Anycast Agility: Network Playbooks to Fight DDoS

A S M Rizvi* *USC/ISI*
Leandro Bertholdo* *University of Twente*
João Ceron *SIDN Labs*
John Heidemann *USC/ISI*

Abstract

IP anycast is used for services such as DNS and Content Delivery Networks (CDN) to provide the capacity to handle Distributed Denial-of-Service (DDoS) attacks. During a DDoS attack service operators redistribute traffic between anycast sites to take advantage of sites with unused or greater capacity. Depending on site traffic and attack size, operators may instead concentrate attackers in a few sites to preserve operation in others. Operators use these actions during attacks, but how to do so has not been described systematically or publicly. This paper describes several methods to use BGP to shift traffic when under DDoS, and shows that a *response playbook* can provide a menu of responses that are options during an attack. To choose an appropriate response from this playbook, we also describe a new method to estimate true attack size, even though the operator's view during the attack is incomplete. Finally, operator choices are constrained by distributed routing policies, and not all are helpful. We explore how specific anycast deployment can constrain options in this playbook, and are the first to measure how generally applicable they are across multiple anycast networks.

1 Introduction

Anycast routing is used by services like DNS or CDN where multiple sites announce the same prefix from geographically distributed locations. Defined in 1993 [49] anycast was widely deployed by DNS roots in the early-2000s [4, 29, 64], and today it is used by many DNS providers and Content Delivery Networks [16, 17, 24, 26, 80].

In IP anycast, BGP routes each network to a particular anycast site, dividing the world into *catchments*. BGP usually associates networks with nearby anycast sites, providing generally good latency [62]. Anycast also helps during Distributed-Denial-of-Services (DDoS) attacks, with each site adds to the aggregate capacity at lower cost than a single very large site. Each site is independent, so should DDoS overwhelm one site, sites that are not overloaded are unaffected.

DDoS attacks are getting larger and more common. Different root servers and anycast services frequently report DDoS events [18,42,47,48]. Different automated tools make it easier to generate attacks [81], and some offer DDoS-as-a-Service, allowing attacks from unsophisticated users for as little as US$10 [68]. DDoS intensity is still growing, with the 2020 CLDAP attack exceeding 2.3 Tb/s in size [66], and the 2021 VoIP.ms attack lasting for over 5 days [50, 65]. The reservoir of attack sources grow with millions of Internet-of-Things devices whose vulnerabilities fuel botnets [35].

Operators depend on anycast during DDoS attacks to provide capacity to handle the attack and to isolate attackers in catchments. Service operators would like to adapt to an ongoing attack, perhaps shifting load from overloaded sites to other sites with excess capacity. Prior studies of DDoS events have shown that operators take these actions but suggested that the best action to take depends on attack size and location compared to anycast site capacity [45]. While prior work suggested countermeasures, and we know that operators alter routing during attacks, to date there has been only limited evaluation of how routing choices change traffic [4,27,36,52]. Only very recent work examined path poisoning to avoid congested paths [70]; there is no specific public guidance on how to use routing during an attack.

The goal of this paper is to guide defenders in traffic engineering (TE) to balance traffic across anycast during DDoS.

Our first contribution is a system with novel mechanism to estimate true attack rate and plan responses. First, we propose a new mechanism to *estimate the true offered load*, even when loss happens upstream of the defender. Estimating the relative load on each site (§3.3) is the first step of defense, so that the defender can match load to the capacities of different sites, or decide that some sites should absorb as much of the attack as possible. Second, we develop a *BGP playbook*: a guide that allows operators to anticipate how TE actions rebalance load across a multi-site anycast system. Together, these two elements provide a system that can automate response to DDoS attacks by adjusting anycast routing according to the playbook, or recommend actions to a human operator.

The second contribution is to understand how well routing options for multi-hop TE work: AS prepending, community strings and path poisoning. While well known, it is not widely understood how *available* and *effective* these mechanisms are. In §6 we show that while AS prepending is available almost anywhere, community strings and path poisoning support varies widely. We also show that their effectiveness varies greatly, in part because today's "flatter Internet" [15] means AS prepending often shifts either nearly all or nearly no traffic. Community strings provide finer granularity control, but we

*Shared first author

show their support is uneven. Path poisoning may provide control multiple hops away, but like community strings it is often filtered, particularly for Tier-1 ASes. When these factors combine with the interplay between multiple sites and an anycast system, a BGP playbook is important to guide defenders. Since the effects of TE are often specific to the peers and locations of a particular anycast deployment, we explore how sensitive our results are to different locations and numbers of anycast sites (§7).

Our final contribution is to demonstrate successful defenses in practice. We replay real-world attacks in a testbed and show TE can defend (§8). Of course no single defense can protect against all attacks, these examples show our approach provides a successful defense to many volumetric and polymorphic DDoS attacks. They show that our algorithm and process contributions (attack size estimation and playbook construction) have practical application.

Our work uses publicly available datasets. Datasets for the input and results from our experiments are available at no charge. Because our data concerns services but not individuals, we see no privacy concerns.

2 Related Work

Anycast routing has been studied for a long time from the perspective of routing, performance, and DDoS-prevention.

BGP to steer traffic: Prior work showed BGP is effective to steer traffic to balance load on links [8, 27, 53]. However, Ballani et al. showed that anycast requires planning and care for effective load balancing [4]. Others proposed to manipulate BGP based on packet loss, latency and jitter [46, 52]. We build on Ballani's recommendation to plan anycast, proposing a BGP playbook, and studying how well it can work.

Chang *et al.* [14] suggested using BGP Communities for traffic engineering [10, 13, 74]. Recent work has examined BGP communities for blackhole routing in IXPs and ISPs [21, 28]. Smith and Glenn examined path poisoning to address link congestion [70]. While each of these are important options in routing for defense, we show a system that guides the operator to select between them. A system with multiple choices is necessary because no single method works against all attacks. For example, we show path poisoning does not work when we poison a Tier-1 AS.

Anycast performance: Most anycast research focused on efficient delivery and stability [11, 39, 40, 59, 79]. Later studies explicitly investigate the proximity of the clients [4, 11, 39].

Some studies try to improve anycast through topology changes [44, 62]. Anycast services for DDoS is already used in commercial solutions *e.g.,* Amazon [63], Akamai [75] and AT&T [72]. However, none of them address how to use routing manipulations as a DDoS defense mechanism.

Anycast catchment control as a DDoS mitigation tool: To our knowledge, the idea of handling DDoS attacks by absorbing or shifting load across anycast sites was first published in 2016 [45]. Kuipers *et al.* [36] refined that work, defining

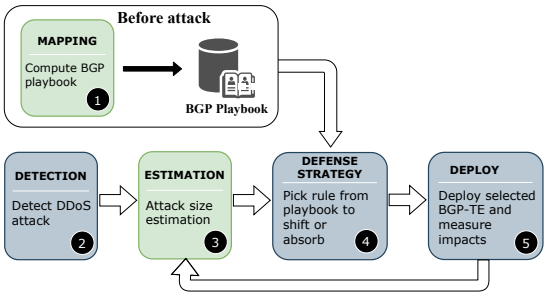


Figure 1: Overview of the decision process.

the traffic shifting approaches that we review in §3.4 and explore through experiment. We develop the idea of a BGP playbook to guide responses, and describe a new approach to estimate attack size, and finally show that responses can be effective with real-world events.

Commercial and automated solutions: Most published commercial anti-DDoS solutions use routing to steer traffic towards a mitigation infrastructure [22]. Sometimes there is a requirement for all the sites to be connected through a private backbone to support traffic analysis [63]. Another defense uses BGP to divert all traffic to a scrubbing center, then tunnels good traffic to the destination [69]. Other methods use DNS manipulation [12], or anycast proxies [30] which cannot be used in DNS anycast deployments itself. Rather than outsourcing the problem, we explore how one can defend it. Other automated defenses include responsive resource management [25], client-server reassignment [34], and filtering approaches [58]. Our method uses TE approaches to efficiently use available resources in anycast.

3 Mechanisms to Defend Against DDoS

In this section we describe our BGP mitigation process; how we pre-compute a BGP playbook, estimate the attack size and select a TE response.

3.1 Overview and Decision Support

In Figure 1 we show how defense against DDoS works. Defense against a DDoS begins with detection ②, then defenders plan a defense ④, carry it out ⑤, and repeat this process until the attack is mitigated or it ends (bottom cycle in Figure 1). Detecting the attack is straightforward, since large attacks affect system performance. The challenge is selecting the best response and quickly iterating.

We bring two new components to attack response (colored light green in Figure 1): mapping before the attack, and estimating attack size when the attack begins. Mapping ① (discussed in §3.2) provides the defender with a *playbook* of planned responses and the information about how they will change the traffic mix across their anycast system. Size estimation ③ (discussed in §3.3) allows the defender to determine how much traffic should be moved and select a promising response from the playbook. Together, these tools help to

understand not only how to reduce traffic at a given site, but also the sites where that traffic will go.

These components come together in our automated response system (§3.4) that iterates between measurement and attack size estimation, defense selection, then deployment. Defense uses the playbook built during mapping; we provide an example playbook in §6.4. We show how these defenses operate in testbed experiments in §8.

Our system is designed for services that operate with a fixed amount of infrastructure on specific anycast IP addresses and do not employ a third-party scrubbing service. Operators of CDNs with multiple anycast services, DNS redirection, or scrubbing services may use our approach, but also have those other tools. However, many operators cannot or prefer not to use scrubbing and DNS redirection: all operators of single-IP services (all DNS root servers), many ccTLDs who value national autonomy, and scrubbing services themselves. Our approach defends against volumetric attacks where we have spare capacities in other sites. Since DDoS causes unavailability of services, suboptimal site selection during an attack is not a concern.

3.2 Measurement: Mapping Anycast

We map the catchments of anycast service before an attack so that the defender can make an informed choice quickly during an attack, building a BGP playbook (§6.4).

To map anycast catchments we used Verfploeter [20]. As an active prober (ICMP echo request), Verfploeter observes the responses of all ping-responsive IPv4 /24s and maps which anycast site receives the responses. We provide a detailed description of anycast and Verfploeter in Appendix A. Since mapping happens before the attack, mapping speed is not an issue.

Alternatively, we can map traffic by observing which customers are seen at each site over time, or measuring from distributed vantage points such as RIPE Atlas [3, 73]. (Operators may already collect this information for optimization.)

Mapping should consider not only the current catchments but also *potential* shifts we might make during the attack. This full mapping is easy to do with Verfploeter, which can be continuously running in an adjacent BGP prefix to map the possible shifts. This mapping process is important to anticipate how traffic may shift. We will show later that BGP control is limited by the granularity of routing policy (§6) and by the deployment of the anycast sites (§7).

A challenge in pre-computed maps with routing alternatives is that routing is influenced by all ASes. Thus, the maps may shift over time due to changes in the routing policies of other ASes. Fortunately, prior work shows that anycast catchments are relatively slow to change [79]. We also show that our BGP playbook is stable over time (§6.4 and Appendix E).

3.3 Estimation of the Attack Size

After the detection of an attack, the first step in DDoS defense is to estimate the attack size, so we can then select a defense strategy of how much traffic to shift. Our goal is to measure *offered load*, the traffic that is sent to (offered to) each site. During DDoS offered load balloons with a mix of attack and legitimate traffic, and loss upstream of the service means we cannot directly observe true offered load. We later evaluate our approach with real-world DDoS events (§4).

Idea: Our insight is that we can *estimate* true offered load based on changes in some known traffic that actually does arrive at the service, even when there is upstream loss.

To know how much offered load actually arrives at the service, we need to estimate some fraction of legitimate traffic. We can then observe how much this traffic drops during the attack, inferring upstream loss. Unfortunately, there is no general way to determine all legitimate traffic, since legitimate senders change their traffic rates, and attackers often make their traffic legitimate-appearing. Our goal is to reliable estimate *some* specific legitimate traffic; we describe several sources next.

Traffic sources: There are several possible sources of known legitimate traffic—we consider known measurement traffic and regular traffic sources that are heavy hitters [5].

For DNS, our demonstration application, RIPE Atlas provides a regular source of known-good traffic, sent from many places. RIPE makes continuous traffic from around 10k publicly available vantage points [55]. Each RIPE vantage point queries every 240 s, and there is enough traffic (about 2500 queries/minute) to provide a good estimate of offered load. (Although RIPE Atlas is specific to DNS, other commercial services often have similar types of known monitoring traffic.)

To find the known-good traffic at each site, we use the catchments of RIPE vantage points with pre-deployed RIPE DNS CHAOS queries (one exists for each root DNS IP, such as measurement ID 11309 for A-root). We can also use Verfploeter or captured traces in the anycast sites. An advantage of using RIPE traffic is that it does not place any new load on the service.

Heavy hitters can provide an additional source of known-good traffic. Many services have a few consistently large-volume users with regular traffic patterns, and while they vary over time, many are often stable. For DNS, we find that most heavy hitters have a strong diurnal variation in rate; we model them with TBATS (Trigonometric seasonality, Box-Cox transformation, ARMA errors, Trend and Seasonal) [19] to factor out such known variation. While an adversary could spoof heavy hitters, that requires a large and ongoing investment to succeed.

Estimation: Our goal is to estimate offered load, $T_{offered}$. We can measure the observed traffic rate, $T_{observed}$, at the access link. We define α as the *access fraction*—the fraction of traffic that is not dropped. Therefore $T_{observed} = \alpha \cdot T_{offered}$.

To estimate the access fraction (α), we observe that known good traffic has the same loss on incoming links as does other good traffic and attack traffic. We estimate the known

traffic rate (from RIPE Atlas measurement traffic, or from heavy hitters, or both), as T_{known}. Then $\alpha \cdot T_{known,offered} = T_{known,observed}$, and our estimate of offered load is $\hat{T}_{offered} = T_{observed} \cdot T_{known,offered} / T_{known,observed}$.

3.4 Traffic Engineering as a Defense Strategy

With knowledge of the offered load, the defender can select an overall defense strategy that will drive traffic engineering decisions. The defender first must determine if the attack exceeds overall capacity or not.

For attacks that exceed overall capacity, the defender's goal is to preserve successful service at some sites, while allowing other sites to operate in degraded mode as *absorbers* [45]. The defender may also choose to shift traffic away from some degraded sites to ease their pain. Unloading the overloaded sites is recognized as *breakwaters* [36].

For moderate-size attacks, the defender should try to serve all traffic, *rebalancing* to shift traffic from overloaded sites to less busy sites. In heterogeneous anycast networks, where some sites have more capacity than others, the defense approach can be different. In these cases, larger, "super"-sites can attract traffic from smaller sites. For moderate-size attacks, it may even be best for smaller sites to shut down if the super-sites can handle the traffic.

Regardless of attack size, traffic engineering allows the defender to shift attack traffic to absorber or breakwater sites. We next describe traffic engineering options, and then how one can automate response. For operators unwilling to fully automate response, our system can still provide recommendations for possible actions and their consequences.

3.4.1 Traffic Engineering to Manage an Attack

Given an overall defense strategy (absorb or rebalance), the defender will use traffic engineering to shift traffic, either automatically (§3.4.2) or as advice under operator supervision. For anycast deployments connected by the public Internet, BGP [8] will be the tool of choice to control routing and influence anycast catchments. Organizations that operate their own wide-area networks may also be able to use SDN to manage traffic on their internal WAN [31, 61]. Fortunately, BGP has well established mechanisms to manage routing policy. We use three BGP mechanisms in the paper: AS-Path prepending, BGP communities and Path Poisoning.

