Drongo: Speeding Up CDNs with Subnet Assimilation from the Client

Marc Anthony Warrior
Northwestern University
warrior@u.northwestern.edu

Uri Klarman
Northwestern University
uri.klarman@u.northwestern.edu

Marcel Flores
Northwestern University
marcelflores2007@u.northwestern.edu

Aleksandar Kuzmanovic
Northwestern University
akuzma@northwestern.edu

ABSTRACT

Currently, the attempt to choose the “best" content replica server for a client is carried out solely by CDNs. While CDNs have a decent view of load distribution and content placement, they receive little input from the clients themselves. We propose a hybrid solution, *subnet assimilation*, where the client participates in the server selection process while still leaving the final say to the CDN. Subnet assimilation allows clients to declare their own “network location," different from the actual one, which in turn drives a CDN towards making *better* decisions. To demonstrate, we introduce Drongo, a client-side system, readily deployable on existing clients without any changes to the CDNs, that employs subnet assimilation to dramatically improve replica server selection.

We implemented and extensively evaluated Drongo on a set of 429 clients spread across 177 countries and 6 major CDNs. We show that Drongo affects 69.93% of all clients, prompting better CDN replica choices which reduce the latency of affected requests by up to an order of magnitude and by 24.89% on average across six major providers, with Google's performance improving by 50% in the *median* case. Our results indicate that client participation holds great opportunities for the advancement of CDN performance.

CCS CONCEPTS

• **Networks** → *Application layer protocols*; Naming and addressing; Location based services;

CoNEXT '17, December 12–15, 2017, Incheon, Republic of Korea
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5422-6/17/12…$15.00
https://doi.org/10.1145/3143361.3143365

KEYWORDS

DNS, CDN, replica server, subnet

ACM Reference Format:
Marc Anthony Warrior, Uri Klarman, Marcel Flores, and Aleksandar Kuzmanovic. 2017. Drongo: Speeding Up CDNs with Subnet Assimilation from the Client. In *CoNEXT '17: CoNEXT '17: The 13th International Conference on emerging Networking EXperiments and Technologies, December 12–15, 2017, Incheon, Republic of Korea.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3143361.3143365

1 INTRODUCTION

Latency between clients and servers on the Internet is a key measure that fundamentally affects users' perception of the Internet speed. It directly impacts end-to-end page load time, which in turn affects user experience and business revenues [50]. Amazon has reported that every 100 ms increase in page load time costs them 1 % in sales [18]. Client-server latency is essential for other killer apps such as web search [9] and video streaming [37]. Reduced latency directly improves throughput, which has been shown both theoretically [43] and empirically [47]. It is thus no exaggeration to say that literally every single *millisecond* of reduced client-server latency on the Internet counts.

Significant efforts have been invested in an attempt to move servers closer to the clients via Content Distribution Networks (CDNs), which distribute content via hundreds of thousands of servers worldwide [1, 11]. Additionally, to minimize the client-server latency, such systems perform extensive network and server measurements and use them to redirect clients to different servers. While this significantly improves performance, it is no secret that CDNs do not always direct clients to the CDN replica that is closest in the network sense [47]. This happens for several reasons: (*i*) CDNs' mapping of the Internet isn't perfect, particularly for regions that are more distant from the core CDN infrastructure [19, 38, 47], (*ii*) CDNs aim to balance the traffic load or deploy other policies, which may conflict with minimizing the client-server latency, and (*iii*) comprehensively measuring the Internet to capture latency variations between CDN replicas and the rest of the IP space over short timescales is challenging.

In this paper, we demonstrate that client-server latency is far from optimal. We embrace a hybrid approach that enables *clients* to join CDNs in addressing the above challenges as follows: (*i*) Clients help with CDN replica mapping; indeed, it is far easier for a single client to find a *subnet* that is consistently suggested well-performing replicas, with respect to the client, than for a CDN to evaluate the entire IP space. (*ii*) Our approach respects load balancing and other CDN policies: CDNs still make all the replica selection decisions and clients respect those decisions. (*iii*) CDNs' measurement burden is shared with clients who measure CDNs and help them come up with better, more fine-grained, decisions. Most importantly, all of this is readily available *today*, with only minor client-side upgrades, without *any* changes to CDNs or to existing protocols, and without the need for broad adoption of our system.

We devise a simple heuristic and a lightweight method that helps clients realize when they are not served by a nearby CDN replica. After obtaining a replica selection from a CDN, the client traceroutes the path towards that replica, and explores if the upstream hops on the path are potentially directed to different replicas. This is done by utilizing EDNS0 client subnet extension (ECS) [32] to issue DNS requests on behalf of hops. If hops are indeed directed to different replicas, then it is possible that the latency between the client and a replica recommended to a hop is smaller than the latency between the client and the replica recommended to the client.[1] We refer to such a scenario as a *latency valley. Our goal is not to find lower-latency replicas – we aim to find* ***subnets*** *to which lower-latency replicas, relative to the client, are consistently suggested.* By ultimately leaving the *decision* and *access* systems entirely up to the CDN, we ensure that any additional policies a CDN may have (such as network access restrictions and load balancing) are unaffected.

By conducting experiments on six major content providers, we show that latency valleys are common phenomena, and demonstrate that they systematically speed up object downloads. In particular, latency valleys can be found across all the CDNs we have tested: 26%–76% of routes towards CDN replicas discover at least one latency valley. Our key practical contribution lies in showing that valley-prone *subnets* that lead to these lower-latency replicas are easily found from the client, are simple to identify, incur negligible measurement overhead, and are persistently valley-prone over timescales of days. Most importantly, we show that we can effectively leverage latency valleys via valley-prone subnets with *subnet assimilation*, an approach in which clients use ECS to improve CDN replica-mapping.

We implement and evaluate Drongo, a client-side system that leverages upstream *subnets* to consistently receive lower-latency replicas than what the client would ordinarily have been recommended. Our measurements show that Drongo can improve requests' latency by up to an order of magnitude. Moreover, we evaluate the optimal parameters to capture these latency gains, and find that 5 measurements on the timescale of days are the only requirement to maximize Drongo's gains. Using the optimal parameters, Drongo affects the replica selection of 69.93% of clients, and affected requests experience a 24.89% reduction in latency in the median case. Moreover, Drongo's significant impact on these requests translates into an overall improvement in client-perceived aggregate CDN performance.

We make the following contributions:

- We introduce the first approach to enable clients to actively measure CDNs and effectively improve their selection decisions, while requiring no changes to the CDNs, and while respecting CDNs' policies.
- We extensively analyze client-side CDN measurements and determine critical time-scales and key parameters that empower clients to leverage their advanced views of CDNs.
- We introduce Drongo, a client-side CDN measurement crowd-sourcing system and demonstrate its ability to substantially improve CDNs' performance in the wild.

2 PREMISE

2.1 Background

CDNs attempt to improve web and streaming performance by delivering content to end users from multiple, geographically distributed servers typically located at the edge of the network [1, 4, 11, 21]. Since most major CDNs have replicas in ISP points-of-presence, clients' requests can be dynamically forwarded to close-by CDN servers. Historically, one of the key reasons for systematic CDN imperfections was the distance between clients and their local DNS (LDNS) servers [23, 35, 39, 45]. This issue was further dramatically amplified in recent years (see [28]) with the proliferation of public DNS resolvers, *e.g.*, [5, 10, 12, 15, 20, 22].

In an attempt to remedy poor server selection resulting from LDNS servers, there has been a recent push, spearheaded by public DNS providers, to adopt the EDNS0 client subnet option (ECS) [42]. With ECS, the client's IP address (truncated to a /24 or /20 subnet for privacy) is passed through the recursive steps of DNS resolution as opposed to passing the LDNS server's address.

[1] We emphasize that all of our measurements are performed between the *client* and a CDN *replica*; no measurements are performed directly from upstream nodes.

