$

$$\sum_{s_i \succ_{h_k} s_j} x_{h_k,s_i,} \ge y_{s_j} \tag{6}$$

Equation (3) specifies that a host h cannot choose a site s_i if it is not enabled. Equation (6) ensures that a client chooses is its most preferred site among all enabled sites. We can extend this model in several ways. For instance, we can weigh each host's RTT with its workload to minimize the workload-weighted average latency. We can also add a load constraint at each site to balance the workloads among multiple sites. That is, if L_{s_i} denotes the maximum load a site s_i can absorb and $l(h)$ represents the load from each host, we can add the following contraint to the above formulation:

$$\sum_{k=1}^{|H|} l(h_k) \cdot x_{h_k,i} \leq L_{s_i} \tag{7}$$

B.1 Reduction of Dominating Set to SPLPO

We now provide a reduction from the NP-hard problem Dominating Set to SPLPO. The reduction implies not only that SPLPO is NP-hard, but that even approximating the minimum cost for SPLPO is NP-hard.

Theorem B.1. *Given an instance of the Dominating Set problem consisting of a graph $G = (V,E)$ and an integer K, in linear time it is possible to generate an instance of the SPLPO problem such that if there exists a dominating set of size K for G, then there is a zero cost solution to the SPLPO instance using $K+1$ sites, but if there is no dominating set of size K, then the cost of any solution to the SPLPO instance using $K+1$ sites is infinite.*

Proof. Given a graph $G = (V,E)$, we create an instance of SPLPO as follows. Make each vertex v a client as well as a site. Call the client c_v, and site s_v, with the distance between c_v and s_v equal to zero. Infinitely far away, create a single site s^* with its own client c^* at distance zero. If vertex v has neighbors $N(v)$, in SPLPO, c_v prefers s_v, then s_w for $w \in N(v)$ in some order, then s^*, and then the rest of the sites in some order. Client c^* prefers s^* first. If the minimum dominating set has size K, then a zero cost solution to SPLPO must use $K+1$ sites. Any solution with distance cost zero must open s^* and a dominating set of G. This is because for c^* to have distance cost zero, s^* must be opened. But once s^* is opened, for each v, one of s_v or s_w for $w \in N(v)$ must be opened, for otherwise c_v will map to s^* and pay infinite cost. Therefore, if we set the number of sites for SPLPO to $K+1$, determining if the optimal cost is zero or infinity is exactly the same problem as determining whether G has a dominating set of size K. □