AS-Path Prepending is a way to de-prefer a routing path, send traffic to other catchments. BGP's AS-Path is the list of ASes back to the route originator. The AS-Path both prevents routing loops and also serves as a rough estimate for distance, with BGP preferring routes with shorter AS-Paths. By artificially inserting extra ASes into the AS-Path, the route originator can de-prefer one site in favor of others. Path prepending is known to be a coarse routing technique for traffic engineering. We measure how fine the control AS-Path prepending provides to anycast in §6.1.

We define **Negative Prepending** as the use of AS-Path prepending to draw traffic towards a site, preferencing one site over others. Prepending can only increase path lengths, but an anycast operator in control of all anycast sites can prepend at all sites except one, in effect giving that site a shorter AS-Path (relative to the other sites) than it had before. "Negative prepending by one at site S" is, therefore, shorthand for prepending by one at all sites other than S.

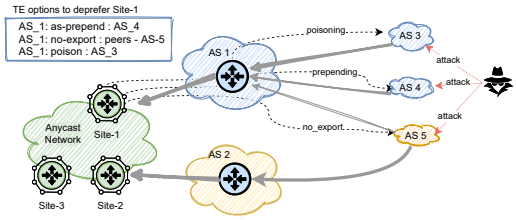


Figure 2: TE techniques to shift traffic from Site-1 to Site-2.

Long AS-Paths due to prepending can make prefixes more vulnerable to route hijacking [41]. However, this issue has a small impact on anycast prefix, as always there is a site announcing without any prepend, keeping the path length limited. We suggest that formal defenses to hijacking such as RPKI are needed even without prepending, and when they are in place, prepending can be an even more valuable tool for TE.

BGP Communities (or community strings) label specific BGP routes with 32 or 64 bits of information. How this information is interpreted is up to the ASes. While not officially standardized, a number of conventions exist where part of the information identifies an AS and the other part a policy such as blackholing, prepend, or set local-preference. Community strings are widely supported to allow ISPs to delegate some control over routing policy to their customers [1, 74].

Path Poisoning is another way to control the incoming traffic. This technique consists of adding the AS of another carrier to the AS PATH. Paths that repeat ASes in different parts of the AS PATH indicate routing loops and must be discarded by BGP.

When using path poisoning we announce a path with both the poisoned AS and own AS (otherwise neighbors may filter our announcement as not from us). We must therefore *also* prepend twice at all other anycast sites, otherwise poisoning also results in a longer AS path.

Figure 2 shows how traffic engineering can be applied to anycast systems in order to modify the catchment. In this example, *site-1* is overwhelmed by an attack. Aiming to shift bins of traffic to *site-2* with spare capacity, we can make BGP announcements. *site-1* poisons AS3, prepends (only showing to AS4), and prevents announcement to AS5 using *not-export* BGP community. These changes decrease load in *site-1*, shifting the traffic to *site-2*.

3.4.2 Automatic Defense Selection

To automate defense we use a centralized *controller*. The controller collects observations for all sites (from external measurements, or assuming the site is saturated if it cannot

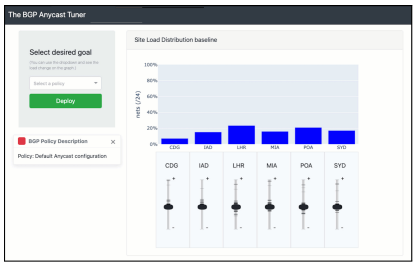


Figure 3: Operator assistance system.

reach the site), then takes action, if required (④ of Figure 1): (1) The controller identifies sites that are overcapacity by comparing estimated load to expected capacity and observed resources at each site. (2) The controller identifies all playbook options that will reduce load at any impacted sites without overloading currently acceptable sites. (3) It selects from any viable options, favoring a uniform distribution and smallest change (or selecting arbitrarily if necessary). If all changes leave some sites overwhelmed, it can choose the "least bad" scenario, or request operator intervention.

After deploying a new routing policy, the decision machine continues to evaluate the traffic level at each site (⑤ of Figure 1). If any site is still overwhelmed after 5 minutes, we try again, repeating size estimation, decision, and action. In the subsequent iterations, the controller only considers the routing options that were considered in the previous iteration (from step (2) of this decision process). We allow time between attempts so announcements can propagate [37]. To avoid oscillation or interference with route flap dampening, after three attempts we escalate the problem to the human operator. We choose these values for timer duration and number of retries from recommendations of operators to avoid oscillation, other options are possible. Explore other options is possible as future work.

Return to service: After a period with no overloaded sites, we can automatically revert any interventions, on the assumption that default routing provides users best service. Leaving interventions in place for some time can help with polymorphic attacks (§8).

3.4.3 Operator Assistance System

We discussed our approach with operators of root DNS and cloud services to get feedback on the approach. While they were enthusiastic about automated defenses to deal with common DDoS events, and to handle events during non-business hours, some operators prefer human-supervised (non-automated) response, and all expected human supervision of response during initial deployment to build trust before full automation.

To support human-supervised response, we design an operator assistance system as an alternative (or precursor) to automation. This system provides a web-based interface that activates route changes, coupled with playbook lookup that recommends good options based on current sensor status (Figure 3). Details are described in Appendix B.

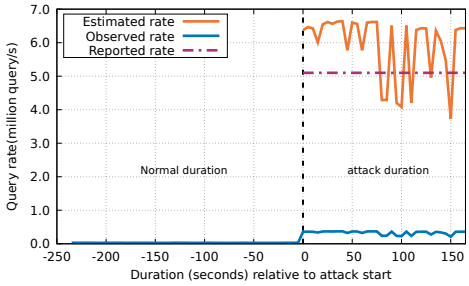


Figure 4: Estimating real-world attack events: estimating Nov. 2015 event with 5.59% access fraction.

4 Evaluation of Offered Load Estimation

We next evaluate estimating offered load with real-world events; a testbed evaluation is in Appendix C.

4.1 Case Studies

We test our approach with two large DNS DDoS events from 2015-11-30 and 2016-06-25. The November 2015 event was a DNS flood, and the June 2016 was a SYN and ICMP flood attack. `B-root` exhibited significant upstream loss in both these events, so we estimate true offered load to `B-root` and compare to observations at other roots for ground truth.

To apply our system we measure the access fraction (α) using the known-good traffic. Table 1 shows the expected typical known-good traffic ("normal"), the observed rate under attack ("observed") and the computed α. Here we use RIPE Atlas as known good [54]. We see similar results (omitted due to space) when using the top 100 heavy hitters.

Figure 4 compares the observed load (the bottom blue line) with the estimated offered load (the middle, varying, orange line) from our system, as compared to the attack rate reported from other roots (the dashed purple line). The offered load columns of Table 1 give numeric values.

Even though the attack was large, we see that the estimated attack size of the 2015 event of 4–6.5 Mq/s is close to the reported 5.1 Mq/s [45,47]. We also see similar results from the 2016 event [48], where we estimate 8–11 Mq/s of total traffic, compared to the 10 Mq/s reported rate (details with figure in §C.2). We also add the result from Testbed experiment which also shows a good accuracy (details in §C.1).

These two events show that even with high rates of upstream loss we are able to get reasonable estimates of total offered load. Our results provide good accuracy when the known-good traffic has 2500 queries/minute with RIPE, and additional known-good traffic can improve accuracy. Use of additional known-good traffic (such as heavy hitters) improves accuracy in these cases by providing a larger signal. However, in practice, even a rough estimation allows a far better response than using directly observed load.

We conclude that attack size estimation is close enough to help plan response to DDoS events.

Scenario/ Date	Dur.	known-good traffic normal	observed	α	offered load during attack normal	observed	reported	estimated	$\hat{\alpha}$	estimated/ reported
2015-11-30	3h	33.08	1.85	0.0559	0.03 M	0.37 M	5.1 M	6.6 M	0.07	1.3
2016-06-25	3h	36.58	0.33	0.0091	0.03 M	0.10 M	10 M	11 M	0.01	1.1
Testbed	5min	425.2	207.0	0.4900	8.5 k	16.3 k	29.2 k	33.2 k	0.56	1.1

Table 1: Estimating sizes of offered load (second from right) based on known-good traffic (second from left) with real-world attacks at `B-root` and testbed experiment. Traffic rates are in queries/second (reporting only the peaks).

Testbed	Used Sites	#
`Peering`	Amsterdam*†(AMS), Boston* (BOS), Belo Horizonte*†(CNF), Seattle* (SEA) Athens* (ATH), Atlanta* (ATL), Salt Lake City* (SLC), Wisconsin* (MSN)	8
`Tangled`	Miami (MIA)*, London (LHR)*, Sydney (SYD)*, Paris (CDG)*, Los Angeles (LAX)*, Enschede (ENS)*, Washington (IAD)*, Porto Alegre (POA)*†	8

Table 2: Testbed and respective sites used in our experiments. Transit providers (*) and IXP (†).

Experiment	Key Takeaways
Path prepending	Works everywhere to effectively de-prefer a site (§6.1.2), but shifts traffic in large amounts (§6.1.3), and has few traffic levels (Figure 6).
Neg. Prepending	Works everywhere to prefer a site (§6.1.2).
BGP communities	Although widely implemented, well-known communities are not universal (§6.2.1).
	When supported, they provide finer-granularity control than prepending (§6.2.2).
BGP path poisoning	Many Tier-1 ASes drop the announcements when it sees Tier-1 ASes in the paths. (§6.3.1)
	Control over traffic is limited by the filters from other ASes. (§6.3.2).

Table 3: Experiment summarization and findings.

5 Evaluation Approach

We next describe how we will evaluate the effectiveness of TE (§6) and that results generalize to different deployments (§7). Traffic engineering in response to DDoS depends on the anycast deployment—where sites are and with whom they peer. We evaluate on two different testbeds. Our approach (estimation, TE, and playbook construction) can be applied anywhere with different anycast setups. We expect network operators will execute our approaches on a test prefix (in parallel with their operational network) prior to an event so that no service interruption happens.

5.1 Anycast Testbeds

We evaluate our ideas on testbeds to see the constraints of real-world peering and deployments. We use two independent testbeds: `Peering` [60] and `Tangled` [7]. Table 2 summarizes information about each testbed with their own set of geographically distributed sites along with their locations (`Peering` supports more sites but we used 8 sites). These sites show different connectivity, and have one or more transits and IXP peers. Most `Peering` sites have academic transits while `Tangled` has more commercial providers. Our testbed is about the same size as many operational networks, since nearly half of real-world networks have five or fewer sites [16].

5.2 Measuring Routing Changes

To measure the effect of a BGP change, we first change the routing announcement at a site, give some time to propagate, confirm that the announcement is accepted, and finally start the anycast measurement.

Route convergence: After a change, we allow some time for *BGP route propagation*. We know that routing and forwarding tables can be inconsistent (resulting in loops or black holes) while prefix is updating [37, 67, 76]. Although routing updates are usually stable within 5 minutes [67], we wait 15 minutes for routing to settle when building our playbook since it is a non-attack period. When the attack is not mitigated after deploying a routing policy, our system moves to a different approach after 5 minutes.

Propagation of BGP policies: Policy filtering could limit the acceptance of announced routes, although in practice these limits do not affect our traffic engineering. Best practices for networks at the edge to filter out AS-Paths longer than 10 hops, and ASes in the middle often accept up to 50 hops, both more prepends than we need. Based on routing observations from multiple global locations using RIPE RIS, we confirm that configurations in our experiments are never blocked due to route filtering in multi-hops away from our anycast sites.

6 Traffic Engineering Coverage and Control

From an estimate of attack load, operators use BGP to shift traffic. We next evaluate three TE mechanisms: AS-Path prepending, community strings and path poisoning. For each we consider when it works and what degree of control it provides. Table 3 summarizes our key results from tests on two testbeds (§5.1); in §7 we evaluate generalizability.

6.1 Control With Path Prepending

First we consider AS-Path prepending as a defense strategy.

6.1.1 Prepending coverage

Support for AS-Path prepending is quite complete—it requires no explicit support from the upstream provider, so we found prepending worked at all sites in both of our testbeds. In `Peering`, we are allowed to use a maximum of three prepends, and in `Tangled` we use up to five prepends. Previous study [14] shows a maximum of 5 prepends is sufficient because 90% of active ASes are located less than six AS hops away. We use RIPE RIS [56] to check the routing visibility

when prepends are in place, and we do not observe changes in the routing propagation for both testbeds. Otherwise, this might reveal the existence of AS path length filters [32, 33].

6.1.2 Does prepending work?

Since AS-Path prepending is widely supported, we next evaluate this attractive TE method.

We explore this question for a representative scenario using `Peering` using three sites from three continents—Europe (Amsterdam-AMS), North America (Boston-BOS) and South America (Brazil-CNF). In §7 we generalize to other configurations. We estimate load by counting /24 blocks in catchments, then compare the baseline with TE options. (We also explored traffic weighted by traffic loads instead of blocks, getting the same qualitative results and shapes with different constants, Appendix F.)

Figure 5 shows the traffic from each site under different conditions. The middle bar in each graph is the baseline, the default condition with no prepending. We then add prepending at each site, with one, two or three prepends in each bar going to the right of center. We also consider negative prepending (§3.4.1) in one to three steps, with bars going left of center.

We first consider the baseline (the middle bar) of all three graphs in Figure 5. Amsterdam (AMS, the bottom, maroon part of each bar) gets about 68% of the traffic. AMS receives more traffic than BOS and CNF because that site has two transit providers and several peers, and Amsterdam is very well connected with the rest of the world.

We next consider prepending at each site (the bars to the right of center). In each case, *prepending succeeds at pushing traffic away from the site*, as expected. For AMS, each prepend shifts more traffic away, with the first prepend cutting traffic from 68% to 37%, then to 29%, then to about 16%. BOS and CNF start with less traffic and prepending has a stronger effect, with one prepend sending most traffic away (at BOS, from 15% to 7%) and additional prepends showing little further change. These non-linear changes are because changing BGP routing with prepending is based on path length, and the Internet's AS-graph is relatively flat [2, 15].

The bar graphs also show that when prepending pushed traffic away from a site, it all goes to some other site. Where it goes depends on routing and is not necessarily proportional to the split in other configurations. For example, after one prepend to AMS, more traffic goes to CNF (the top sky blue bar) than to BOS (the middle yellowish bar). These unexpected shifts are why we suggest pre-computing a "playbook" of routing options before an attack (§3.2) to guide decisions during an attack and anticipate the consequences of a change.

We also see that negative prepending succeeds at drawing traffic towards the site—in each case the bars to the left of center see more traffic in the site that is not prepending while the others prepend. AMS sees relatively little change (68% to 89%) since it already has most traffic, while BOS and CNF each gain up to 68% of traffic.

All three sites show some networks that are "stuck" on that site, regardless of prepending. One reason for this stickiness is when some networks are only routable through one site because they are downstream of that exchange. We confirm this by taking traceroute to two randomly chosen blocks that are stuck at BOS. Traceroutes and geolocation (with Maxmind) confirm they are in Boston, at MIT and a Comcast network (based on the penultimate traceroute hop). We have used the local-preference BGP attribute to move such stuck blocks, but a systematic exploration of that option is future work.