However, even when the actual client's network location is accurate, numerous factors prevent CDNs from providing optimal CDN replica selection [38]. First, creating accurate network mapping for the Internet is a non-trivial task given that routing can inflate both end-to-end latency and latency variance, *e.g.*, [44]. As a result, CDN selections could be heavily under-performing (see [38] for a thorough analysis). The underlying causes are diverse, including lack of peering, routing misconfigurations, side-effects of traffic engineering, and systematic diversions from destination-based forwarding [33]. Second, CDN measurements are necessarily coarse grained – measuring each and every client (each individual IP address) from a CDN is impossible for scalability reasons and because the actual source IP address isn't available due to ECS truncation. On top of this, CDNs often have other optimization goals, *e.g.*, load balancing, which can divert clients further away from close-by replicas.

Our key idea is to enable *clients* to join CDNs in addressing the above challenges. In particular, it is far easier for a single client to find a subnet that is consistently recommended well-performing replicas than it is for a CDN to evaluate the entire IP space. Not only does the client-supported approach scale, it enables far better CDN replica selection decisions. In summary, in our approach, (*i*) clients conduct measurements to find other subnets that lead to potentially lower-latency CDN replicas, (*ii*) they evaluate the performance of such subnets and associated replicas via light, infrequent measurements over long time-scales, and (*iii*) finally, they utilize these subnets to drive CDNs' decisions towards lower-latency replicas.

2.2 Respecting CDN Policies and Incentives for Adoption

As stated above, CDNs can sometimes deploy *policies* that can prevent them from serving clients nearby replicas. In principal, there are two such scenarios. First, on longer timescales, a CDN may have a business logic where it wants a certain IP subnet, and no one else, to be able to use a certain CDN cluster or server. For example, an ISP may allow a CDN to deploy a CDN server inside its network, but on the condition that only IP addresses owned by the ISP may benefit from that server. Our approach is completely compatible with such arrangements because such a policy can be easily enforced via access network-level firewalls, which forces our system to avoid such CDN replicas. Second, on shorter timescales, CDNs may deploy load-balancing policies that distribute the traffic load such that clients are not always directed to the closest replica server. Our system respects such load-balancing policies because it always selects the first CDN replica from a recommended set, *i.e.*, does not opportunistically select the "best" replica from the set.

Clients have clear incentives to adopt our approach since it directly improves their performance. While it is certainly the case that CDNs share the same incentives for our system's adoption, this is even more true with the proliferation of *CDN brokers*, *e.g.*, [8]. CDN brokers actively measure performance to multiple CDNs and they can redirect a client to a different CDN on the fly in case the QoE isn't satisfactory [40]. It has been demonstrated that this particularly hurts large CDNs, which often have better replicas in the vicinity of the clients, which the broker is unfortunately unaware of; this leads to the loss of clients and revenues for CDNs [40]. Thus, utilizing clients' help in selecting the best CDN replicas in their vicinity directly benefits CDNs.

We thus argue that an *unrestricted* adoption of the ECS option, which is the key mechanism that enables the subnet assimilation used by our system (Section 2.3), is in the best interest of *every* CDN. In particular, the unrestricted ECS option enables a client to use the ECS field to specify a subnet different from the client's to change the way a CDN maps the client to its replicas. While most CDNs do enable ECS in its unrestricted form, *e.g.*, Google and EdgeCast among others, Akamai does so only via OpenDNS [16] and GoogleDNS [13] public DNS services, using the actual IP address of the requester. As such, Akamai's CDN is currently not directly usable by our system, as our system requires sending ECS requests on behalf of hop IPs. One possible reason for such a restriction imposed by Akamai might be the ability of third-parties to accurately reverse-engineer a CDN's scale and coverage without significant infrastructural resources, *e.g.*, [26, 46]. However, that even without unrestricted ECS, Akamai's network has been quite comprehensively analyzed in the past, *e.g.*, [36, 47, 48]. One of our main contributions lies in showing that the benefits achievable by letting clients help improve CDNs' decisions far outweigh the potential drawbacks of unrestricted ECS adoption.

2.3 Subnet Assimilation

Subnet assimilation is the deliberate use of the ECS field to specify a subnet *different* from the client's to change the way a CDN maps the client to its replicas. In this paper we show that the assimilation of subnets found along the path between the client and its "default" replica may allow the client to "reposition" itself in the CDN's mapping scheme, such that the client will consistently receive *lower-latency* replica recommendations from the CDN. Subnet assimilation is a key mechanism used both to *detect* and *utilize* lower-latency CDN replicas.

Latency valley. Figure 1 illustrates the key heuristic that helps clients realize when they are not optimally served by a CDN. Consider a client C, redirected to a CDN server S_1, as illustrated in the figure. If a hop on the path is not redirected

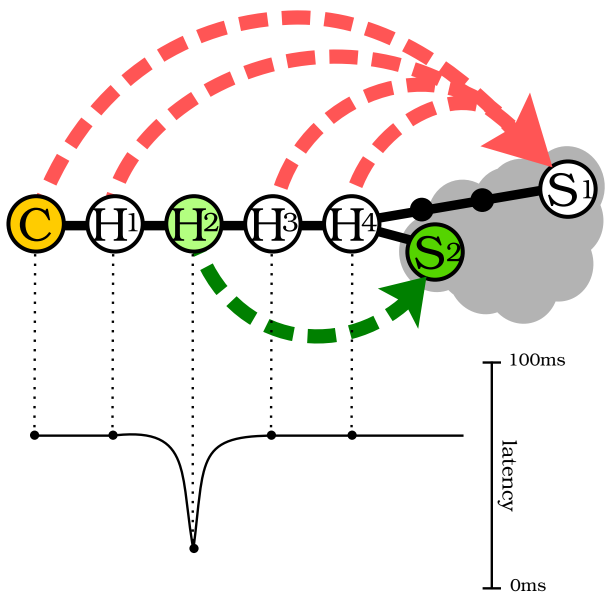


Figure 1: Illustration of a latency valley.

to the same server S_1, then it is possible that a lower-latency replica, *e.g.*, S_2, has been observed, such that the latency between C and S_2 is smaller than the latency between C and S_1. We refer to such a scenario as a *latency valley*.

For the remainder of this paper, we refer to the set of replicas suggested to the client's actual subnet as the *client replica set* (*CR*-set), and a replica from the *CR*-set as a client replica (*CR*). We refer to the routers along the presumed path from the client to any given *CR* as *hops*. Next, we refer to the set of replicas suggested to a hop, discoverable via subnet assimilation, as a *hop replica set* (*HR*-set). We denote a measurement from the client to a *CR* as a client replica measurement (*CRM*). Likewise, we denote a replica from the *HR*-set as a hop replica (*HR*) and a measurement to that replica from the client as a hop replica measurement (*HRM*). Finally, we define a latency valley to be any occurrence of the following inequality: $HRM/CRM < v_t \leq 1$. We take $v_t = 1$ until we optimize our parameter selection in Section 5.1.

It is imperative to note that our goal is not to find lower-latency replicas, perhaps included in an *HR*-set. Instead, we aim to find *subnets* to which lower-latency replicas are consistently suggested; in other words, subnets that are prone to valley-occurrences in replica sets obtained at any time.

2.4 Identifying Latency Valleys

In order to identify latency valleys, we first identify the hops along the path, which we achieve using traceroute. We are using the upstream path for simplicity; the downstream path, which may be assymetric, could yield different results, but would require more overhead or control of the CDN for execution. We then identify the corresponding HR-set for each hop, using ECS queries which assimilate the hops' subnets. Lastly, we must compare the *HRM*s to the *CRM*. Unless otherwise noted, we calculate *HRM*s and *CRM*s using ping RTTs (obtained by averaging the results of three back-to-back pings), and we refer to these values as *latencies*. Throughout this paper, latency units are milliseconds. For simplicity, we refer to the ratio HRM/CRM, used in our latency valley definition, as the *latency ratio*. A latency ratio below 1 indicates that the *HR* has outperformed the *CR*, a latency ratio above one indicates the *CR* has outperformed the *HR*, and finally, a latency ratio equal to 1 indicates that the *CR* and *HR* performed equally (which, in general, only happens when *CR* and *HR* refer to the same replica).