In summary, the experiment shows that AS prepend does work and can shift traffic among sites, however, this traffic shift is not uniform.

6.1.3 What granularity does prepending provide?

Having established that prepending can shift traffic, we next ask: how much control does it provide? This question has two facets: how much traffic can we push away from a site or attract to it, and how many different levels are there between minimum and maximum.

Limits: Figure 5 suggested that in `Peering`, with those three sites, there is a limit to the traffic that can shift. AMS, BOS, and CNF always get about 16%, 7% and 3% of blocks, regardless of prepending.

Figure 6 confirms this result with a 5-site deployment (two from Europe, one from North America, one from South America and one from Australia) in our other testbed (`Tangled`). X axis is presented with the number of prepends applied to each site. The number zero (0) represents the `baseline`, the positive numbers (1-5) are the number of prepending applied and the negative numbers represent negative prepends. As depicted, each site can capture at most 55–65% of blocks, and can shed at most 95% of blocks, even with up to 5 prepends. We can also see that we do not get a granular control as only three points are between the minimum and maximum.

We conclude that while prepending can be a useful tool to shift traffic, it provides relatively limited control.

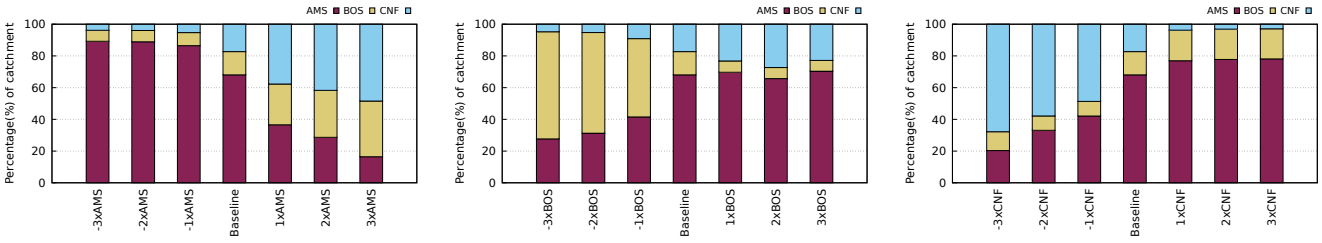
6.2 Control with BGP Communities

We next show that BGP community strings have the opposite trade-off: what options they support vary from site to site, but when available, they provide more granular control over traffic. We use whatever community strings that can be supported at each site. Specific values for the same concept often vary.

6.2.1 Community string coverage

ASes must opt-in to exchange community strings with peers, as opposed to prepending's near-universal support (since AS paths are used for loop detection, prepending works unless it is explicitly filtered out). Explicit support is required because communities are only a tagging mechanism; the actions they trigger are at the discretion of peering AS. Prior work has studied the diverse options supported by community strings [28].

To evaluate coverage, we review support for BGP communities in the testbeds we use. The testbeds provide information



(a) AMS site. (b) BOS site. (c) CNF site.

Figure 5: `Peering`: Impact of path prepending in catchment distribution with AMS, BOS and CNF sites on 2020-02-24.

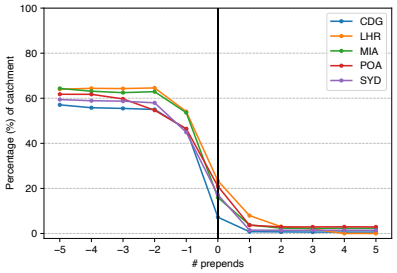


Figure 6: `Tangled`: Effect of path prepending on catchments.

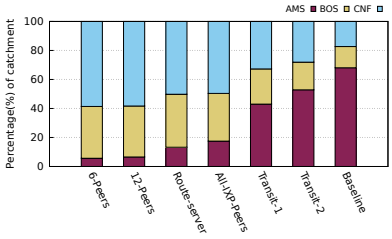


Figure 7: `Peering`: Community strings (at AMS) on catchments for AMS, BOS, CNF on 2020-02-25.

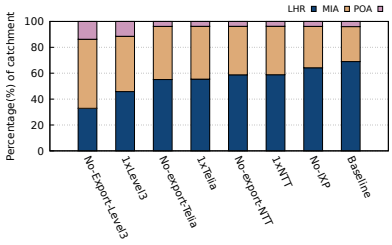


Figure 8: `Tangled`: using different communities to shift traffic on site LHR on 2020-04-05.

about two dozen locations with diverse peers. Each one of these peers has been evaluated about its support to this feature.

In Table 4 we describe path prepending and poisoning support and what types of community strings are supported at each site. We group communities by class: advertisement options (no-peer, no-export to customers, and no export to anyone), selective prepending, and peers and transits that support selective advertisement. We also show the number of non-transit peers and transits.

`Peering` allows selective announcement to the transits and peers at each site, although the number of peers and transits varies. Many sites with one transit provide no alternatives. We considered selective announcement options at AMS, with 854 peers (106 bilateral peers including 2 route servers with 748 peers), and 2 transit providers [60]. CNF has one transit provider and 129 peers (with only 6 bilateral peers, other peers are connected through 2 route servers). For our Verfploeter measurement, we consider the peers and route servers with bilateral BGP sessions. A single peer covers a small fraction of the address space in our Verfploeter measurement. For some peers, we observed no coverage at all which requires further investigation with the peers to confirm our observation. Hence, all the selective announcement options do not make difference in the catchment distribution (see the catchment in AMS with 12 peers compared to the transit-1 in Figure 7). The *options* column of Table 4 summarizes these results, showing how many routing options we have using community strings.

We evaluate `Tangled` to provide a second deployment with different peers. `Tangled` built its anycast network over cloud providers, crowd-sourced transit providers and IXPs. All transit providers and IXPs sites support communities as described in Table 4. With `Tangled`, the POA site has 250 peers and most of them support communities strings.

We conclude that the number of options at each anycast site may vary depending on the number of connections with peers and transits. This uncertainty shows the need for a playbook that shows the possible options.

6.2.2 At what granularity do community strings work?

We next examine how well community strings work and what granularity of control they provide. We use community strings to make BGP selective announcements, where we propagate our route only to specific transit providers or IXP peers.

For our experiment, we use `Peering`, varying announcements at AMS and observing traffic when anycast is provided from AMS, BOS and CNF (the same topology as §6.1.2). As described in §6.2.1 selective announcement community strings are provided only at AMS and CNF, and they affect our Verfploeter measurement only at AMS with several peers together, two transits one by one, and route servers.

To select the target ASes for selective announcement, we sort all the working peers of AMS site, based on the size of their customer cone using CAIDA's AS rank list [9]. We then choose the 6 largest IXP peers and the 12 largest, as the left two bars in Figure 7. We then examine the route server, announced separately (the next bar), and then all IXP peers including route servers. Finally, we see the coverage with each of the two transit providers, announced separately.

First, we see that selective announcement provides more control than prepending, as AMS shifts from baseline 68% of blocks to other configurations from 53 to 6% of blocks.

Second, we see that there is some overlap in some com-

Site:	Peering								Tangled							
Routing policy	AMS	BOS	CNF	SEA	ATH	ATL	SLC	MSN	MIA	LHR	IAD	CDG	LAX	ENS	SYD	POA
AS-path prepend	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
no-peer	✓	–	✓	–	–	–	–	–	✓	✓	–	✓	–	–	✓	✓
no-export	△	–	–	–	△	–	–	–	✓	✓	–	✓	–	–	✓	✓
no-client	–	–	–	–	–	–	–	–	✓	–	–	–	–	–	–	–
Selective prepend	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	✓	–	–	✓	✓
Selective announcement	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	-	-	✓	✓
Path poisoning	✓	✓	✓	✓	✓	✓	✓	✓	✓	–	–	–	–	–	–	✓
# non-transit peers	854	0	129	0	0	0	0	0	0	0	0	0	0	0	0	250
# transits	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2
# options	856	1	130	1	1	1	1	1	1	1	1	1	1	1	1	252

Table 4: Traffic engineering options on each testbed sites. ✓: supported, -: not supported, △: not tested.

binations. For example, each transit reaches more than half of all blocks reachable from AMS, so we know some blocks are reachable from both transit providers. Thus, while there is some control over how many blocks to route to AMS, some peers are very "strong" and will pick up many blocks if they are allowed to announce our prefix.

Third, we see the important role of route servers. While direct coordination with 12 IXP peers brings only 7% blocks at AMS, a route server lets AMS reach more ASes and 14% of the blocks alone.

Finally, we see that transit providers play an important role. AMS site has two transit providers—BIT BV (AS12859) and Netwerkvereniging Coloclue (AS8283). Announcing to AS8283 attracts more traffic to AMS than announcing to AS12859. Different AS relationship of these two transits with their upstream provides us a different traffic distribution.

As shown in our experiments, when compared to AS path prepending, BGP communities provide way more better control over traffic distribution.

To investigate if the results found on `Peering` can be generalized, we made a set of experiments on `Tangled`. Like `Peering`, we select 3 sites from three continents—London(LHR), Miami (MIA) and Porto Alegre (POA), and use communities for selective prepending and selective announcement from LHR. In Figure 8, we show the catchment distribution after using the community strings from LHR. In the baseline, when no communities are used, LHR handles 69% of traffic. From right to left, we see a gradual decrease in the catchment distribution from 69% to 33%. Stop announcing to IXP peers reduces traffic from 69% to 64%. But using prepending and no export communities in AS2914 (NTT America), AS1299 (Telia Company) and AS3356 (Level 3), we can get 30-60% of the catchments in LHR.

Both testbeds show that community strings are not widely available in all sites, and that even well-known communities are not fully adopted. However, community strings can provide finer-grained control. Selective announcement mostly provides more "flexibility" depending on how many IXP peers and transits are connected. We also find that some sites do not provide the support that we expect which means community strings require an extra step like contacting the transit provider for an explicit agreement.

6.3 Control with Path Poisoning

We next turn to path poisoning, and show that like community strings, coverage and granularity are limited by routing filters deployed in upstream peers.

6.3.1 Poisoning coverage

Support for path poisoning is dependent on the ASes we are poisoning and on route filters deployed by our upstream ASes.

We find that many ISPs, especially Tier-1 ASes, filter out AS paths that poison *any* Tier-1 AS. Tier-1 ASes deploy these filters to block BGP announcements from customers that contain other Tier-1 ASes in the path to prevent route leaks [43, 71]. This filtering often makes path poisoning ineffective to control traffic.

To verify that poisoning Tier-1 ASes is often ineffective from filtering, we poison Tier-1 ASes announcing only from AMS in `Peering`, a unicast set-up blocking the impacts of other sites, and make traceroutes from 1000 RIPE vantage points to our prefix. Our measurement shows the evidence of filters when we poison Tier-1 ASes—AS7018 (AT&T), AS6453 (Tata Communications America), and AS1299 (Telia Company). We observe many vantage points fail to reach our prefix as they are dependent on Tier-1 ASes for their routes. Some others change their paths avoiding Tier-1 ASes. We also validate route disappearance via most Tier-1 ASes using RouteViews telescopes [77].

Although poisoning Tier-1 ASes is often ineffective, poisoning is effective with most non-Tier-1 ASes. Unfortunately, these ASes carry little traffic when they are not immediate upstreams. Poisoning these small ASes only has little impact on traffic. We again traceroute after poisoning a non-Tier-1 AS (AS57866), and observe that Tier-1 ASes propagate the poisoned path. This proves poisoned paths with Tier-1 and non-Tier-1 ASes are treated differently by other ASes.

6.3.2 What granularity does poisoning provide?

Path poisoning coverage is limited because one cannot usually poison a Tier-1 AS. This same filtering limits the granularity that poisoning allows: poisoning Tier-1 ASes is not allowed, poisoning non Tier-1 ASes has little impact when they are multiple hops away because they represent little traffic. Poisoning immediate neighbors may shift traffic, but is more complex than just not announcing to them. We confirm these

Routing Policy	Traffic to Site (%) AMS	BOS	CNF
(a) 6peers, 12peers	~5	~35	~55
(b) Route-server	15	35	55
(c) All-IXP-Peers/Poison transits	15	35	45
(d) 3xPrepend AMS	15	35	45
(e) 2xPrepend AMS	25	35	45
(f) 1xPrepend AMS	35	25	35
(g) -3xPrepend BOS	25	65	5
(h) -2xPrepend BOS	35	65	5
(i) -1xPrepend BOS	45	45	15
(j) -3xPrepend CNF	25	15	65
(k) -2xPrepend CNF	35	5	55
(l) -1xPrepend CNF	45	5	45
(m)Transit-1	45	25	35
(n) Transit-2	55	15	25
(o) Poison Tier-1/Transit-2	35	25	35
(p) Poison Transit-1	55	25	25
(q) Baseline	65	15	15
(r) 1,2xPrepend BOS	65	5	25
(s) 3xPrepend BOS	75	5	25
(t) 1,2,3xPrepend CNF	75	15	5
(u) -1,-2,-3xPrepend AMS	85	5	5

Table 5: Policies and traffic distribution (in 10% bins); groups sorted by rough fraction of traffic to AMS, and colors showing the traffic compared to the baseline distribution.

observations with detailed experiments in Appendix D, but we conclude that path poisoning is not generally an effective tool for traffic engineering.

6.4 Playbook Construction

Based on our understanding of prepending, communities and poisoning, we can now build a playbook of possible traffic configurations for this anycast network. In practice, we build the playbook automatically using scripts that connect to BGP, then iterate through different BGP configurations, then run Verfploeter [20] to measure new catchments. Playbooks are necessarily specific to each anycast deployment, but we show in §7 that the process generalizes. Using a playbook, an operator does not need a single "best" approach, rather a combination of approaches in the playbook ensures a greater control over traffic distribution.

A playbook is a list of variations of routing policy and the resulting traffic distributions. Table 5 shows the playbook for our testbed, with the baseline of 65% blocks to a site shown in white. We group different levels of prepending (positive or negative) at each site, and show selected community string and poisoning configurations.

To summarize the many configurations from Table 5, Table 6 identifies which combinations result in specific traffic ratios at each site. Each letter in this table refers back to a specific configuration from Table 5. During an attack, if the anycast system begins at the baseline configuration (q), if AMS is overloaded, the operator could select a TE configuration higher in the table (perhaps 'e', 'g', or 'j'). The operator can then see the implications of that TE choice on other site (for example, 'e' increases load on both other sites, with 'g' increases load on BOS but decreases it at CNF).

Traffic to Site (%)	AMS	BOS	CNF
0-10	a	k, l, r, s, u	g, h, t, u
10-20	b, c, d	j, n, q, t	i, q
20-30	e, g, j	f, m, o, p	n, r, p, s
30-40	f, h, k, o	a, b, c, d, e	f, m, o
40-50	i, l, m	i	c, d, e, l
50-60	n, p	–	a, b, k
60-70	q, r	g, h	j
70-80	s, t	–	–
80-90	u	–	–
90-100	–	–	–
Traffic options	9	6	7

Table 6: `Peering` playbook (AMS, BOS, and CNF)

An operator may also use a playbook with traffic load for two reasons. First, loads in most interesting services have diurnal pattern. Second, loads from each /24 prefix may vary because of the number of clients behind each prefix (more on Appendix F). Building the playbook with load is computationally simple; an operator can just use the same catchment mapping along with the per prefix load.

Even with attack size estimation, attacks are accompanied by uncertainty, and attacker locations may be uneven. However, the playbook provides a much better response than "just relying on informal prior experience" in two ways: the defender can anticipate the consequences of the TE action (that traffic will go somewhere!), and the defender can choose between different possible outcomes if the first is incomplete.

Playbook flexibility and completeness: Table 6 helps quantify the "flexibility" that traffic engineering allows us in this anycast deployment. Using these 10% traffic bins, we see that AMS has 9 options, CNF 7, and BOS only 6. Because AMS and CNF mostly swap traffic after TE changes, and because BOS is less well connected, no configuration with three sites allows BOS to take traffic within 50-60% range, and no 3-site configuration can drive BOS or CNF over 70%.

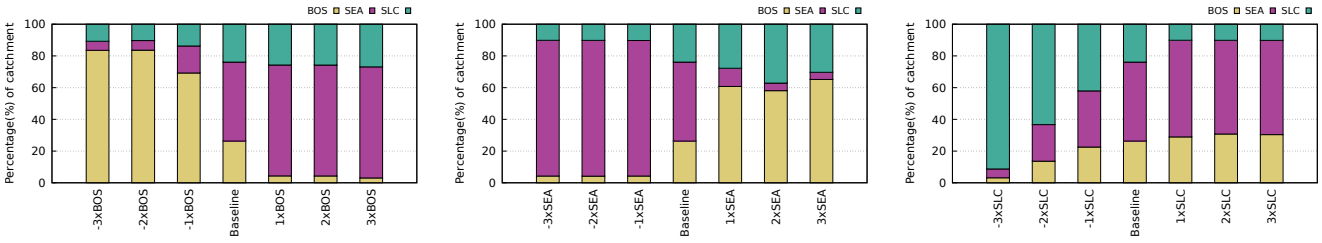
This analysis shows more central sites like AMS, and it may suggest the need for topology changes (perhaps adding another site in Europe or Asia to share AMS' load).

7 Deployment Stability and Constraints

In §6 we showed BGP-based TE provides considerable flexibility. Building playbooks supports defenders by allowing them to explore how transit providers, prepending, community strings, and poisoning affect their specific deployment. We next look at how stable the results are depending on choice of sites and the number of sites. While the details of the playbook vary for each deployment, and we do not claim our testbeds represent all possible deployments, we show our approach is flexible and can respond to attacks in different deployments—our approach generalizes.

7.1 Effects of Choice of Anycast Sites

First we see how sites affect our playbook. New sites change catchments because they depend on location and peering,



(a) BOS site. (b) SEA site. (c) SLC site.

Figure 9: `Peering`: Impact of choosing BOS, SEA and SLC sites on 2020-02-28

In §6.1, we studied catchments with three specific `Peering` sites on three continents: AMS, at a large, commercial IXP in Europe; CNF with an academic backbone transit in Brazil; and BOS, an academic site in the U.S. We now switch to three educational sites all in the United States: SEA, at University of Washington on the west coast; SLC, at the University of Utah in the Rockies; and BOS, at Northeastern University in Boston on the east coast.

More important than just geographic location, site connectivity is the most important factor in choosing sites. Multiple transit providers increase the chance of having more BGP options to affect traffic control and granularity. While a poorly connected site inside a university network tends to provide less traffic control options.

Prepending baseline: Figure 9 shows catchment sizes for the three North American sites with positive and negative prepending. Now the baseline distribution is unbalanced, but less so than before, with SEA capturing 50% of blocks. We discussed SEA's heavy traffic with the `Peering` operators. They suspect that SEA is near to the Seattle IXP, making its paths one hop from many commercial providers. Which site has the greatest visibility depends on its peering and will vary from deployment to deployment.

Prepending coverage and granularity: As with our prior experiments, we can adjust prepending to see how traffic shifts. With these three sites, traffic shifts very quickly for BOS and SEA after one positive or negative prepend. SLC has more flexibility, perhaps because it has the smallest catchment at the baseline, and gains more coverage with each step of negative prepending, to 42%, 63%, and 91% of blocks. Often (but not always), we see that academic sites exhibit less granularity because either they have few peers, or their peers are academic networks with similar connectivity. As a result, minor changes in AS-Path length place one site further from the others. In addition, this less granular control shows the importance of building a playbook that is specific to a given deployment, or when the anycast topology changes.

Community coverage: While communities are common at IXPs and transit providers, academic networks (NRENs) have a more simple set of communities. None of those academic sites provide community strings.

This observation confirms our prior coverage observation: community string support is not uniformly available. We also looked at other combinations of sites in `Peering` and found similar results (in the extended version of the paper [57]).

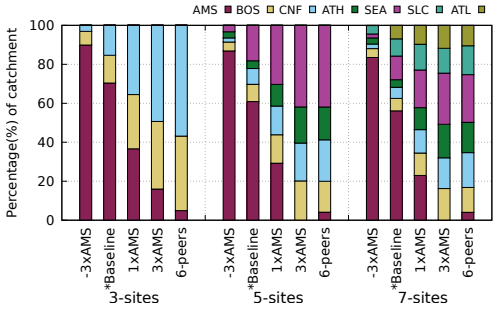


Figure 10: `Peering`: Impacts of changing the number of anycast sites from 2020-04-07 to 2020-04-10.

Path poisoning: We repeated our path poisoning experiments with three sites in Boston, Salt Lake City and Seattle. We confirm that Tier-1 ASes typically cannot be poisoned (§6.3.1). We also see filters designed to prevent route leaks [71] also interfere with poisoning.

Our experiments confirm that while catchments are deployment-specific, our qualitative results hold—prepending works but is coarse, and community strings and poisoning are not supported everywhere.

7.2 Effects of Number of Anycast Sites

Next, we vary the *number* of sites and see how that changes control traffic. We select 3, 5 and 7 sites from each testbed, and build a playbook to evaluate defense options. Figure 10 shows selected configurations, grouped by number of sites.

Baseline: With more sites, overall capacity increases and *baseline load at each site falls*. For example, in Figure 10, the baselines (with an asterisk*) at the largest site (AMS) shifts from 70% of blocks with three sites to 61% and 56% with 5 and 7 sites. Smaller sites shift less (BOS goes from 14% to 6% and 6%, and CNF from 15% to 8% and 6%). Greater capacity and distribution requires a larger and distributed attacker to exhaust the overall service. We see similar results on our alternate testbed `Tangled`, as described in the extended version of the paper [57].

Traffic flexibility: With more sites, *the largest site usually shows the largest changes* and has the fewest catchment sizes. Comparing baseline to one prepending in Figure 10, AMS shifts from 70% to 37% with three sites, from 61% to 29%

Months	AMS(%)	BOS(%)	CNF(%)
2020-02	68.1	14.6	17.3
2020-04	70.4	14.2	15.4
2020-06	65.3	14.1	20.6

Table 7: Percent blocks in each catchment over time.

with five, and from 56% to 23% with seven, always dropping by half.

Even with more sites, some blocks are often "stuck" at a particular site. With three negative prepends, AMS gets most of the traffic, but it tops out at 90% with three sites, and only 87% and 84% with five and seven. We conclude that each site has its own set of "stuck blocks" that are captive to it and will not move with traffic engineering.

With more sites, the *fine control of BGP communities becomes more important* because path-prepending becomes less sensitive. For example, selective announcements with communities are need for AMS with 5 or 7 sites; prepending three times shifts all traffic.

New sites: Adding more sites also shows how our playbook can help guide deployment of new sites. Predicting traffic shifts for a new site is difficult, but experimenting with a test prefix can build a playbook pre-deployment.

7.3 Playbook Stability Over Time

A playbook has a limited use if routing changes immediately. We know routing changes when links fail, or when ISPs begin new peering or purchase new transit. For how long is a playbook applicable?

To answer this question, Table 7 shows the fraction of /24 blocks going to each catchment over time for the baseline configuration. We see that the fraction of blocks is generally quite stable, with only about 5% of blocks shifting in or out of a site. In addition, prior work has shown very strong anycast stability over hours to days [38, 79]. We checked the stability of `B-root` catchment. We found that after two weeks 0.35% prefixes, and after one month only 0.65% prefixes changed their catchment (more on Appendix E). While catchments are relatively stable, we expect operators will refresh playbooks periodically (perhaps weekly or monthly).

8 Defenses at Work

In this section we describe four real-world attacks processing the traffic in our system. We show that we can successfully respond to a different types of attacks in different ways.

Methodology: We use real-world attacks from `B-root` server operator, the Dutch National Scrubbing Center, and from an anonymized enterprise network. These events include polymorphic, adversarial, and a volumetric attack.

We evaluate these events by simulating traffic rates against a three-site anycast network. The first two events use `Peering` with our AMS, BOS, CNF configuration from §6. We vary this topology, using BOS, SEA, SLC from §7.1 in the last event. We replay the traffic in simulation, assigning traffic to each anycast site based on catchments measured in our experiments. We do not simulate the gradual route propagation, but instead have routing take effect 300 s after a change (a conservative bound, most routing changes happen in half that time). We then evaluate traffic levels at each site and compare that to a target capacity.

For each attack we run our system in defense, estimating the attack size and selecting a pre-computed playbook response. Since our playbook allows different responses: when we have choices we select different methods of defense: prepending, negative prepending, or community strings (Figure 11).

A 2017 polymorphic attack: Our first event is a DNS flood from 2017-03-06 in `B-root` [51] (Figure 11a). This event was a volumetric polymorphic attack where the attack queries have common formats like *RANDOM.qycl520.com\032* (from 0 s) and *RANDOM.cailing168.com\032\032* (changed at 4750 s, so polymorphic in nature). We assume 60k packets/s (30 Mb/s) capacity at each anycast site. The event was small enough that `B-root` was able to fully capture it across all active anycast sites at the time. The event lasted about 5 hours, but we show only the first 2.25 hours. Services and attacks capacity today will both be much larger; we use a small attack, scaling the attack and capacity up would show similar results.

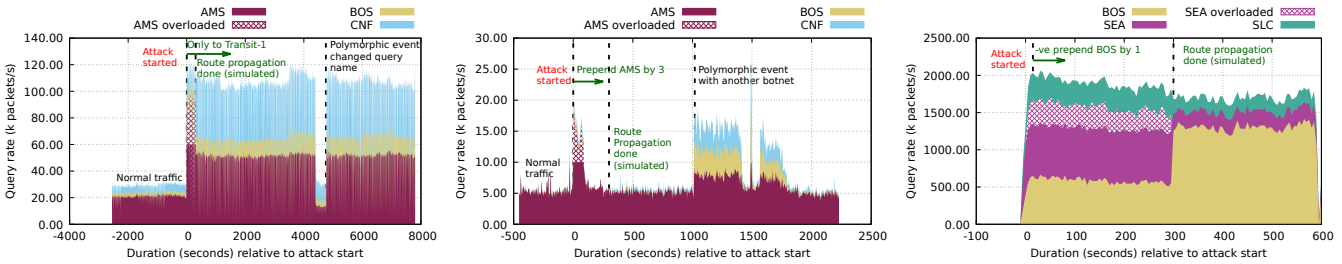
In Figure 11a we can identity AMS site receives 100k packets/s traffic that is more than the capacity (shown as the maroon striped area). Our system notices the attack from bitrate alerts. It then estimates the AMS overload by computing the offered load using observed load and access fraction. The system maps networks to number of packets to each site using the pre-computed playbook (Table 6). Using this mapping our system/operator can then select a response. From Figure 11a, we can see the impact of the selected routing approach—announcing only to Transit-1 using community string. After 300 s, we can see no striped area which indicates the attack is mitigated.

The attacker changes the query names at 4750 s, making this attack polymorphic. Filtering on query names would need to react, but our routing changes can still mitigate the attack regardless of this type of change.

A 2021 variable-length polymorphic attack: We next examine an HTTP-attack launched on an enterprise network on 2021-09-05 in Figure 11b. This polymorphic attack changes after each of three pauses. The initial attack consists of millions of HTTP GETs (15k packets/s) launched from an IoT botnet; it terminates when the enterprise's operator deploys IP-based filtering. About 1000 s later, a different botnet launched a multi-vector attack combining HTTP GETs using random paths (to avoid caching) and spoofed TCP ACKs. We then see a lull, brief burst, another lull, and a burst to the end.

The initial attack at time 0 overloads one site (AMS), prompting our routing response. After the estimation, we begin a route shift away from AMS, but the attack ends quickly (after 90 s), while routes are still changing.

Since the normal traffic sources originate from Europe,



(a) A polymorphic attack at `B-root` defended with community strings.

(b) An adversarial event at an enterprise mitigated using positive prepending.

(c) An event captured at the Dutch National Scrubbing Center defended using negative prepending.

Figure 11: Different attacks with various responses.

most traffic went to AMS even after three prepends. At 1020 s the attack botnet changes, with more attack traffic from Asia and South America (based on IP geolocation from MaxMind) Our route changes in response to the initial attack are still in place, and the renewed attack is successfully spread over all three sites, allowing AMS to tolerate the new attack.

Shifting attacks like this are common with more sophisticated adversaries. Any approach (including ours) that defends with routing changes is limited by route propagation times, so the applicability of such defenses is limited for short-lived attacks like what occurred at 0 s. However, spreading traffic protects against many types of attack, as we see the renewed attacks after 1000 s. Varying attacks like this show the importance of reviewing defense effectiveness as the attack continues.

An example attack on a different anycast topology: Finally, we consider an LDAP amplification attack, at the Dutch National Scrubbing Center on 2021-08-25.

In this case we simulate a super-site at BOS, capable of absorbing 1500k packets/s, while the other sites (SEA and SLC) support about half (700k packets/s). In Figure 11c, the purple cross-hatched area shows how much the traffic will overwhelm SEA, a smaller site, but can be handled at the super-site. We respond with negative prepending, with the traffic shift to BOS visible at 300 s. This response mitigates the attack (no striped area).

Other attacks: We have assessed additional attacks, and describe them in Appendix G. The additional polymorphic and volumetric attacks show that routing can successfully address attacks after routes propagate.

9 Limitations and Future Work

Our playbook of routing options (§3) is effective against many attacks (§8 and Appendix G). However, like any defense, it is not impervious. We next describe known limitations and areas of future work.

First, Internet routing is distributed, requiring time to converge. The effects of routing defenses cannot be seen until convergence. We do not make changes faster than 5 minutes.

Routing convergence time implies that routing changes will have limited applicability to short-lived attacks (less than 5 minutes). Although routing changes will not hurt the service, their benefits may not occur until routing shifts.

In addition, routing convergence means that polymorphic attacks that shift traffic sources quickly will be more effective. Routing changes are robust to polymorphic attacks that change method but take effect by traffic volume, they will spread load regardless of what it is, as we show in events in §8. However, when defending an attack where traffic shifts locations faster than routing converges, one must provision for the worst case volume to any site under the heaviest traffic it sees. Rapid shifts make defense harder, but not impossible.

Finally, we assume the anycast catchments of the underlying service change slowly (over days). We showed in §7.3 that this assumption generally holds.