The above operation cannot be executed on-the-fly, as the time consumed by this process will outweigh any latency gains achieved. Moreover, multiple measurements might be required. Fortunately, our experiments in Section 3 show that a small number of measurements per hop — less than 10 — are required, and that the measurement results remain applicable on timescales of days. Because we can rely on measurements obtained in idle time, real-time measurements are not necessary for our system. Thus, our proposed technique will use past measurements to predictively choose a good subnet for future assimilation.

3 EXPLORING VALLEYS

In this section we establish that latency valleys are common phenomena in the wild, and that they offer substantial performance gains. In addition, we lay the groundwork for finding valleys, a prerequisite to harnessing their performance gains via subnet assimilation.

3.1 Testing for Valleys

We perform our preliminary tests using PlanetLab nodes, a platform which offers a large number of vantage points from around the globe, primarily deployed in academic institutions [31]. We further investigate latency valleys as experienced by a large variety of clients using the RIPE Atlas platform [17], as detailed in Section 5. While PlanetLab lacks the scale and variety offered by RIPE Atlas, the flexibility and freedom it provides made it a prime choice for our preliminary analysis. Our PlanetLab experiments use 95 nodes spread across 51 test locations, and we obtain the IP addresses of hops between nodes and CDN replicas via traceroute.

While latency valleys are a common occurrence, as we demonstrate below, many hops between the clients and the provider's servers can be discarded when searching for valleys. One category of such hops is of those which reside within the same local network as the client, which will be suggested the same replicas as the client. Another disposable category — hops with private IP addresses — will not be recognized by the EDNS server and will yield generic replica choices, regardless of the hops' locations. To ensure that a hop is indeed usable, a hop must (*i*) belong to a different /16 subnet than the client, (*ii*) have a different domain than the client, and (*iii*) belong to a different ASN than the client. We only filter hops that fail these conditions at the *beginning* of the route; once a hop is observed that meets the above three constraints, we stop filtering for the remainder of the route.

3.1.1 Provider Selection. While ECS is gaining momentum among CDNs, as it allows them to better estimate their clients' location, it is important to note that ECS was only recently deployed, and its implementation varies among the different providers. "A Faster Internet," an initiative headed by Google and OpenDNS, is drawing together a growing number of large CDNs and content providers to adopt and promote the adoption of ECS [42]. In order to attain a set of CDNs for our tests, we have selected those providers which implement an unrestricted form of ECS, as described in Section 2.2.

In order to best select the CDNs for our tests we scraped URLs from over 3000 sites arbitrarily selected from the Alexa Top 1M list [2]. Then, we reduced our set of URLs to those that ended in a known file type, in order to ensure our ability to perform download tests on the selected URLs. When applicable, we resolved CNAME domains to their respective CDN domains. We then performed multiple DNS queries for each URL to determine if unrestricted ECS is implemented. From the remaining URLs we have extracted the 6 CDN domains which have appeared most frequently, along with their respective URLs.

Before we can begin seeking subnets with *better* mapping-groups, it is first necessary to prove that subnets with *different* mapping-groups exist and are easy to find. We define *usable route length* as the number of hops along the path that fulfill the above filtering. Next, we define *divergence* as the fraction of usable hops along the path which were recommended at least one replica that was not recommended to the client.

Figure 2 depicts usable route length and divergence for different CDNs. The figure shows that, for example, when requesting replicas from Google, each client had an average usable route length of 7.8 and the divergence is approximately 92%. It is evident from Figure 2 that hops are indeed suggested different replicas than their client. As we encounter a wider variety of recommendations, we increase our number of opportunities to find latency valleys. The high divergence shown in Figure 2 indicates that other, and possibly lower-latency, replicas can be mapped to clients if they were to assimilate their hops' IP addresses.

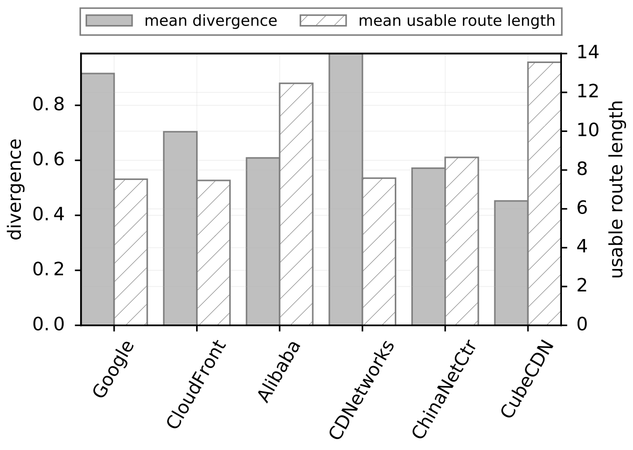


Figure 2: Average divergence and average usable route length per CDN

For the remainder of this paper, we use the following set of providers: Google, Amazon CloudFront, Alibaba, ChinaNet-Center, CDNetworks, and CubeCDN, a set which is both diverse and comprehensive. Google's CDN infrastructure is massive and dispersed: as of 2013, around 30000 CDN IP addresses spread across over 700 ASes were observable in one day [25]. As of this writing, Amazon CloudFront offered over 50 points-of-presence, spread throughout the world [6]. CDNetworks offers over 200 points-of-presence, also globally dispersed, and heavily employing anycast for server selection. Alibaba and ChinaNetCenter offer over 500 CDN node locations, each, within China where they are centered, in addition to a growing number of service locations outside of China, [3, 30]. Finally, CubeCDN is a smaller CDN with locations spread primarily across Turkey [14].

3.1.2 Test Execution. In order to assert the existence of latency valleys in the wild, we have executed a series of trials using the aforementioned URLs, and the PlanetLab nodes as clients. Each trial consists of the following steps:

(1) We randomly select a URL
(2) Client retrieves *CR*-set for the selected URL's domain
(3) For each *CR*, client uses traceroute to identify hops
(4) Using subnet assimilation, client retrieves the *HR*-set for each hop
(5) Client measures *CRM*s for every *CR* in its *CR*-set and *HRM*s for every *HR* it has seen (across all *HR*-sets obtained in that trial)

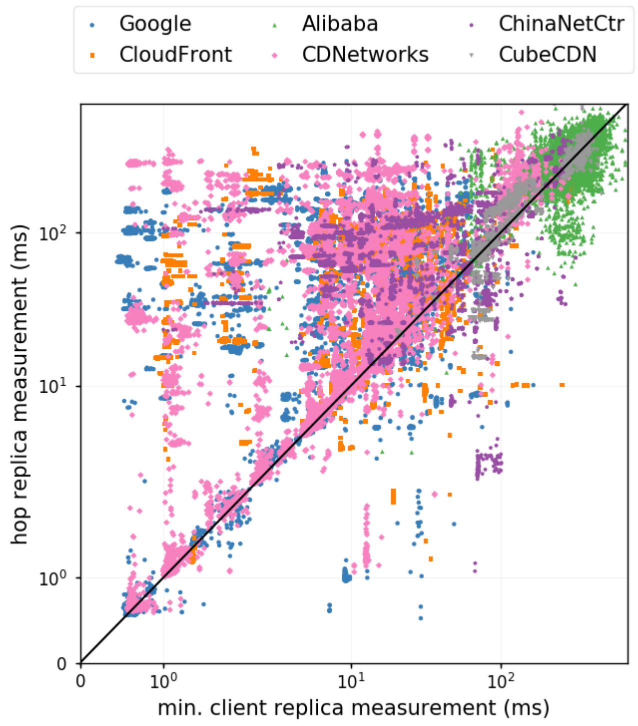


Figure 3: Scatter plot of HRMs and minimum CRMs. The area below the diagonal is the *valley region.*

We have performed a series of 45 trials per PlanetLab node, executed between one and two hours apart. For the remainder of this Section, we refer to this dataset.

3.2 Valley Prevalence

In order to establish the prevalence and significance of latency valleys, we now compare *HRM*s to their respective *CRM*s. It is important to note that it is possible that a client has multiple *CR*s, and hence, multiple *CRM*s, and in practice a client can only choose one replica. Therefore, we compare *all HRMs* to the *minimum CRM* obtained in their trial, thus establishing a *lower bound* for our expected performance gain, and allowing us to easily assert any gains or losses provided by the alternative replica choices (the *HR*s).