Although we change routing during an attack to balance load across catchments, we do not explicitly attempt to locate attack origins. As future work, we could use such information to improve defense selection.

Attack response depends on human factors in service operators and attackers. Explicitly studying such human factors is potential future research. Our current work focused on the technical feasibility of our defenses.

10 Conclusions

This paper provides the first public evaluation of multiple anycast methods for DDoS defense. Our system estimates attack size, selects a strategy from a pre-computed playbook, and automatically performs traffic engineering (TE) to rebalance load or to advise the operator. Our contributions are attack-size estimation and playbook construction. We experimentally evaluate TE mechanisms, showing that prepending is widely available but offers limited control, while BGP communities and path poisoning are the opposite.

Acknowledgments: ASM Rizvi and John Heidemann's work on this paper is supported, in part, by the DHS HSARPA Cyber Security Division via contract number HSHQDC-17-R-B0004-TTA.02-0006-I. Joao Ceron and Leandro Bertholdo's work on this paper is supported by Netherlands Organisation for scientific research (4019020199), and European Union's Horizon 2020 research and innovation program (830927). We would like to thank our anonymous reviewers for their valuable feedback. We are also grateful to the `Peering` and

Tangled admins who allowed us to run measurements. We thank Dutch National Scrubbing Center for sharing DDoS data with us.

References

[1] AMPATH. Bgp resources. https://ampath.net/AMPATH_BGP_Policies.php. [Online; accessed 12-Oct-2021].

[2] APNIC. BGP-stats routing table report—Japan view. https://mailman.apnic.net/mailing-lists/bgp-stats/archive/2020/05/msg00001.html, May 1 2020.

[3] Vaibhav Bajpai, Steffie Jacob Eravuchira, and Jürgen Schönwälder. Lessons learned from using the ripe atlas platform for measurement research. *ACM SIGCOMM Computer Communication Review*, 45(3):35–42, 2015.

[4] Hitesh Ballani, Paul Francis, and Sylvia Ratnasamy. A measurement-based deployment proposal for IP anycast. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pages 231–244, 2006.

[5] Ran Ben-Basat, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9, 2016.

[6] Terry Benzel, Robert Braden, Dongho Kim, Cliford Neuman, Anthony Joseph, Keith Sklower, Ron Ostrenga, and Stephen Schwab. Experience with deter: a testbed for security research. In *2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006.*, pages 10–pp. IEEE, 2006.

[7] Leandro M. Bertholdo, João M. Ceron, Wouter B. de Vries, Ricardo de Oliveira Schmidt, Lisandro Zambenedetti Granville, Roland van Rijswijk-Deij, and Aiko Pras. Tangled: A cooperative anycast testbed. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 766–771, 2021.

[8] Matthew Caesar and Jennifer Rexford. BGP routing policies in ISP networks. *IEEE Network Magazine*, 19(6):5–11, November 2005.

[9] CAIDA. AS rank. https://asrank.caida.org/, 2020. [Online; accessed 12-Oct-2021].

[10] CAIDA. CAIDA UCSD BGP community dictionary. https://www.caida.org/data/bgp-communities/, 2020. [Online; accessed 12-Oct-2021].

[11] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast CDN. In *Proceedings of the 2015 Internet Measurement Conference*, pages 531–537, 2015.

[12] Mark D Carney, Jeffrey A Jackson, Andrew L Bates, and Dante J Pacella. Method and apparatus for mitigating distributed denial of service attacks, November 24 2015. US Patent 9,197,666.

[13] R. Chandra, P. Traina, and T. Li. BGP communities attribute. Technical Report 1997, RFC Editor, 1996.

[14] Rocky KC Chang and Michael Lo. Inbound traffic engineering for multihomed ASs using AS path prepending. *IEEE network*, 19(2):18–25, 2005.

[15] Yi-Ching Chiu, Brandon Schlinker, Abhishek Balaji Radhakrishnan, Ethan Katz-Bassett, and Ramesh Govindan. Are we one hop away from a better Internet? In *Proceedings of the ACM Internet Measurement Conference*, pages 523–529, Tokyo, Japan, October 2015. ACM.

[16] Danilo Cicalese, Jordan Augé, Diana Joumblatt, Timur Friedman, and Dario Rossi. Characterizing ipv4 anycast adoption and deployment. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–13, 2015.

[17] Danilo Cicalese and Dario Rossi. A longitudinal study of IP anycast. *ACM SIGCOMM Computer Communication Review*, 48(1):10–18, 2018.

[18] Cloudflare. Famous DDoS attacks | the largest DDoS attacks of all time. https://www.cloudflare.com/learning/ddos/famous-ddos-attacks/. [Online; accessed 12-Oct-2021].

[19] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. Forecasting time series with complex seasonal patterns using exponential smoothing. *Journal of the American Statistical Association*, 106(496):1513–1527, 2011.

[20] Wouter B. de Vries, Ricardo de O. Schmidt, Wes Hardaker, John Heidemann, Pieter-Tjerk de Boer, and Aiko Pras. Verfploeter: Broad and load-aware anycast mapping. In *Proceedings of the ACM Internet Measurement Conference*, London, UK, 2017.

[21] Christoph Dietzel, Anja Feldmann, and Thomas King. Blackholing at IXPs: On the effectiveness of DDoS mitigation in the wild. In *International Conference on Passive and Active Network Measurement*, pages 319–332. Springer, 2016.

[22] Ramin Ali Dousti, Frank Scalzo, and Suresh Bhogavilli. Automated ddos attack mitigation via bgp messaging, March 22 2018. US Patent App. 15/273,510.

[23] Xun Fan and John Heidemann. Selecting representative ip addresses for internet topology studies. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 411–423. ACM, 2010.

[24] Xun Fan, John Heidemann, and Ramesh Govindan. Evaluating anycast in the domain name system. In *2013 Proceedings IEEE INFOCOM*, pages 1681–1689. IEEE, 2013.

[25] Seyed K Fayaz, Yoshiaki Tobioka, Vyas Sekar, and Michael Bailey. Bohatei: Flexible and elastic ddos defense. In *24th USENIX Security Symposium*, pages 817–832, 2015.

[26] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern CDNs. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, 2015.

[27] Ruomei Gao, Constantinos Dovrolis, and Ellen W Zegura. Interdomain ingress traffic engineering through optimized AS-path prepending. In *International Conference on Research in Networking*, pages 647–658. Springer, 2005.

[28] Vasileios Giotsas, Georgios Smaragdakis, Christoph Dietzel, Philipp Richter, Anja Feldmann, and Arthur Berger. Inferring BGP blackholing activity in the internet. In *Proceedings of the Internet Measurement Conference*, pages 1–14. ACM, 2017.

[29] T. Hardie. Distributing authoritative name servers via shared unicast addresses. Technical Report 3258, RFC Editor, 2002.

[30] Lee Hahn Holloway, Srikanth N Rao, Matthew Browning Prince, Matthieu Philippe François Tourne, Ian Gerald Pye, Ray Raymond Bejjani, and Terry Paul Rodery Jr. Mitigating a denial-of-service attack in a cloud-based proxy service, October 7 2014. US Patent 8,856,924.

[31] Chi-Yao Hong, Subhasree Mandal, Mohammad Al-Fares, Min Zhu, Richard Alimi, Kondapa Naidu B., Chandan Bhagat,

Sourabh Jain, Jay Kaimal, Shiyu Liang, Kirill Mendelev, Steve Padgett, Faro Rabe, Saikat Ray, Malveeka Tewari, Matt Tierney, Monika Zahn, Jonathan Zolla, Joon Ong, and Amin Vahdat. B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in Google's software-defined WAN. In *Proceedings of the ACM SIGCOMM Conference*, Budapest, Hungary, August 2018. ACM.

[32] Geoff Huston. BGP in 2017. https://labs.apnic.net/?p=1102, Jan 8 2018. [Online; accessed 12-Oct-2021].

[33] Team Cymru Inc. Secure Cisco IOS BGP template. https://www.team-cymru.com/secure-bgp-template.html. [Online; accessed 12-Oct-2021].

[34] Quan Jia, Huangxin Wang, Dan Fleck, Fei Li, Angelos Stavrou, and Walter Powell. Catch me if you can: A cloud-enabled ddos defense. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 264–275. IEEE, 2014.

[35] Brian Krebs. Krebsonsecurity hit with record DDoS. *KrebsOnSecurity, Sept*, 21, 2016.

[36] Jan Harm Kuipers. Anycast for DDoS. https://essay.utwente.nl/73795/1/Kuipers_MA_EWI.pdf, 2017. [Online; accessed 12-Oct-2021].

[37] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed Internet routing convergence. *ACM SIGCOMM Computer Communication Review*, 30(4):175–187, 2000.

[38] Matt Levine, Barrett Lyon, and Todd Underwood. TCP anycast—don't believe the FUD. Presentation at NANOG 37, June 2006.

[39] Zhihao Li, Dave Levin, Neil Spring, and Bobby Bhattacharjee. Internet anycast: Performance, problems, & potential. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 59–73, 2018.

[40] Ziqian Liu, Bradley Huffaker, Marina Fomenkov, Nevil Brownlee, et al. Two days in the life of the DNS anycast root servers. In *International Conference on Passive and Active Network Measurement*, pages 125–134. Springer, 2007.

[41] Doug Madory and Matt Prosser. Excessive BGP AS path prepending is a self-inflicted vulnerability. Presentation at RIPE 79, October 2019.

[42] Marek Majkowski. Memcrashed - major amplification attacks from UDP port 11211. https://blog.cloudflare.com/memcrashed-major-amplification-attacks-from-port-11211/, 2018. [Online; accessed 12-Oct-2021].

[43] Tyler McDaniel, Jared M Smith, and Max Schuchard. Flexsealing bgp against route leaks: peerlock active measurement and analysis. *arXiv e-prints*, pages arXiv–2006, 2020.

[44] Stephen McQuistin, Sree Priyanka Uppu, and Marcel Flores. Taming anycast in the wild Internet. In *Proceedings of the Internet Measurement Conference*, pages 165–178, 2019.

[45] Giovane C. M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Müller, Lan Wei, and Christian Hesselman. Anycast vs DDoS: Evaluating the November 2015 root DNS event. In *Proceedings of the ACM Internet Measurement Conference*, November 2016.

[46] Priyadarsi Nanda and AJ Simmonds. A scalable architecture supporting QoS guarantees using traffic engineering and policy based routing in the Internet. *International Journal of Communications, Network and System Sciences*, 2009.

[47] Root Server Operators. Events of 2015-11-30. https://root-servers.org/media/news/events-of-20151130.txt, 2015. [Online; accessed 12-Oct-2021].

[48] Root Server Operators. Events of 2016-06-25. https://root-servers.org/media/news/events-of-20160625.txt, 2016. [Online; accessed 12-Oct-2021].

[49] Craig Partridge, Trevor Mendez, and Walter Milliken. Host anycasting service. Technical Report 1546, RFC Editor, 1993.

[50] The Canadian Press. Canadian communications company voip.ms hit by cyber attack. https://www.thestar.com/business/2021/09/21/canadian-communications-company-voipms-hit-by-cyber-attack.html/, 09 2021.

[51] LANDER project. Lander:b root anomaly-20170306. https://ant.isi.edu/datasets/readmes/B_Root_Anomaly-20170306.README.txt, 2019. [Online; accessed 12-Oct-2021].

[52] Bruno Quoitin, Cristel Pelsser, Olivier Bonaventure, and Steve Uhlig. A performance evaluation of BGP-based traffic engineering. *International journal of network management*, 15(3):177–191, 2005.

[53] Bruno Quoitin, Cristel Pelsser, Louis Swinnen, Olivier Bonaventure, and Steve Uhlig. Interdomain traffic engineering with BGP. *IEEE Communications magazine*, 41(5):122–128, 2003.

[54] RIPE. Measurements. https://atlas.ripe.net/measurements/10310/. [Online; accessed 12-Oct-2021].

[55] RIPE. Root dns observations. Measurement ID 1009 (A-Root), 1010 (B-Root), etc., 2021.

[56] RIPE Network Coordination Centre. RIPE - Routing Information Service (RIS). https://https://www.ripe.net/analyse/internet-measurements/routing-information-service-ris, 2020.

[57] ASM Rizvi, Joao Ceron, Leandro Bertholdo, and John Heidemann. Anycast agility: Adaptive routing to manage ddos. *arXiv preprint arXiv:2006.14058*, 2020.

[58] ASM Rizvi, John Heidemann, and Jelena Mirkovic. Dynamically selecting defenses to DDoS for DNS (extended). Technical Report ISI-TR-736, USC/Information Sciences Institute, May 2019.

[59] Sandeep Sarat, Vasileios Pappas, and Andreas Terzis. On the use of anycast in DNS. In *Proceedings of 15th International Conference on Computer Communications and Networks*, pages 71–78. IEEE, 2006.

[60] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *Proc. ACM CoNEXT*, Orlando, FL, December 2019.

[61] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V. Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with Edge Fabric: Steering oceans of content to the world. In *Proceedings of the ACM SIGCOMM Conference*, pages 418–431, Los Angeles, CA, USA, August 2017. ACM.

[62] Ricardo de O. Schmidt, John Heidemann, and Jan Harm Kuipers. Anycast latency: How many sites are enough? In *International Conference on Passive and Active Network Measurement*, pages 188–200, Sydney, Australia, March 2017.

[63] Thomas Bradley Scholl. Methods and apparatus for distributed backbone internet ddos mitigation via transit providers, February 3 2015. US Patent 8,949,459.

[64] A. Shaikh, R. Tewari, and M. Agrawal. On the effectiveness of

DNS-based server selection. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 3, pages 1801–1810 vol.3, 2001.

[65] AX Sharma. Phone calls disrupted by ongoing ddos cyber attack on voip.ms. https://arstechnica.com/gadgets/2021/09/canadian-voip-provider-hit-by-ddos-attack-phone-calls-disrupted/, 09 2021.

[66] AWS Shield. Aws shield - threat landscape report – q1 2020. https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf, 08 2020.

[67] R. B. da Silva and E. Souza Mota. A survey on approaches to reduce BGP interdomain routing convergence delay on the Internet. *IEEE Communications Surveys & Tutorials*, 19(4):2949–2984, 2017.

[68] Daniel Smith. The growth of DDoS-as-a-service: Stresser services. https://blog.radware.com/security/2017/09/growth-of-ddos-as-a-service-stresser-services/, 2017. [Online; accessed 12-Oct-2021].

[69] Donald J Smith, Michael Glenn, John A Schiel, and Christopher L Garner. Network traffic data scrubbing with services offered via anycasted addresses, May 24 2016. US Patent 9,350,706.

[70] Jared M. Smith and Max Schuchard. Routing around congestion: Defeating DDoS attacks and adverse network conditions via reactive BGP routing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 599–617. IEEE, 2018.

[71] Job Snijders. Practical everyday bgp filtering with as_path filters: Peer locking. *NANOG-67, Chicago, June*, 2016.

[72] Oliver Spatscheck, Zakaria Al-Qudah, Seunjoon Lee, Michael Rabinovich, and Jacobus Van Der Merwe. Multi-autonomous system anycast content delivery network, December 10 2013. US Patent 8,607,014.

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

[74] One Step. BGP community guides. https://onestep.net/communities/. [Online; accessed 12-Oct-2021].

[75] Eric Sven-Johan Swildens, Zaide Liu, and Richard David Day. Global traffic management system using IP anycast routing and dynamic load-balancing, March 8 2011. US Patent 7,904,541.

[76] Renata Teixeira, Steve Uhlig, and Christophe Diot. BGP route propagation between neighboring domains. In *International Conference on Passive and Active Network Measurement*, pages 11–21. Springer, 2007.

[77] University of Oregon. Route Views Project. http://www.routeviews.org/routeviews/, 2021.

[78] USC/ISI. Usc/isi ant datasets. https://ant.isi.edu/datasets/all.html, 2019. [Online; accessed 12-Oct-2021].

[79] Lan Wei and John Heidemann. Does anycast hang up on you? In *2017 Network Traffic Measurement and Analysis Conference (TMA)*, pages 1–9, Dublin, Ireland, July 2017. IEEE.

[80] Fernanda Weiden and Peter Frost. Anycast as a load balancing feature. In *Proceedings of the 24th international conference on Large installation system administration*, pages 1–6. USENIX Association, 2010.

[81] Curt Wilson. Attack of the Shuriken: Many hands, many weapons. https://www.arbornetworks.com/blog/asert/ddos-tools/, 2012. [Online; accessed 12-Oct-2021].

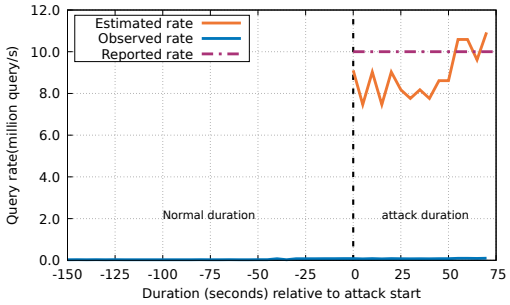


Figure 12: Estimating real-world attack events: estimating June 2016 event with 0.91% access fraction.

Appendix A Anycast and Verfploeter

IP anycast is a routing method used to route incoming requests to different locations (sites). Each site uses the same IP address, but at different geographic locations. Anycast then uses Internet routing with BGP to determine how to associate users to different sites—that is known as site's anycast *catchment*. BGP has a standard path selection algorithm that considers routing policy and approximate distance [8].

Operators can influence the routing decisions process using different traffic engineering techniques (TE) to manipulate BGP. We describe TE techniques in §3.4.1 and how they can be used to rebalance the load during a DDoS attack.

We use Verfploeter [20] to find out the client to anycast site mapping. Using Verfploeter we build our BGP playbook with various BGP changes (§6.4). The main intuition behind Verfploeter is to send pings to millions of address blocks [23, 78], using an anycast prefix as source address. The replies to these pings will be routed to the nearest anycast site by the inter-domain routing system from which we can map address blocks to the anycast sites.

Appendix B Operator Assistance System

To assist operators (§3.4.3), we provide an interface for defense. To react and reconfigure the anycast network, the operators can use a web interface similar to an equalizer, choosing the percentage of load to be increased or dropped at an anycast site. The possible ranges of slider positions are based on the playbook alternatives or presets of routing policies. This process hides the playbook complexity from the operator, making the process less error-prone and more intuitive, but still giving the operator a full control of the BGP routing.

In Figure 3 we can visualize a snapshot of this interface. Each slider represents an anycast site and each site has predetermined settings indicated by "notches". The positions of the "notches" are the results of all the measurements obtained to create our playbook. The bar graph shows the results of the measurement process, indicating how many networks will be attracted to each anycast site. The operators can visualize the forecasted traffic to each position and then apply the configuration on the production network.

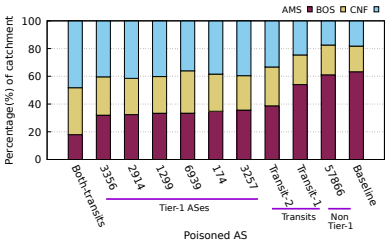


Figure 13: `Peering`: Impact of path poisoning (from AMS on 2021-04-09).

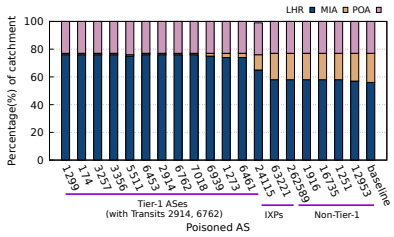


Figure 14: `Tangled`: Impact of path poisoning (from MIA on 2021-04-11).

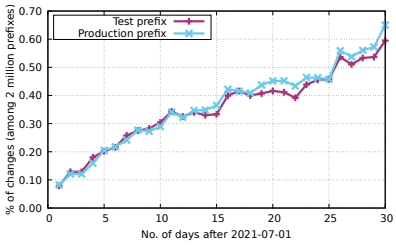


Figure 15: One month of catchment stability in `B-root`.

Policy / Day		AMS(%)				BOS(%)				CNF(%)			
		00 GMT	06 GMT	12 GMT	18 GMT	00 GMT	06 GMT	12 GMT	18 GMT	00 GMT	06 GMT	12 GMT	18 GMT
Baseline	Day-1 load	77	84	84	84	10	8	8	7	13	8	8	9
	Day-2 load	77	84	84	80	10	8	7	9	13	8	9	11
	Catchment			68				15				17	
1xPrepend AMS	Day-1 load	43	49	49	58	18	20	18	13	39	32	33	29
	Day-2 load	43	46	46	50	18	18	18	18	39	36	36	32
	Catchment			37				25				38	
1xPrepend BOS	Day-1 load	78	85	83	83	4	3	4	3	18	12	13	14
	Day-2 load	78	85	83	79	4	4	4	4	18	12	13	16
	Catchment			70				7				23	
1xPrepend CNF	Day-1 load	83	88	87	87	11	10	9	8	6	2	3	5
	Day-2 load	83	89	87	85	11	9	9	10	6	2	4	5
	Catchment			77				19				4	
Transit-1	Day-1 load	88	93	92	91	5	4	5	3	6	3	4	5
	Day-2 load	88	93	92	90	5	4	4	5	7	2	4	6
	Catchment			38				24				38	

Table 8: Load distribution with `Peering` catchment and `B-root` load. Catchment: 2020-02-24, Load: 2020-02-25 and 2020-02-26 (only showing selected policies). Catchment distribution remains similar over the course of the day showing by a single value.

Appendix C Detailed Attack Size Estimation

We validate our attack estimation with both testbed experiments and real-world events.

C.1 Testbed Experiment

We validate our model with experiments in a testbed (DETER [6]) where we can control all factors, where actual offered load is estimated and topology is fixed.

We provide the details of our testbed experiment in the extended version of this paper [57].

C.2 Case Studies: 2016-06-25 Event

We showed real-world case studies in §4.1. Here we show that our approach works for another event from 2016-06-25 (Figure 12). We observe our estimation (varying orange line) is close to the reported line (dashed purple line). We can also see that our observation is only a tiny fraction of the true offered load (bottom blue line).

Both these results show the effectiveness of our approach with both testbeds and real-world events.

Appendix D BGP Poisoning Granularity

With poisoning coverage limited by filters (§6.3.1), we next examine what granularity control it provides. We expect to see limited range since we cannot poison Tier-1 ASes, and small ASes carry little traffic.

We test path poisoning in both `Peering` and `Tangled` using three sites from each testbed. As expected, we observe the same traffic distribution when we poison any Tier-1 AS—30-35% load at AMS (`Peering` in Figure 13) and 1-3% load at MIA (`Tangled` in Figure 14).

When we poison a non-Tier-1 AS that is more than one hop away, we observe a small change in the traffic distribution. In `Peering`, we can see that poisoning AS57866 reduces a small fraction of traffic from AMS (Figure 13). We observe a similar outcome in `Tangled` (Figure 14).

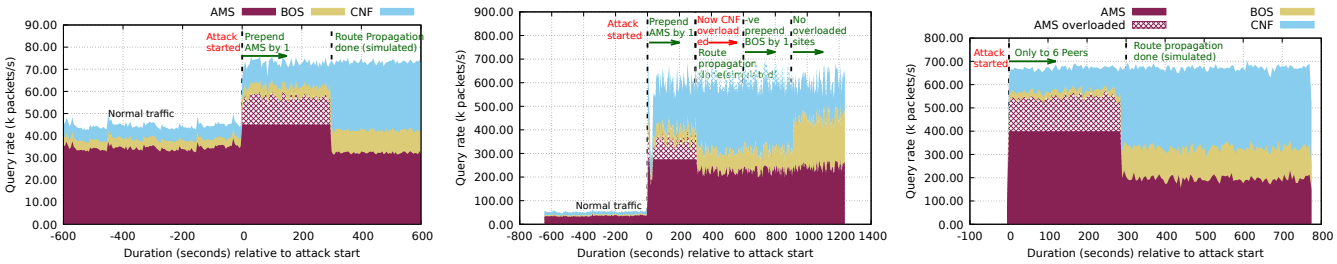
Our results prove that poisoning Tier-1 ASes is limited by the filters, and poisoning non-Tier-1 ASes that are multi-hops away can change only a small fraction of traffic.

Poisoning an immediate upstream is equivalent to not announcing to them, so we do not consider that case here.

Appendix E Catchment Stability

Our insight is that we can use a playbook for several days or weeks since the catchment remains stable over the time (§7.3). To test this we use one month of `B-root` catchment mapping with test and production prefixes. We observe the stability in `B-root` catchment.

From Figure 15, we can see that the catchment remains stable over time. In two weeks, only 0.35% prefixes, and in one month, only 0.65% prefixes changed their catchment when we compare the catchment with day 1 considering ∼2



(a) A 2020 event at `B-root` defended using positive prepending.

(b) A 2021 event at `B-root` defended using negative prepending.

(c) A 2021 event at the Dutch National Scrubbing Center mitigated using community strings.

Figure 16: Different attacks with various responses (extended).

million prefixes. This shows only a tiny fraction of prefixes changes the catchment even after a month irrespective of the changes made by the ASes. Hence, building the playbook once every week/month should be sufficient.

We also make the catchment mapping at different times of the day. We found catchment distribution remains similar at different times of the day.

Appendix F Load Distribution

Our playbook with catchment (§6.4) distribution gives an adequate prediction of traffic distribution which we successfully apply in §8. Since services care about load, we want to see how the load is distributed in different routing changes. An operator can simply make the load playbook based on the already computed catchment mapping without making additional BGP announcements.

In Table 8, we can see different routing changes and their impacts over load distribution in different times of the day. Load changes over the day—fewer load at 00 GMT in AMS site since most Europe sleeps at that time. BOS and CNF receive more load at 00 GMT as that is a busy hour for these two regions. We can also observe that some prefixes contribute more load due to the difference in number of clients behind each prefix. For this reason, BOS prefixes (mostly North American prefixes) contribute less load compared to the prefixes at other two sites. We can also see that load remains stable at the same time of different days (varies within 5% most of the time).

We can also see that the relative catchment distribution follows the load distribution, however, it is not exactly the same. Decisions will be even better when an operator considers different load playbooks at different times of the day. Building multiple load playbooks is simple since we can just use the same catchment mapping (catchment mapping remains stable (Appendix E)).

Appendix G More Attacks And Mitigation

We evaluate more attack events captured at `B-root` and at the Dutch National Scrubbing Center. We follow the same methodology mentioned in §8. We use the same playbook built with AMS, BOS and CNF sites (§6) from `Peering`.

A 2020 volumetric attack at `B-root`: We observed an ephemeral volumetric event at `B-root` on 2020-02-14 where the attackers used a single query name—peacecorps.gov. This event lasted very briefly for 3 minutes. In practice, no routing approach can work against such short-lived attacks due to the propagation delay of BGP. We stretched the event with similar traffic rate so that we can see the impact if the attack continues for more time.

In this event also, AMS is overloaded with 60k packets/s when the assumed capacity is 40k packets/s (Figure 16a). We prepend AMS by 1 so that the traffic shifts away from AMS. After 300 s, we can see no overloaded striped area in AMS.

These volumetric attacks are common at root servers. Routing based approaches can defend against such attacks.

A 2021 `B-root` event where our system iterates: We evaluate another event at `B-root` occurred on 2021-05-28. In this event, the queries were IP fragmented (large packet size), and the common query name was pizzaseo.com (we stretched the event since it was short-lived). When the attack started, our system finds AMS site overloaded (Figure 16b). Our system finds prepending from AMS is the best approach to reduce traffic from AMS. However, after prepending AMS by 1, CNF site gets the most redirected traffic, and becomes overloaded. Redirected attack sources prefer CNF over BOS. When our system finds CNF site overloaded, it deploys an approach that will reduce traffic from CNF since it is now overloaded. Our system deploys negative prepending to push more traffic towards BOS site. After 900 s, we can see there is no overloaded site. This event shows how our system can gradually find out the best routing approach.

Defending with community strings: We next consider an attack observed at the Dutch National Scrubbing Center on 2021-08-27. This attack was a volumetric DNS amplification.

In this attack, AMS is overloaded. Consulting the playbook, we select a response using community strings to shift traffic, retaining six IXP peers at AMS, while dropping all other peers and transits. The impact of this change is visible at 300 s in Figure 16c, as the attack is successfully spread across all sites.

This example shows how different community strings provide control over traffic distribution. We show more events in the extended version of this paper [57].

H Artifact Appendix

H.1 Abstract

In this artifact, we provide datasets and software tools related to our paper "Anycast Agility: Network Playbooks to Fight DDoS" [9]. Our artifact contains several datasets generated from our anycast experiments and analysis. Our datasets provide a snapshot of the results that we generated during our experiments. Some of our experimental results are dependent on the current state of the network interconnections and policies. However, due to the anycast stability, we expect to get similar results if we redo the experiments now. Our published datasets support our key results and are publicly available. We also provide tools and scripts that can be useful for other researchers.

H.2 Artifact check-list (meta-information)

- **Algorithm:** We provide an algorithm to select the best routing option from a BGP playbook containing multiple routing options and their impacts over traffic distribution (Section 3.4.2 of the paper [9]). We provide a working Python script for this selection algorithm. We include instructions about this tool in our anygility tool page [10].
- **Compilation:** We use shell/python script and java program for our tools. One needs to install Python and Java to run our tools. We depend on Verfploeter software, and we mention a series of other dependencies in the software READMEs [10].
- **Binary:** Some of our tools require extra binary files. We include those binary files with our software package, and provide instructions.
- **Data set:** We provide several datasets generated from our experiments and analysis [11]. Some software tools require extra datasets to run (e.g. IP hitlist). We include a sample dataset file with the software tool.