With our PlanetLab dataset, we compared each *HRM* to the minimum *CRM*, *i.e*, the best client replica, obtained in the same respective trial. Figure 3 shows the results of this assessment. We plot *HRM* on the y axis and minimum *CRM* on the x axis, thus creating a visual representation of the latency ratio. We distinguish between the CDNs using different colors. To better explain the results we draw the equality line where *HRM* = minimum *CRM*. Every data point along this line represents a hop's subnet which is being suggested a replica that performs on a par with the replicas suggested to the client's subnet. More importantly, every data point below the equality line represents a valley occurrence, as the alternative replica's *HRM* is smaller than the *CRM*. The percentage of valleys, by provider, ranges from 14.02% (CloudFront) to 38.58% (CubeCDN), with an average of 22% across all providers. Table 1 shows the results, *i.e.*, Column 2 (% Valleys Overall) lists these percentages for each CDN.

Now that we have established that valleys exist, we wish to determine where we can expect to find valleys. We continue using the minimum *CRM* from each trial, for reasons described above. However, in order to better reflect practical scenarios where only one of a given set of replicas is used, we now also begin choosing an individual replica from an *HR*-set. Instead of choosing the minimum *HRM* for a given hop, we conservatively choose the *median.* Each hop's chosen *HRM* is the median of its *HR*-set for that trial. We continue this pattern (choosing the client's best *CRM* from the trial and a hop's median *HRM*) for the remainder of our PlanetLab data analysis. Our PlanetLab data thus represents a *lower bound* on valleys and their performance implications; we compare the *median* performance of our proposed procedure to the absolute *best* the existing methods have to offer. In Section 5, using our RIPE Atlas testbed, we remove this constraint to demonstrate the real-world performance of our system.

We further wish to consider how common valleys are within a given route from the client to the provider. Column 3 (Average % of Valleys per Route) of Table 1 shows, given some route in one trial, the average percent of usable hops that incur valleys. We see for Alibaba that, on average, over a third of the usable hops in a given route are likely to incur valleys, and for CDNetworks, nearly a fourth of the route. Meanwhile, for CloudFront, we see that on average, valleys occur for only a small portion of hops in a given route. Column 4 (% Routes with a Valley) details the percentage of all observed routes that contained at least one valley in the trial in which they were observed. For Alibaba and CDNetworks, around 75% of the observed routes contain valleys, while more than 50% for Google.

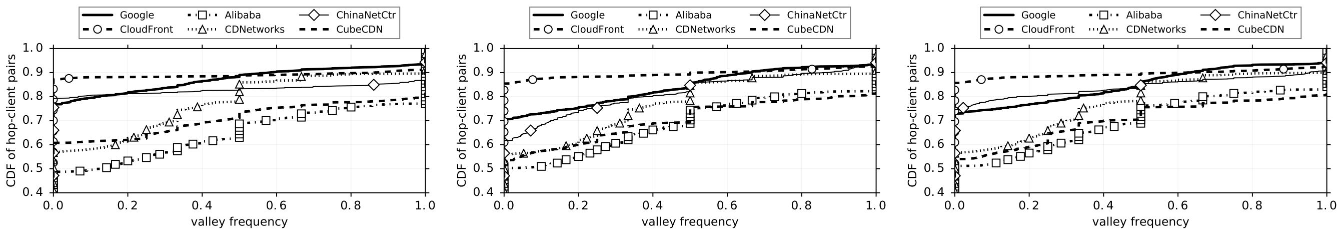
3.2.1 Do Valleys Persist? Next we assess whether subnets are persistently valley-prone. For some number of trials, how often can the client *expect* a valley to occur from some particular hop? To answer this, we need to be able to describe how frequently valleys *occurred* for some hop subnet in a given set of trials.

Valley frequency We define a new simple metric, the valley *frequency*, as follows: if we look at a hop-client pair across a set of N trials, and v trials are valleys, the valley frequency, v_f, is

$$v_f = \frac{v}{N}.$$

Table 1: Detailed findings for each provider, based on PlanetLab data

Provider	% Valley Overall	Avg %Valleys per Route	% Routes with Valley	% Hop-Client Pairs w/ Valley Freq. >0.5
Google	20.24%	16.41%	53.30%	10.98%
Amazon CloudFront	14.02%	8.72%	25.82%	10.00%
Alibaba	33.68%	35.94%	75.83%	30.97%
CDNetworks	15.61%	24.41%	73.08%	14.09%
ChinaNetCenter	27.42%	14.26%	38.10%	16.74%
CubeCDN	38.58%	17.95%	25.49%	26.32%



(a) ping **(b) total download time** **(c) total post-caching download time**

Figure 4: CDFs of clients by valley frequency. In 4a, the subnet-response measurements are ping times averaged from bursts of 3 back to back pings. In 4b, the subnet-response measurements are total download times on first attempts, while in 4c the subnet-response measurements are total download times on consecutive attempts (repeated downloads that take place immediately after first attempt to account for the potential impact of caching). Downloads were performed by using curl, where we set an IP as a destination and set the domain as the HOST attribute. Measurements for 4b and 4c were performed back-to-back so that 4c reflects download times with presumably primed caches.

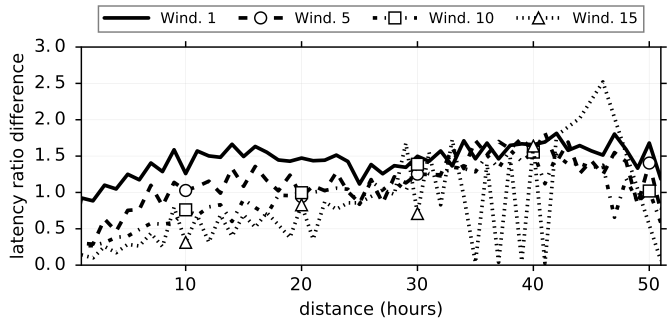
A frequency of 0.5 would imply that in half the trials performed, a valley was found in said hop. For each provider, we plot the valley frequency for each hop-client pair across our PlanetLab dataset.

Figure 4 shows the results. Note that Figures 4b and 4c use the total download time and the post-caching total download time, respectively, for the subnet measurements as opposed to pings.[2] *We further note that their results closely follow those obtained using pings, as in Figure 4a.* For simplicity, and due to download limitations of our RIPE testbed, we revert to only using pings for the remainder of this paper.

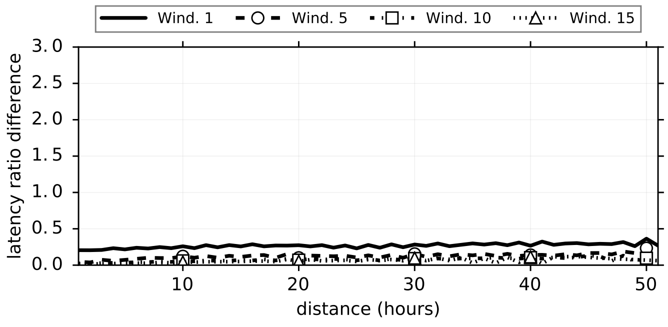
Figure 4 shows that approximately 5%-20% of hop-client pairs resulted in valleys 100% of the time (see y-axis in the figures for x=1.0) for every trial across the entire three day test period. Such hops will be easy to find given a large enough dataset. However, of more interest to us are hop-client pairs that inconsistently incur valleys. For example, perhaps if valleys can be found 50% of the time for some hop-client pair — *i.e.*, a valley frequency of 0.5 — that hop may be a sufficiently persistent producer of valleys for us to reliably expect good replica choices. This would provide our system with more opportunities to employ subnet assimilation and, ideally, improve performance. Column 5 (% Hop-Client Pairs with Valley Frequency > 0.5) of Table 1 shows the percentage of hop-client pairs that have valley frequencies (calculated across all 45 trials), greater than 0.5, *i.e.*, hop-client pairs that incurred valleys in the majority of their trials.

3.2.2 Can We Predict Valleys? While valley frequency can tell us how often valleys occurred on a *past* dataset, the metric makes no strong implications about what we can expect to see for a *future* dataset. To answer this, let us consider how the latency ratio changes as we increase the time passed between the trials we compare. In addition, per our reasoning in the v_f metric, we may want to compare a set of consecutive trials — a *window* — to another consecutive set of the same size, rather than only comparing individual trials (which is essentially comparing windows of size 1). Given a set of trials, we take a window of size N and compare it to all other windows of size N, including those that overlap, by taking the difference in their latency ratios (HRM/CRM). If we plot this against the time distance between the windows, we can observe the trend in latency ratios as the distance in

[2]We fetched .png and .js files, 1kB – 1MB large, hosted at the CDNs.



(a) All client-hop pairs



(b) Valley at least once

Figure 5: In both figures, we compare the change in latency ratio between two trial windows to the distance in time between the two windows. In figure a, we use all hop client pairs. In figure b, we restrict our set to pairs that experience at least one valley in at least one of their 45 trials.

time between windows increases. An upward slope would imply that the latency ratio is drifting farther apart as the time gap increases, while a horizontal line would imply that the change in latency ratio is not varying with time (*i.e.*, windows an hour apart are as similar to each other as windows days apart). A jagged or wildly varying line would indicate that future behavior is unpredictable.

Figure 5 plots the latency ratio versus distance in time for several window sizes, ranging from 1 to 15. In these plots, we use the windows' median latency ratio values for comparison. In particular, in Figure 5a, we use our entire dataset, *i.e.*, consider all hop-client pairs, *independently* of if they ever incur latency valleys or not, and plot latency ratio differences over time. For example, assume that for a client-hop pair $HRM = 80$ ms at one measurement point, while it rises to 120 ms at another measurement point. Meanwhile, assume that CRM remains 100 ms at both measurement points. Thus, the latency ratio HRM/CRM changes from 80/100 to 120/100, *i.e.*, from 0.8 to 1.2. The latency ratio difference is 0.4. Figure 5a shows that the latency ratio difference values both increase and vary wildly as windows become more distant. This shows that hop-subnet performance, overall, is likely extremely unpredictable. We hypothesize this results from unmapped subnets receiving generic answers, as observed in [47]. However, subnet assimilation requires that we have some idea of how a subnet will perform in advance.

Since valley-prone subnets are of particular interest to us, in Figure 5b, we reduce our dataset to only include client-hop pairs that have at least *one* valley occurrence across all 45 of its trials combined. The plot's behavior becomes dramatically more stable after applying this simple, minor constraint. The effects of the window size also become apparent in Figure 5b. Even with a window of size 1, the slope is very small and the line is nearly flat; after over 50 hours, two windows are only 10% to 20% more dissimilar than two windows a single hour apart. For window sizes greater than 5, latency ratios are often within 5% of each other, regardless of their distance in time. Going from a window size of 1 to 5 shows drastic improvement, while each following increase in window size shows a smaller impact. The results clearly show that a client can effectively identify valley-prone subnets and predict valleys with a sufficiently long window size.

3.2.3 Valley Utility Analysis. Figure 6 shows the distribution of the *lower bound* latency ratios seen by the set of all valley occurrences for each provider. The latency ratio represents a lower bound because we use the minimum latency measure for replicas recommended to the client. For example, if the minimum CRM is 100 ms and HRM is 80 ms, then the latency ratio is 0.8. Thus, the closer to 0 the latency ratios are, the larger the gain from subnet assimilation. The red line shows median, the box bounds the 25th and 75th percentiles, and the whiskers extend up to data points 1.5 times the interquartile-range (75th percentile - 25th percentile) above the 75th percentile and below the 25th percentile. Data points beyond the whiskers, if they exist, are considered outliers and not shown.

Figure 6 shows that most of the providers show opportunities for significant latency reduction. From this plot, it appears that Amazon CloudFront and ChinaNetCenter offer the greatest potential for gains. With the exception of CDNetworks, we see all of our providers have 25th percentiles near or below a latency ratio of 0.8, a 20% performance gain. We also observe the diversity of valley "depth". For example, we see in Figure 6 that ChinaNetCenter's interquartile range spans over 40% of the possible value space (between 0 and 1). With such a wide variety of gains for 50% of its valleys, it opens the door to being more selective. Rather than simply chasing *all* valleys, we could set strict requirements,

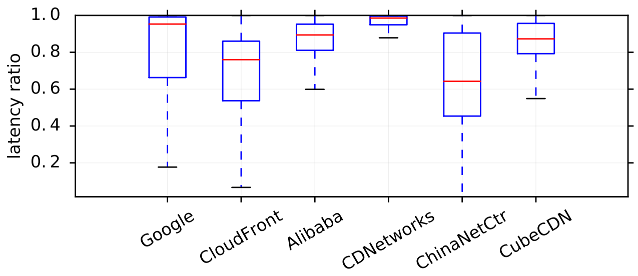


Figure 6: Lower bound of latency ratio of all valley occurrences.

tightening our definition of what's "good enough" for subnet assimilation, as we evaluate in Section 5.1.

CDNetworks' performance is more tightly bounded, as its interquartile range covers less than 10% of the value space and sits very near to 1. This implies CDNetworks probably offers very little opportunity for our technique to improve its replica selection. We hypothesize that this is a byproduct of CDNetworks anycast implementation; for replica selection, anycast depends more on routing and network properties than DNS and IP selection [27]. In addition, Google's median and 75th percentile latency ratios sit very near 1.0. However, by being more selective as described above, we may be able to pursue better valleys in the lower quartiles, below the median. We could potentially improve our system's valley-prone hop selection by filtering out "shallow" valleys from our consideration. We demonstrate and discuss the effects of different levels of selectiveness in Section 5.1.

4 DRONGO SYSTEM OVERVIEW

We introduce Drongo, a client-side system that employs subnet assimilation to speed up CDNs. Drongo sits on top of a client's DNS system. In a full deployment, Drongo will be installed as a LDNS proxy on the client machine, where it would have easy access to the ECS option and DNS responses it needs to perform trials. Drongo is set by the client as its default local DNS resolver, and acts as a middle party, reshaping outgoing DNS messages via subnet assimilation and storing data from incoming DNS responses. In our current implementation, Drongo uses Google's public DNS service at 8.8.8.8 to complete DNS queries.

Upon reception of an outgoing query from the client, Drongo must decide whether to use the client's own subnet or to perform subnet assimilation with some known, alternative subnet. If Drongo has sufficient data for that subnet in combination with that domain, it makes a decision whether or not to use that subnet for the name resolution. If Drongo lacks sufficient data, it issues an ordinary query (using the client's own subnet for ECS).

4.1 Window Size

We now face the question: What is a sufficient amount of data needed by Drongo to ensure quality subnet assimilation decisions? We choose to measure the data "quantity" by the number of trials for some subnet, where the subnet is obtained via traceroutes performed during times when the client's network was idle. A sufficient quantity must be enough trials to fill some predetermined window size. As we observed in Section 3.2.1, the marginal benefit of increasing the window size decreases with each additional trial. To keep storage needs low, while also obtaining most of the benefit of a larger window, we set Drongo's window size at 5.

4.2 Data Collection

Drongo must execute trials, defined in Section 3.1.2, in order to fill its window and collect sufficient data. As demonstrated in Figure 5b, *when* these trials occur is of little to no significance. In our experiments, we perform our trials at randomly sampled intervals; our trial spacing varies from minutes to days, with a tendency toward being near an hour apart. This sporadic spacing parallels the variety of timings we expect to happen on a real client: the client may be online at random, unpredictable times, for unpredictable lengths of time.