  However, we do not include large data files with our software tools. But these datasets can be downloaded separately (we provide the instruction in §H.3.1).
- **Run-time environment:** We tested our tools in Linux operating system. Peering toolbox on Fedora 34, and Tangled on Ubuntu 18.04 LTS and macOS 12. In some cases, our tools require root access. Our tools notify the users when it needs root access.
- **Run-time state:** Our key idea related to network playbook (Section 3.1 and 6.4 of the paper [9]) is dependent on the network interconnections and policies. We include the dates of experiments in our datasets. Since anycast is stable, we expect a similar outcome if we rerun the experiment.
- **Execution:** Some of our tools might need a long time to run. For example, our automated playbook builder announces different routing configurations, runs Verfploeter, and captures traces after a fixed interval. If we consider the whole process from measurement to playbook for 7 sites, it takes around 27-35 hours. For 3 sites it was around 17-24 hours. If we have more sites, or more routing policies, it would take even more time.
- **Security, privacy, and ethical concerns:** In the required cases, we anonymize IP addresses to prevent IP disclosure. As an example, we anonymize IP addresses in the DDoS attack datasets. For privacy reasons, we restrain ourselves from sharing certain attack data from Dutch national scrubbing center, and from an enterprise.
- **Metrics:** We provide datasets related to anycast catchments and DDoS attacks. Each dataset reports different metrics. We provide the details of these metrics in our README files. Our README files are included with the dataset packages.
- **Output:** We provide experimental outputs from Tangled and Peering testbeds. Tangled provides the measurement output in csv format while Peering provides raw captured traces in pcap format. These data files are parsed to generate output files in human-readable formats or graphs. The graphs are built using jupyter notebook and gnuplot scripts. We provide these scripts in our dataset webpage [11].
- **Experiments:** We provide scripts to automatically announce different routing configurations in both Peering and Tangled testbeds. We provide our generated datasets from these experiments. We provide some sample data to test our route selection process independent from running the whole measurement process.
- **How much disk space required (approximately)?:** Software tarballs are about 500KB. Our datasets related to the anycast experiments require around 100 GB disk space. Our attack datasets are large since we provide the whole day traffic captures (around 500 GB each). As our datasets are large, a user can download a portion of the datasets.
- **How much time is needed to complete experiments (approximately)?:** Some of the experiments may take a whole day (building a playbook with all routing options). Measurement process can take days depending the chosen measurement. Our decision maker can take decision within seconds. Parsing tools may need different times depending on the data size.
- **Publicly available (explicitly provide evolving version reference)?:** Our evolving datasets and software tools are publicly available at https://ant.isi.edu/datasets/anycast/anycast_against_ddos/index.html.
- **Code licenses (if publicly available)?:** Our tools are free; so anyone can redistribute it and/or modify it under the terms of the GNU General Public License, version 2, as published by the Free Software Foundation. We include this license notice with every tools that we make publicly available.
- **Data licenses (if publicly available)?:** We follow the data sharing policy through the participation of the LACREND project in the DHS IMPACT program [5].
- **Archived (explicitly provide DOI or stable reference)?:** Our stable reference for this artifact is here: https://zenodo.org/record/6473023 with DOI 10.5281/zenodo.6473023.

H.3 Description

We provide datasets and tools for measuring anycast agility against DDoS. Our datasets are available upon request [5]. We provide datasets about the traffic distribution after BGP changes in testbeds, attack data from a DNS root server and from a national scrubbing

Software tools	Software dependencies	Software source	Dataset dependencies	Dataset source
Traffic Estimator	Java	openjdk-11.0.13	pcap traces	With dataset
	tshark	Wireshark	RIPE IPs	Included
playbook builder	Access to Peering	Required Testbed access	Hitlitst	With dataset
	Pinger	Provided + open source		
playbook tuner	Python	Python 3.10.2	Playbook	Included
			Load	Included
load_parser+ ParsingLoad	shell+Java	openjdk-11.0.13	Dataset dir. with pcaps	With dataset
	pingextract	Provided + open source	Load file	With dataset
BGPTuner	Python bgptuner-requirements.txt	Python 3.8	Playbook with specific site list	Included
measurement scripts + tangler-cli	Bash Python Verfploter ExaBGP Access to Tangled	Bash 4.4 Python 3.8 Verfploter 0.1.42 ExaBGP 4.1.2	—	—
vp-cli	Python	Python 3.8	Verfploter 0.1.42 files	Included
make-playbook	Python	Python 3.8	stats files	Included
run-playbook	Python ExaBGP Access to Tangled	Python 3.8 ExaBGP 4.1.2	Routing Playbook	Included

Table 1: Software tools dependencies.

center, other data related to anycast catchment stability, and other supporting data for our software tools. We provide codes for traffic estimation, for reproducing experiments, and for parsing the collected data.

H.3.1 How to access

Our datasets are available from the institutional storage system [6]. We provide the datasets based on requests [5]. After getting a request, we provide the download instructions. Our software tools will be available to download from its own webpage [10].

H.3.2 Hardware dependencies

Our whole uncompressed datasets size is over 1 TB. However, a user can download the partial datasets [6]. An interested user may want to look over the meta data of each dataset (using the README files), and keep the required amount of free storage.

H.3.3 Software dependencies

We provide several tools for different purposes [10]. We tested our software tools in Linux operating system. Some of our tools are dependent on external data sources and binaries. In most cases, we provide a sample data source with the package, and for other cases one can download the datasets with our released dataset. We provide the required binaries with our tools. One might need to install dependencies like Python or Java. We detail dependencies on (Table 1).

H.3.4 Data sets

We provide a full list of datasets in our web page [11].

We release datasets related to catchment distribution after routing configuration changes. We announce different BGP options, run Verfploeter to ping millions of responsive targets, and then capture the responses at every site. Our dataset includes raw pcap files captured from these measurements, and parsed data files in human-readable format.

We also provide DDoS attack data collected from B-root and Dutch national scrubbing center from 2015 to 2021.

Within other datasets, we provide datasets for anycast stability, and other supporting datasets to run our software tools.

The READMEs for these datasets are available with the dataset package.

H.3.5 Models

N/A

H.3.6 Security, privacy, and ethical concerns

We see no privacy concerns with our shared datasets. In cases like the DDoS attack data, we only share the /24 prefixes to hide the exact IP.

H.4 Installation

Instructions for running the tools are available in the webpages [10].

H.5 Evaluation and expected results

We provide the key results of the paper by mentioning the figures and tables, and list the corresponding datasets and tools in Table 2. Next, we list the key results, then we describe how can one get these results, and possible variations in the results. Please check the detailed steps to regenerate the graphs from the provided datasets.

H.5.1 Results with traffic estimation:

We propose a new technique to estimate the true offered load when we have loss in the upstreams (Section 3.3 and 4 [9]). We show our traffic estimation technique works well with the real world-attack events. For traffic estimation, we provide a tool named *TrafficEstimator* [14]. Using our traffic estimation tool, we show that we can correctly estimate the true offered load for real-world DDoS events.

To reproduce the same result, one needs to feed the attack traces to our program (provided as attack data in peering dataset [12]). One needs to have the pcap traces that we used, and needs to install tshark (with Wireshark) to feed the traffic content to our program. We used tracefiles for 2015-11-30 and 2016-06-25 events. A user needs to know the attack start time to use the right pcap files to observe the estimation outputs. The provided README tells the attack start time. We also need to provide a list of RIPE IPs that our program will use (already provided with the tool). We provide the instructions for running this tool in our webpage [14].

If running correctly, one can regenerate the same results that we showed in the paper. Depending on the start and end time of the attack trace, we might get a slightly different estimation. But on average we expect to get the same results.

Detailed steps:

We show the results generated for 2015 and 2016 events. This covers Figure 4, Figure 12, and Table 1. We use the following datasets:

1. Non-attack traffic 2015: B_Root_Anomaly-20151130/29/20151129-065024-00175689.pcap.xz,
2. Non-attack traffic 2016: B_Root_Anomaly-20160625/24/20160624-200008-00356777.pcap.xz,
3. Attack traffic 2015: B_Root_Anomaly-20151130/30/20151130-065209-00177422.pcap.xz,
4. Attack traffic 2016: B_Root_Anomaly-20160625/25/20160625-221823-00357641.pcap.xz,
5. RIPE IPs 2015: ripe-ips-2015-11-30.txt (provided with the tool),
6. RIPE IPs 2016: ripe-ips-2016-06-25.txt (provided with the tool).

The first step is to calculate the RIPE traffic rate during normal period (known-good traffic - normal column of Table 1). To get this value, we feed non-attack traffic to our estimator to get the RIPE traffic rate during normal period. We use the following command to get this:

For 2015 event: xzcat B_Root_Anomaly-20151130/29/20151129-065024-00175689.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2015-11-30.txt 192.228.79.131,2001:500:84::9077:f4f0

For 2016 event: xzcat B_Root_Anomaly-20160625/24/20160624-200008-00356777.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2016-06-25.txt 192.228.79.62,2001:500:84::ad9b:d590

Please wait for some time to see the generated output in the command prompt. The given addresses (192.228.79.* and 2001:500:84::*) are the B root server addresses (different because of the different anonymization keys). This command will generate an output like:

2015: “Time diff: 5.01 Counter-packets: 193 Rate: 38.47” 2016: “Time diff: 5.06 Counter-packets: 195 Rate: 38.52”

Key results [9, 11]	**Shared datasets**	**Related tools**
Figure 3	sample dataset provided with the tool	*tangled tools [15]* bgp-tuner
Figure 4, Table 1, Figure 12	*peering and root DNS dataset [12]* B_Root_Anomaly-20151130 B_Root_Anomaly-20160625	*TrafficEstimator and selection tools [14]* TrafficEstimator
Figure 5	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-24	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 6	*tangled dataset [13]* Usenix_anygility_5_sites_2022-03-24_NEW	*tangled tools [15]* measurement scripts tangler-cli, vp-cli Anygility-Tangled-Catchment-load-distribution.ipynb
Figure 7	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224, community (3 sites) 2020-02-25	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 8	*tangled dataset [13]* community dataset (3 sites)	*tangled tools [15]* measurement scripts tangler-cli, vp-cli
Table 5, Table 6	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-24, community strings (3 sites) 2020-02-25, poisoning (3 sites) 2021-04-09	*peering tools [15]* load_parser ParsingLoad
Figure 9	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3 sites) 2020-02-28	*peering tools [15]* playbook_builder load_parser ParsingLoad
Figure 10	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: prepending (3,5,7 sites) 2020-02-24, 2020-04-07, 2020-04-08 Community (3, 5, 7 sites) 2020-02-25 and 2020-04-19	*peering tools [15]* playbook_builder load_parser ParsingLoad
Table 7	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: baseline (3 sites) 2020-02, 2020-04, and 2020-06	*peering tools [15]* load_parser ParsingLoad
Figure 11	*peering and root DNS dataset [12]* B_Root_Anomaly_message_question-20170306	*peering tools [15]* ParsingLoad TimeBasedPrefixLoad AnycastSiteLoad
Figure 13	*peering and root DNS dataset [12]* anycast_catchment_distribution-20200224: poisoning (3 sites) 2021-04-09	*peering tools [15]* load_parser ParsingLoad
Figure 14	*tangled dataset [13]* poisoning dataset (3 sites)	*peering tools [16]* measurement scripts tangler-cli, vp-cli
Figure 15	*peering and root DNS dataset [12]* anycast_catchment_stability-20210701	-
Figure 16	*peering and root DNS dataset [12]* B_Root_Anomaly_message_question-20200214 B_Root_Anomaly_message_question-20210528	*peering tools [15]* ParsingLoad TimeBasedPrefixLoad AnycastSiteLoad

Table 2: Paper key results with datasets and tools. We provide the scripts to generate the graphs for our key results in our webpage [11].

We waited until 5 s to fix the final rate of the RIPE IPs. This rate is the cumulative rate measured from the start time. known-good traffic - normal column from Table 1 has a similar value.

The second step is to run the same TrafficEstimation java utility to find the estimated rate. We run the following commands for this:

2015 event: xzcat B_Root_Anomaly-20151130/30/20151130-065209-00177422.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2015-11-30.txt 192.228.79.131,2001:500:84::9077:f4f0 38.47

2016 event: xzcat B_Root_Anomaly-20160625/25/20160625-221823-00357641.pcap.xz | sudo tshark -r - -T fields -e frame.time_epoch -e ip.src | java -jar TrafficEstimator.jar ripe-ips-2016-06-25.txt 192.228.79.62,2001:500:84::ad9b:d590 38.52.

Please note that this command has an extra parameter (38.47 and 38.52) which we got from the previous command outputs. This command will generate two types of output lines. For 2015 event, we are showing a snapshot after 20 s, and for 2016 event we are showing a snapshot after 42 s.

2015 event output:

Time diff: 19.99 Counter-packets: 37 Rate: 1.85 1448866349.106 Count-packets: 1604914 Observed rate: 320982.8 Estimated: 6674713.88

2016 event outptut:

Time diff: 41.98 Counter-packets: 14 Rate: 0.33

1466893148.1316 Count-packets: 451957 Observed rate: 90370.72 Estimated: 11186300

Our program shows the RIPE rate when it finds new RIPE IPs in DNS traffic (starting after 1 minute). The observed rate line is printed at every 5 s. So, the users normally observe more number of first line.

The first line for 2015 event indicates that after 20 s during the attack period, our program receives 37 RIPE packets at a rate of 1.85 RIPE packets/s. This value corresponds to the known good traffic - observed column value from Table 1. Dividing by the prior normal rate of 38.47, we get the access fraction value, α. The first line for 2016 event indicates that the program gets 14 RIPE packets within 41.98 s with a rate of 0.33 RIPE packets/s. This value indicates the known good traffic - observed column value from Table 1. Dividing by the prior rate of 38.52 RIPE packets/s, we get the value of access fraction, α in Table 1. Please note that, because of a different RIPE IP list and measurement start time (using different pcap files), we are getting a slightly different value than what we have in the table.

Our program generates the second line at every 5 s. This line indicates the timestamp at every 5 s, packet count within that 5 s, the observed rate (packet count / 5.0), and the estimated traffic rate (observed rate / α). This observed rate corresponds to the offered load during attack - observed rate column of Table 1. For 2015 event, the sample output value is close to 0.32M packets/s, and for 2016 event this value is 0.09M packets/s. These two values are similar to what we have in Table 1 (0.37 and 0.10). A user will observe variable rates at different times. This observed rate is then divided by the calculated access fraction (α) to get the estimated offered load—offered load during attack - estimated column ($\sim$6.6M queries/second for 2015 event and $\sim$11M queries/second for 2016 event), which is close the reported rate of 5M queries/second and 10M queries/second, respectively [7, 8]. We use the estimated values from our TrafficEstimation program to generate the graphs—Figure 4 and Figure 12. Depending on the attack start time and RIPE IPs, the estimated values may vary slightly but we expect to get a similar trend. The offered load during attack - normal column indicates the normal traffic rate at a given time which we can measure from B root traffic (TrafficEstimator tool can measure this; we just need to feed the normal traffic with the known RIPE rate parameter) but we are skipping this detail since it is not directly related to the key outcomes. $\hat{\alpha}$ is calculated by dividing observed rate by the reported rate.

Our outcomes for known-traffic measurement, and estimated rate measurement may vary depending on the RIPE IPs we used and the traffic data we are using. We tried 5 s of traffic to find out the known traffic rate. This choice is arbitrary, a user can wait for some more time. Given the RIPE IPs that we provided, a user may expect to see 25-50 RIPE queries per second. Please note that, we used a subset of RIPE IPs. A larger RIPE IP set along with their consistent signal would ensure more stable RIPE query rate. We also provided some snapshots for the estimated rate measurement. Please note that, they are just snapshots. Estimated rates are dependent on the observed traffic rates (always varying), and the access fraction.