4.3 Decision Making

Here we detail Drongo's logic, assuming it has sufficient data (a full window) with which to make a decision about a particular subnet. For some domain, Drongo must decide whether a subnet is sufficiently valley-prone for subnet assimilation. If so, Drongo will use that subnet for the DNS query; if not, Drongo will use the client's own subnet. From Figure 5b, we know we can anticipate that future behavior will resemble what Drongo sees in its current window if Drongo has seen at least one valley occurrence for the domain of interest from the subnet under consideration. However, in Figure 6, we see that many valleys offer negligible performance gains, which might not outweigh the performance risk imposed by subnet assimilation. To avoid these potentially high risk hops, Drongo may benefit from a more selective system that requires a high valley frequency in the training window to allow subnet assimilation. We explore the effects of changing the v_f parameter in Section 5.

It is possible that for a single domain, multiple hop subnets may qualify as sufficiently valley-prone for subnet assimilation. When this occurs, Drongo must attempt to choose the best performing from the set of the qualified hops. To do this, Drongo selects the hop subnet with the highest valley frequency in its training window; in the event of a tie, Drongo chooses randomly.

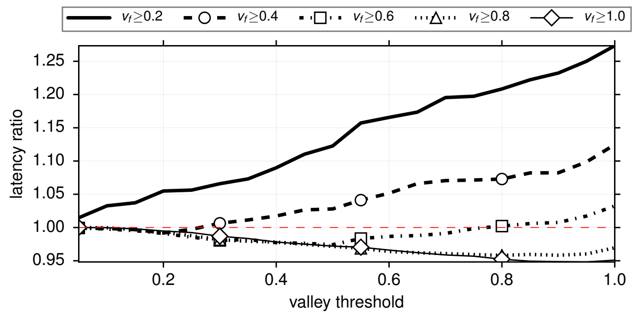


Figure 7: Average latency ratio of overall system as we vary v_f and v_t

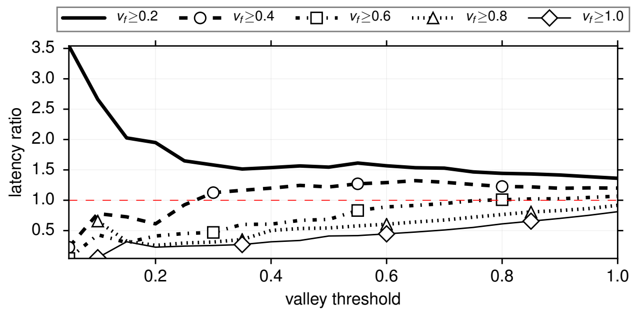


Figure 8: Average latency ratio of cases where subnet assimilation was performed

5 DRONGO EVALUATION

Here, we evaluate Drongo's ability to intelligently choose well-mapped hop subnets for subnet assimilation, and the performance gains imposed on clients where subnet assimilation is applied.

Using the RIPE Atlas platform [17], we collect a trace of 429 probes spread across 177 countries for 1 month. In our experiments, we use the first portion of a client's trials as a training window. Following the completion of the training window, we use the remaining trials to test Drongo's ability to select good hops for real time subnet assimilation. Each client performed 10 trials per provider: trials 0 through 4 to form its training window, and trials 5 through 9 to test the quality of Drongo's decisions. In our evaluation, Drongo *always selects the **first** CR from a CR-set and the **first** HR from a HR-set*, mirroring real world client behavior — no real-time on-the-fly measurements are conducted, and *all* decisions are based on the existing window.

5.1 Parameter Selection

Figure 7 shows the effects of Drongo on our entire RIPE trace's average latency ratio, *i.e.*, we consider *all requests* generated by clients, including those that aren't affected by Drongo. The figure plots average latency ratio (shown on the y axis) as a function of the valley threshold, v_t, shown on the x axis. The valley threshold, introduced in Section 2.3, determines maximum latency ratio a hop-client pair must have to classify as a valley occurrence. For example, when the threshold equals 1.0, all latency valleys are considered; on the other hand, when v_t is 0.6, Drongo triggers client assimilation only for latency valleys that have promise to improve performance by more than 40%, *i.e.*, latency ratio smaller than 0.6. The figure shows 5 curves, each representing a different valley frequency parameter, varied between 0.2 and 1.0.

Figure 7 shows the average results, across all tested clients. We make several insights. If the valley frequency is small, *e.g.*, 0.2, Drongo will unselectively trigger subnet assimilation for all valleys that have frequency equal to or larger than 0.2, which requires only one valley occurrence for our window size of 5. Conversely, as the minimum v_f parameter increases, overall system performance improves — the latency ratio drops below 1.0. Thus, the valley frequency is a strong indicator of valley-proneness, further supporting our prior findings from Figure 5b.

Meanwhile, two more characteristics stand out as we vary v_t. First, the valley frequency parameter can completely alter the slope of the latency ratio plotted against v_t. This behavior echoes the observation we made in the previous paragraph: with extremely loose requirements, Drongo does not sufficiently filter out poor-performing subnets from its set of candidates. Intersetingly, with a strict v_f (closer to 1.0), the slope changes, and the average latency ratio *decreases* as we raise the valley threshold. This is because if Drongo is too strict, it filters out too many potential candidates for subnet assimilation. Second, we see that the curve eventually bends upward with high v_t values, indicating that even the v_t can be too lenient. The results show that the minimum average latency ratio of 0.9482 (y-axis) is achieved for the valley threshold of 0.95 (x-axis). Thus, with $v_f == 1.0$ and $v_t == 0.95$, Drongo produces its maximum performance gains, averaging 5.18% (1.0 - 0.9482) across *all* of our tested clients. By balancing these parameters, Drongo filters out subnets that offer the least benefit: subnets where valleys seldom occur and subnets where valleys tend to be shallow.

Figure 8 plots the average latency ratio in the same manner as Figure 7, yet *only* for queries where Drongo applied subnet assimilation. First, the figure again confirms that lower valley frequency parameters degrade Drongo's performance. Second, paired with our knowledge of Figure 7, it becomes clear that as valley threshold decreases, the number of latency valleys shrinks while the *quality* of the remaining valleys becomes more potent. This is why the latency ratio decreases as v_t decreases. However, if the threshold is too

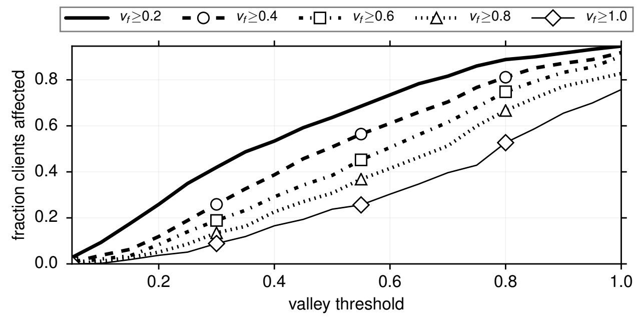


Figure 9: Percentage of clients where subnet assimilation was performed

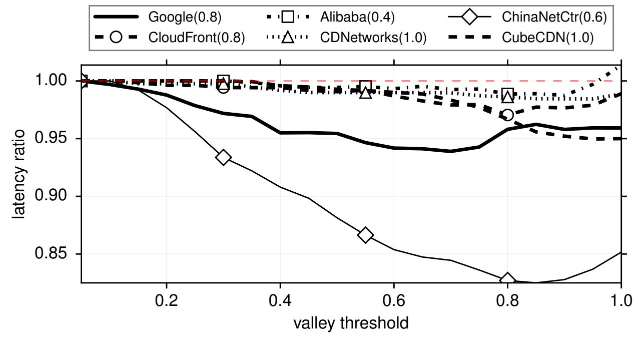


Figure 10: Per-provider system performance for all queries. Optimal v_f is set for each provider and noted in parentheses

small, the number of available valleys becomes so small that the performance becomes unpredictable, *i.e.*, outlier behavior can dominate, causing the spike in average latency ratios for valley thresholds under 0.2.

Finally, Figure 9 plots the percent of clients for which Drongo performed subnet assimilation for at least one provider. Necessarily, the less strict the frequency constraint is ($v_f \geq 0.2$ is the least strict constraint), the more frequently Drongo acts. As shown in Figure 9, 69.93% of our clients were affected by Drongo with v_f and v_t set at the the peak aggregate performance values (1.0 and 0.95, respectively) found above.