H.5.2 Building BGP playbook:

We propose a BGP playbook to fight against DDoS attacks. We build the BGP playbook with different routing options and their impacts over traffic distribution (Section 6 and 7 [9]). We show that BGP playbook can help the operators to select the right routing option during an attack event, and a playbook can provide a granular control over traffic distribution.

To reproduce the result, a user needs to announce different BGP configurations, and then run Verfploeter/pinger [3] to learn the prefixes to anycast site catchment. We provide scripts (*playbook_builder* in Peering and *tangler-cli* in Tangled) for our testbeds to makes these announcements automatically [15, 16]. One needs to have access to the testbeds to run this experiment. We used `Peering` [17] and `Tangled` [2] testbeds. These testbeds authorize an anycast prefix for a specific time period. One needs to ask for permission with a proposal to use these testbeds [1, 18]. Our script is dependent on verfploeter/pinger tool which is available online [3], and we provide a binary. This tool needs a target hitlist of IPs which we provided with our dataset (search for internet_address_history_it88w20191127 [6]). We provide a tool named getting_hitlist_ips to parse this raw hitlist file to get the list of responsive IPs. The instruction to run these tools is available in our webpage [15, 16].

To validate our results, we also provide the datasets that we got from our experiments. We include captured pcap files, and data in human-readable format for Peering [12], and in csv format for Tangled [13]. To reproduce results from the collected data, we also provide tools called *load_parser* and *ParsingLoad* in Peering [15], and *measurement scripts* in Tangled [16].

Our result is dependent on the stability of the network state. Since anycast catchment is fairly stable, we expect to get a slight variation but similar results if we rerun the experiment.

Detailed steps: We provide an example here to reproduce Figure 5 from our paper. Other similar graphs and tables like Figure 5—Figue 7, Figure 8, Table 5, Table 6, Figure 9, Figure 10, Table 7, Figure 13, Figure 14 can be generated using the similar process. Please note that figures for community strings and path poisoning (Figure 7, Figure 10, and Figure 13) for `Peering` utilizes only ParsingLoad utility alone (we provide the details later in this subsection).

At first, one needs to run *playbook_builder* tool to make BGP announcements for every prepending option. This step is dependent

on getting access from the Peering testbed. Also, Internet routing changes, and we will not get the same outputs that we received while doing the experiment. As a result, we provide the collected data in pcap form to skip this step. Please find this dataset in peering and root DNS dataset—prepending (3 sites) 2020-02-24. The other datasets for other figures mentioned in prior paragraph are also provided.

To recreate Figure 5, we provide the following datasets:

1. The pcap files in peering and root DNS dataset: anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224,
2. The IP hitlist internet_address_hitlist_it88w-20191127/internet_address_hitlist_it88w-20191127.fsdb.bz2,
3. Some "load" data, provided with the software tool (we consider catchment in this figure so a full load data is not important).

After having these data, one needs to run anygility-peering/src/getting_histlist_ips/getting_hitlist_ips on the hitlist:

bzcat /data/internet_address_hitlist_it88w-20191127/internet_address_hitlist_it88w-20191127.fsdb.bz2 | python3 ./getting_hitlist_ips/data/ip_list_20191127.txt.

This will create a text file, ip_list_20191127.txt, containing one responsive IP address per line.

Then one needs to run anygility-peering/src/load_parser/load_parser.sh on the pcaps with the generated IP hitlist and sample load-file, and its corresponding load-date (e.g. –load=. –ldate=2022-02-01 to use the one provided with the tool):

bash load_parser.sh --numbers=3 --sites=AMS,BOS,CNF --date=2020-02-24 --dir=/data/anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/ --load=.--ldate=2022-02-01 --hitlist=/data/ip_list_20191127.txt Please note that the trailing / in the –dir argument is necessary.

This will run the ParsingLoad java utility for each announcement configuration, which will

- generate .dat files with ping responses from the .pcap files using pingextract utility.
- compute catchment data, both in terms of /24-blocks and "load" and store these as .txt files inside the data directory. For each announcement configuration, two files <DATE>-catchment-percentage.txt and <DATE>-load-percentage.txt are created. In addition, a combined all-<DATE>-load-<LOAD-DATE>.txt file is created in the data root directory.

The content of all-<DATE>-load-<LOAD-DATE>.txt consists of multiple blocks of this form:

<routing-configuration-path>

- <missing /24 count> <missing /24 relative>

site_1 <site_1 /24 count> <site_1 /24 relative> <site_1 /24 relative received>

[...]

site_n <site_n /24 count> <site_n /24 relative> <site_n /24 relative received>

multiple <multiple /24 count> <multiple /24 relative> <multiple /24 relative received>

- <missing load count> <missing load relative>

site_1 <site_1 load count> <site_1 load relative> <site_1 load relative received>

[...]

site_n <site_n load count> <site_n load relative> <site_n load relative received>

multiple <multiple load count> <multiple load relative> <multiple load relative received>

Figure 5 then shows bar-graphs created from the <site_x /24 relative received> values.

Using ParsingLoad alone: The script load_parser utilizes ParsingLoad for each of the path prepending configurations. When we are not parsing path prepending configurations, we can just utilize ParsingLoad utility alone. We utilize ParsingLoad alone for community strings and path poisoning (Figure 7 and Figure 13). We run ParsingLoad for each of these routing configuration separately.

java -jar ParsingLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-ALL-PEERS/ /nfs/lander/traces/verfploeter/broot_verfploeter/Peering/Peering_Mapping/2020/community_strings/2020-02-25-AMS,BOS,CNF-AMS-ONLY-PEERS/ 2020-02-25 loads/ 2020-02-22

The output has the same format like all-<DATE>-load-<LOAD-DATE>.txt as we mentioned above. We combine these generated files to build Figure 7 and Figure 13. We use ParsingLoad separately for each routing configuration with community strings and path poisoning. But a script for all the community string and path poisoning options is also possible. For path poisoning, we used poisoning datasets (inside anycast_catchment_distribution-20200224) for AS174 (Tier-1), AS8283 (Transit-2), and AS12859 (Transit-1).

H.5.3 Selection from the playbook:

We provide a tool [14] to select the right routing configuration from the BGP playbook (Section 3.4.2 [9]). Using this tool, we show that an automated approach can be useful to select the right routing approach.

Our selection tool provides output based on the current playbook, and offered load. To show how the selection tool works, we provide a sample playbook (based on Table 5 [9]), and a load file. When the users run the tool with the given inputs, they can see the selection output. We also include a tool named *bgp-tuner* for showing the graphical interface [16].

Depending on the playbook and offered load, one can observe a different output, which can be a complete different policy selection.

Detailed steps: We provided a sample playbook and offered load file with the playbook_tuner tool. Please run the following command to see the outputs from this program:

cat load.txt | ./playbook_tuner –setup "playbook.txt"

This will result the following output:

Overloaded site: AMS

Suggested config: 1AMS, Estimated load distribution: 41292.64 29494.75 41292.64

Other configs: Poison-Tier-1, Estimated load distribution: 41292.64 29494.75 41292.64

Other configs: Poison-Tier-2, Estimated load distribution: 41292.64 29494.75 41292.64

This tells that prepending AMS by 1 would provide the best possible load distribution. Some other options are also possible.

H.5.4 Attack mitigation:

We show that BGP playbook is helpful to mitigate the real-world DDoS events.

To reproduce the same result, we provide the B-root attack traces in pcap and in message question formats [12]. Due to privacy reason,

we cannot share the attack data from the Enterprise and Dutch National Scrubbing Center. We also provide the catchment distribution for different BGP changes [12, 13]. Matching the attack prefixes and attack loads to the prefix-wise catchment gives us the traffic distribution at different sites. If one wants to test the B-root event, they need to run *TimeBasedPrefixLoad* tool to get the per prefix attack load [15]. Then one needs to run *AnycastSiteLoad* program to get the per anycast site load [15].

Since the attack and catchment mapping are fixed, we expect to get the same results that we showed in the paper.

Detailed steps: We show the detailed steps to generate Figure 11(a) here. All other subfigures of Figure 11 and Figure 16 can be generated using the similar process.

To generate Figure 11(a), we need the following datasets:

1. peering and root DNS dataset: B_Root_Anomaly_message_question-20170306/: Figure 11(a) shows 10000 s of traffic. To make the data processing faster, we recommend to use a subset of this whole timeframe. We recommend the user to download the datasets from 06:40:00 AM to 06:50:00 AM to reproduce a fraction of the whole timeframe combining both attack and non-attack period. The file names represent the dates and times (format: YYYYMMDD-HHMMSS-*).
2. peering and root DNS dataset: anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/2020-02-24-AMS,BOS,CNF/
3. peering and root DNS dataset: /anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-Transit-1-Trial-2/(update: this Trial-2 dataset is newly added. We also provided Trial-1 dataset for 2020-02-25-AMS,BOS,CNF-AMS-Transit-1 which will give a similar output, but we did not use that in the paper).

At first, run the TimeBasedPrefixLoad java utility on the downloaded message_question format data. We only need time, source IP and message length for our measurement. message_question formatted files have several attributes/columns. We used fsdb tool to retrieve the times, source IPs, and message length [4]. Please follow the instruction to install FSDB from here: https://www.isi.edu/~johnh/SOFTWARE/FSDB/perl-Fsdb-2.74_README.html. Next, use the following command to run TimeBasedPrefixLoad jar to generate the prefix-wise load for each 5 s:

xzcat B_Root_Anomaly_message_question-20170306/06/20170306-044* | dbcol time srcip msglen | java -jar TimeBasedPrefixLoad.java output-20170306/ 192.228.79.64,2001:500:84::bb26:87a2.

Here, dbcol is a utility from FSDB to select the right column from the message_question format dataset. output-20170306 will have multiple txt files named with a number indicating the time segment. This command will generate prefix-wise load at every 5 s in output-20170306 directory: <network_prefix> <number_load> <bytes>.

Then we run AnycastSiteLoad java utility to find out the per site load at every 5 s. We run this utility for two routing configurations—one without any routing change and one with announcing only to Transit-1.

java -jar AnycastSiteLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Path_Prepending_AMS,BOS,CNF-20200224/2020-02-24-AMS,BOS,CNF/ 2020-02-24 output-20170306/ 2017-03-06,

java -jar AnycastSiteLoad.jar 3 AMS,BOS,CNF anycast_catchment_distribution-20200224/Community_Strings_AMS,BOS,CNF-20200225/2020-02-25-AMS,BOS,CNF-AMS-Transit-1-Trial-2/ 2020-02-25 output-20170306/ 2017-03-06.

Please note that these two commands utilize output-20170306 that we generated in our previous step. These two commands generate two files in the corresponding catchment directory named as <CATCHMENT-DATE>-load-<ATTCK-DATE>-ingress.txt. The output format inside the file: <time> <site-1> <count-site-1> <bit-site-1> <...> <site-n> <count-site-n> <bit-site-n>. The first file contains load without any routing change, the second file contains load after announcing only to Transit-1. We combine these two files to show non-attack period (no policy deployed), and period when the route propagation is done (when we deployed Transit-1).

To match the results with the Figure 11(a), the first output file will contain (<count-site-n> column) traffic load during normal period (before 0 s from the graph with around 20k packets/s). The first output file also contains the attack traffic (AMS load over 60k packets/s after 160 s of the first file). This is similar to the traffic from 0 s to 300 s of Figure 11(a). After that we announce only to Transit-1 (after 300 s of Figure 11(a)). The second output file contains this data (after 160 s from the file).

H.6 Notes

Considering the real datasets are big, and time expensive to run, we include smaller datasets collected using a small hitlist fraction (0.1% of original size) in experiments with Tangled. While the produced playbook will differ from paper results, we believe it can help for testing purpose. For Peering tools, we sometimes include smaller sample supporting data files.

If desired, we can provide access to the Tangled testbed. Access to Peering testbed is dependent on the approval from Peering admins.

H.7 Version

Based on the LaTeX template for Artifact Evaluation V20220119.

References

[1] Tangled admins. Tangled anycast testbed. https://anycast-testbed.nl/, 2019. [Online; accessed 15-Feb-2022].

[2] Leandro M Bertholdo, Joao M Ceron, Wouter B de Vries, Ricardo de Oliveira Schmidt, Lisandro Zambenedetti Granville, Roland van Rijswijk-Deij, and Aiko Pras. Tangled: A cooperative anycast testbed. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 766–771. IEEE, 2021.

[3] Wouter De Vries. Verfploeter/pinger: Active measurement of anycast catchements. https://ant.isi.edu/software/verfploeter/pinger/index.html, 2019. [Online; accessed 15-Feb-2022].

[4] John Heidemann. John heidemann / software / fsdb. https://www.isi.edu/~johnh/SOFTWARE/FSDB//, 1991. [Online; accessed 19-Mar-2022].

[5] Analysis of Network Traffic (ANT) group. Ant dataset requests. https://ant.isi.edu/datasets/requests.html, 2022. [Online; accessed 15-Feb-2022].

[6] Analysis of Network Traffic (ANT) group. Ant datasets. https://ant.isi.edu/datasets/index.html, 2022. [Online; accessed 15-Feb-2022].

[7] Root Server Operators. Events of 2015-11-30. https://root-servers.org/media/news/events-of-20151130.txt, 2015. [Online; accessed 12-Oct-2021].

[8] Root Server Operators. Events of 2016-06-25. https://root-servers.org/media/news/events-of-20160625.txt, 2016. [Online; accessed 12-Oct-2021].

[9] A S M Rizvi, Leandro Bertholdo, João Ceron, and John Heidemann. Anycast agility: Network playbooks to fight DDoS. In *Proceedings of the 31st USENIX Security Symposium*, page to appear. USENIX, August 2022.

[10] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. anygility - anycast agility tools: playbook builder and decision maker. https://ant.isi.edu/software/anygility/index.html, 2022. [Online; accessed 2-Mar-2022].

[11] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Artifacts about anycast agility against ddos. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/index.html, 2022. [Online; accessed 2-Mar-2022].

[12] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Datasets about anycast agility against ddos in peering testbed. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/peering/index.html, 2022. [Online; accessed 2-Mar-2022].

[13] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Datasets about anycast agility against ddos in tangled testbed. https://ant.isi.edu/datasets/anycast/anycast_against_ddos/tangled/index.html, 2022. [Online; accessed 15-Feb-2022].

[14] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos. https://ant.isi.edu/software/anygility/system/index.html, 2022. [Online; accessed 2-Mar-2022].

[15] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos in peering testbed. https://ant.isi.edu/software/anygility/peering/index.html, 2022. [Online; accessed 2-Mar-2022].

[16] A S M Rizvi, Leandro Bertholdo, Joao Ceron, and John Heidemann. Tools about anycast agility against ddos in tangled testbed. https://ant.isi.edu/software/anygility/tangled/index.html, 2022. [Online; accessed 2-Mar-2022].

[17] Brandon Schlinker, Todd Arnold, Italo Cunha, and Ethan Katz-Bassett. PEERING: Virtualizing BGP at the Edge for Research. In *Proc. ACM CoNEXT*, Orlando, FL, December 2019.

[18] Peering The BGP Testbed. Peering the bgp testbed. https://peering.ee.columbia.edu/, 2019. [Online; accessed 15-Feb-2022].