Summary: In this section, we selected parameters: $v_f = 1$ and $v_t = 0.95$, where Drongo reaches its peak aggregate performance gains of 5.18%.

5.2 Per-Provider Performance

In the previous subsection, we used a constant parameter set across all six providers in our analysis. In this section, we analyze Drongo on a per-provider basis. By choosing an optimal parameter for *each provider*, as we do below, Drongo's aggregate performance gains increases to 5.85%. In Figure 10,

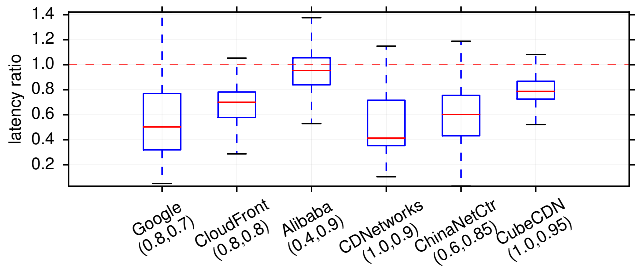


Figure 11: Per-provider system performance for queries where subnet assimilation was applied. Parentheses contain optimal values for each respective provider, formatted (v_f, v_t)

we show how Drongo impacts the average performance of each provider, while setting valley frequency to each individual provider's respective optimal value (noted in parentheses under each provider name). Note that, for most providers, the individual optimal valley frequency lies near the value obtained in Section 5.1 (1.0). While the intricacies of each provider's behavior clearly hinge on opaque characteristics of their respective networks, we offer several explanations for the observed performance. CDNetworks, which sees small gains across all valley threshold values, further validates the hypothesis proposed in Section 3.2.3 regarding anycast networks. Meanwhile, Google, the largest provider of our set on a global scale, also experienced the second highest peak average gains from Drongo. is not well served by Google, it is likely that there exist nearby subnets being directed to *different* replicas. In other words, Drongo has great opportunity for improving CDNs that use fine grained subnet mapping.

Finally, Figure 11 shows Drongo's performance, per-provider, exclusively in scenarios when it applies subnet assimilation. Comparing to the results shown in Figure 6 for the Planet Lab experiments, we observe significant differences. Most notably, the latency valley ratios are much smaller in Figure 11 than in Figure 6. For example, the Google's median latency ratio is close to 1.0 in Figure 6, indicating insigificant gains. On the contrary, Figure 11 Google's median latency ratio is around 0.5, implying gains of 50% (1.0 − 0.5) and up to an order of magnitude in edge cases. Considering all providers, Drongo-influenced replica selections are, on average, 24.89% better performing than Drongo-free selections.

There are three reasons for this behavior. First, Figure 6 shows the *lower-bound* system performance for PlanetLab, as the *best* CR is always used for comparison. Such a restriction is not imposed in the the RIPE Atlas scenario. Second, the RIPE set is more diverse and includes widely distributed endpoints, thus allowing CDNs greater opportunity for subnet mapping error. Third, contrary to the Planet Lab scenario where we used all the valleys under the valley threshold of 1.0, here we use optimal v_t values, such that Drongo successfully filters out most "shallow" valleys that would dilute performance gains.

6 RELATED WORK

Given a large number of CDNs, with vastly different deployment and performance in different territories, *CDN brokering* has emerged as a way to improve user-perceived CDNs' performance, *e.g.*, [7, 34]. By monitoring the performance of multiple CDNs, brokers are capable of determining which particular CDN works better for a particular client at a point in time. Necessarily, this approach requires content providers to contract with multiple CDNs to host and distribute their content. Contrary to CDN brokering, Drongo manages to improve performance of each individual CDN. Still, it works completely independently from CDNs and it is readily deployable on the clients. Moreover, Drongo is completely compatible with CDN brokering since it in principle has the capacity to improve the performance of whichever CDN is selected by a broker.

There has been tremendous effort expended in the ongoing battle to make web pages load faster, improve streaming performance, *etc.* As a result, many advanced client-based protocols and systems have emerged from both industry (*e.g.*, QUIC, SPDY, and HTTP/2) and academia [24, 41, 50] in the Web domain, and likewise in streaming, *e.g.*, [37]. While Drongo is also a client-based system, it is *generic* and helps speed-up all CDN-hosted content. By reducing the CDN latency experienced by end users, it systematically improves all the above systems and protocols, which in turn helps all associated applications.

Recent work has demonstrated that injecting fake information can help protocols achieve better performance. One example is routing [49], where fake nodes and links are introduced into an underlying linkstate routing protocol, so that routers compute their own forwarding tables based on the augmented topology. Similar ideas are shown useful in the context of traffic engineering [29], where robust and efficient network utilization is accomplished via carefully disseminated bogus information ("lies"). While similar in spirit to these approaches, Drongo operates in a completely different domain, aiming to address the CDN pitfalls. In addition to using "fake" (source subnet) information in order to improve CDN decisions, Drongo also invests efforts in discovering valley-prone subnets and determining conditions when using them is beneficial.

7 DISCUSSION

A mass deployment of Drongo is non-trivial and carries with it some complexities that deserve careful consideration. There is some concern that maintainence of CDN allocation policies may still be compromised, despite our efforts to respect their mechanisms in Drongo's design. We propose that a broadly deployed form of Drongo could be carefully tuned to effect only the most extreme cases. Figure 11 shows that subnet assimilation carries some risk of a loss in performance, so it is in clients' best interests to apply it conservatively. First, we know from Figure 9 that the number of clients using assimilated subnets can be significantly throttled with strict parameteres. Further, in today's Internet, many clients are already free to choose arbitrary LDNS servers ([25, 36]). Many of these servers do not make use of the client subnet option at all, thus rendering the nature of their side-effects — potentially disrupting policies enforced only in DNS — equivalent to that of Drongo. It is difficult to say whether or not Drongo's ultimate impact on CDN policies would be any more sigificant than the presence of such clients.

Mass adoption also raises scalability concerns, particularly regarding the amount of measurement traffic being sent into the network. To keep the number of measurements small while ensuring their freshness, a distributed, peer-to-peer component, where clients in the same subnet share trial data, could be incorporated into Drongo's design. We leave this modification for future work.

8 CONCLUSIONS

In this paper, we proposed the first approach that enables clients to actively measure CDNs and effectively improve their replica selection decisions, without requiring any changes to the CDNs and while respecting CDNs' policies. We have introduced and explored latency valleys, scenarios where replicas suggested to upstream subnets outperform those provided to a client's own subnet. We have asserted that latency valleys are common phenomena, and we have found them across all the CDNs we have tested in 26-76% of routes. We showed that valley-prone upstream subnets are easily found from the client, are simple to identify with few and infrequent measurements, and once found, are persistent over timescales of days.

We have introduced Drongo, a client-side system that leverages the performance gains of latency valleys by identifying valley-prone subnets. Our measurements show that Drongo can improve requests' latency by up to an order of magnitude. Moreover, Drongo achieves this with exceptionally low overhead: a mere 5 measurements suffice for timescales of days. Using experimentally derived parameters, Drongo affects the replica selection of requests made by 69.93% of clients, improving their latency by 24.89% in the median case. Drongo's significant impact on these requests translates into an overall improvement in client-perceived aggregate CDN performance.

REFERENCES

[1] 2016. Akamai. (2016). http://www.akamai.com/.

[2] 2016. Alexa Top Sites. (2016). https://aws.amazon.com/alexa-top-sites/.

[3] 2016. Alibaba Cloud CDN. (2016). https://intl.aliyun.com/product/cdn.

[4] 2016. Amazon CloudFront Ű Content Delivery Network (CDN). (2016). https://aws.amazon.com/cloudfront/.

[5] 2016. Amazon Route 53. (2016). https://aws.amazon.com/route53/.

[6] 2016. AWS Global Infrastructure. (2016). https://aws.amazon.com/about-aws/global-infrastructure/.

[7] 2016. Cedexis. (2016). http://www.cedexis.com/.

[8] 2016. Conviva. (2016). http://www.conviva.com/.

[9] 2016. The Cost of Latency. (2016). http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/.

[10] 2016. GoDaddy: Premium DNS. (2016). https://www.godaddy.com/domains/dns-hosting.aspx.

[11] 2016. Google CDN Platform. (2016). https://cloud.google.com/cdn/docs/.

[12] 2016. Google Cloud Platform: Cloud DNS. (2016). https://cloud.google.com/dns/.

[13] 2016. Google Public DNS. (2016). https://developers.google.com/speed/public-dns/docs/using?hl=en.

[14] 2016. Netdirekt: Content Hosting - CDN. (2016). http://www.netdirekt.com.tr/cdn-large.html.

[15] 2016. Neustar DNS Services. (2016). https://www.neustar.biz/services/dns-services.

[16] 2016. OpenDNS. (2016). https://www.opendns.com/.

[17] 2016. RIPE Atlas - RIPE Network Coordination Centre. (2016). https://atlas.ripe.net/.

[18] 2016. Shopzilla: faster page load time = 12 percent revenue increase. (2016). http://www.strangeloopnetworks.com/resources/infographics/web-performance-andecommerce/shopzilla-faster-pages-12-revenue-increase/.

[19] 2016. SwedenŠs Greta wants to disrupt the multi-billion CDN market. (2016). https://techcrunch.com/2016/08/30/greta/.

[20] 2016. Verisign Managed DNS. (2016). http://www.verisign.com/en_US/security-services/dns-management/index.xhtml.

[21] 2016. Verizon Digital Media Services. (2016). https://www.verizondigitalmedia.com/.

[22] 2016. Verizon ROUTE: Fast, Reliable Enterprise-Class Services for Domain Name System (DNS). (2016). https://www.verizondigitalmedia.com/platform/route/.

[23] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. 2010. Comparing DNS Resolvers in the Wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC ’10)*. ACM, 15–21. DOI: https://doi.org/10.1145/1879141.1879144

[24] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V. Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing Web Content to Improve User Experience on Mobile Devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 439–453. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/butkiewicz

[25] Matt Calder, Xun Fan, Zi Hu, Ethan Katz-Bassett, John Heidemann, and Ramesh Govindan. 2013. Mapping the Expansion of Google’s Serving Infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference (IMC ’13)*. ACM, 313–326. DOI: https://doi.org/10.1145/2504730.2504754

[26] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference (IMC ’15)*. ACM, 531–537. DOI: https://doi.org/10.1145/2815675.2815717

[27] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. 2015. Analyzing the Performance of an Anycast CDN. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference (IMC ’15)*. ACM, 531–537. DOI: https://doi.org/10.1145/2815675.2815717

[28] F. Chen, R. Sitaraman, and M. Torres. 2015. End-User Mapping: Next Generation Request Routing for Content Delivery. In *Proceedings of ACM SIGCOMM ’15*. London, UK.

[29] Marco Chiesa, Gaboe Retvari, and Michael Schapira. 2016. Lying Your Way to Better Traffic Engineering. In *Proceedings of the 2016 ACM CoNEXT (CoNEXT ’16)*. ACM.

[30] ChinaNetCenter. 2016. ChinaNetCenter - Network. (2016). http://en.chinanetcenter.com/pages/technology/g2-network-map.php.

[31] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: An Overlay Testbed for Broad-coverage Services. *SIGCOMM Comput. Commun. Rev.* 33, 3 (July 2003), 3–12. DOI: https://doi.org/10.1145/956993.956995

[32] C. Contavalli, W. van der Gaast, D. Lawrence, and W. Kumari. 2016. *Client Subnet in DNS Queries*. RFC 7871. RFC Editor.

[33] Tobias Flach, Ethan Katz-Bassett, and Ramesh Govindan. 2012. Quantifying Violations of Destination-based Forwarding on the Internet. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference (IMC ’12)*. ACM, 265–272. DOI: https://doi.org/10.1145/2398776.2398804

[34] Aditya Ganjam, Faisal Siddiqui, Jibin Zhan, Xi Liu, Ion Stoica, Junchen Jiang, Vyas Sekar, and Hui Zhang. 2015. C3: Internet-Scale Control Plane for Video Quality Optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. USENIX Association, Oakland, CA, 131–144. https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/ganjam

[35] Cheng Huang, Ivan Batanov, and Jin Li. 2012. A Practical Solution to the client-LDNS Mismatch Problem. *SIGCOMM Comput. Commun. Rev.* 42, 2 (March 2012), 35–41. DOI: https://doi.org/10.1145/2185376.2185381

[36] Cheng Huang, Angela Wang, Jin Li, and Keith W. Ross. 2008. Measuring and Evaluating Large-scale CDNs Paper Withdrawn at Mirosoft’s Request. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC ’08)*. ACM, 15–29. http://dl.acm.org/citation.cfm?id=1452520.1455517

[37] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. 2014. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. In *Proceedings of the 2014 ACM Conference on SIGCOMM (SIGCOMM ’14)*. ACM, 187–198. DOI: https://doi.org/10.1145/2619239.2626296

[38] Rupa Krishnan, Harsha V. Madhyastha, Sushant Jain, Sridhar Srinivasan, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. 2009. Moving Beyond End-to-End Path Information to Optimize CDN Performance. In *Internet Measurement Conference (IMC)*. Chicago, IL, 190–201. http://research.google.com/archive/imc191/imc191.pdf

[39] Zhuoqing Morley Mao, Charles D. Cranor, Fred Douglis, Michael Rabinovich, Oliver Spatscheck, and Jia Wang. 2002. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *Proceedings of the General Track of the Annual Conference on USENIX Annual Technical Conference (ATEC ’02)*. USENIX Association, Berkeley, CA, USA, 229–242. http://dl.acm.org/citation.cfm?id=647057.759950

[40] Matthew K. Mukerjee, Ilker Nadi Bozkurt, Bruce Maggs, Srinivasan Seshan, and Hui Zhang. 2016. The Impact of Brokers on the Future of Content Delivery. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks (HotNets '16)*. ACM, 127–133. DOI: https://doi.org/10.1145/3005745.3005749

[41] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/netravali

[42] OpenDNS. 2016. A Faster Internet: The Global Internet Speedup. (2016). http://afasterinternet.com.

[43] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose. 1998. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '98)*. ACM, 303–314. DOI: https://doi.org/10.1145/285237.285291

[44] Vern Paxson. 1996. End-to-end Routing Behavior in the Internet. *SIGCOMM Comput. Commun. Rev.* 26, 4 (Aug. 1996), 25–38. DOI: https://doi.org/10.1145/248157.248160

[45] Ingmar Poese, Benjamin Frank, Bernhard Ager, Georgios Smaragdakis, and Anja Feldmann. 2010. Improving Content Delivery Using Provider-aided Distance Information. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (IMC '10)*. ACM, 22–34. DOI: https://doi.org/10.1145/1879141.1879145

[46] Florian Streibelt, Jan Böttger, Nikolaos Chatzis, Georgios Smaragdakis, and Anja Feldmann. 2013. Exploring EDNS-client-subnet Adopters in Your Free Time. In *Proceedings of IMC '13 (IMC '13)*.

[47] Ao-Jan Su, David R. Choffnes, Aleksandar Kuzmanovic, and Fabián E. Bustamante. 2006. Drafting Behind Akamai (Travelocity-based Detouring). *SIGCOMM Comput. Commun. Rev.* 36, 4 (Aug. 2006), 435–446. DOI: https://doi.org/10.1145/1151659.1159962

[48] Ao-Jan Su and Aleksandar Kuzmanovic. 2008. Thinning Akamai. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement (IMC '08)*. ACM, 29–42. DOI: https://doi.org/10.1145/1452520.1452525

[49] Stefano Vissicchio, Olivier Tilmans, Laurent Vanbever, and Jennifer Rexford. 2015. Central Control Over Distributed Routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, 43–56. DOI: https://doi.org/10.1145/2785956.2787497

[50] Xiao Sophia Wang, Arvind Krishnamurthy, and David Wetherall. 2016. Speeding up Web Page Loads with Shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. USENIX Association, Santa Clara, CA, 109–122. https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/wang