NORTHWESTERN UNIVERSITY

Understanding and Improving Content Distribution Through Expansive Network Measurements

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Computer Science

By

Marc Anthony Warrior

EVANSTON, ILLINOIS

August 2019

Abstract

Understanding and Improving Content Distribution Through Expansive Network Measurements

Marc Anthony Warrior

In response to exponentially increasing demand for digital media, today's Internet landscape has evolved into a multitude of diverse and interdependent distribution systems designed to move content as efficiently as possible. While many of these systems have *individually* been explored in depth by both academic and industrial communities, a cross-sectional investigation of the *relationships* between competing or coexisting content distribution systems and resources is generally absent from the current narrative. Further, when such expansive studies are given consideration, they are avoided due to the daunting challenges they present. Scope and vantage point concerns become non-trivial when designing experiments that span multiple network resources, and third-party systems may lack transparency for the curious researcher.

In this thesis, I assert that expansive network measurements such as these are not only feasible, but *essential* to our efforts to understand and improve modern content distribution systems. I demonstrate that anchoring cross-sectional measurements in client-side machines provides the real-world perspectives necessary for optimizing actual client experience. Rather

than examine the performance of a single resource-client pair, I instead obtain, for each client considered, *relative* measurements across the set of systems and resources visible to the client or its peers. Each additional considered Internet resource or system provides relative context that highlights otherwise unobservable outlier properties.

With this approach, I achieve the following: First, I discover and resolve sub-optimal resource-client mappings using only a lightweight, client-side implementation. Next, I quantify the extent to which clients are exposed to the same network resources as each other, and I further leverage these results to systematically identify opportunities to improve client performance. Finally, I enable scalable assessment of a crowdsourced ecosystem's content aggregation and distribution patterns.

Thesis Committee

Aleksandar Kuzmanovic, Northwestern University, Committee Chair

Fabián Bustamante, Northwestern University, Committee Member

Peter A. Dinda, Northwestern University, Committee Member

Theophilus A. Benson, Brown University, Committee Member

Acknowledgements

The past five years have been a whirlwind, and the list of people I have to thank for helping me through this process has grown much longer than I could have ever anticipated. With that said, I will try my hardest in this section to give honor where honor is due.

First and foremost, I would like to thank my advisor, Aleksandar Kuzmanovic, who welcomed me into his lab and provided me with more creative freedom than 22 year old me knew how to handle. Your guidance has helped shape me from a student into a real researcher, and I am truly grateful for that. Thank you for trusting me to succeed even when evidence suggested I would not.

I would also like to express my deepest appreciation for my Committee. Flashbacks to my qualifying exam questions from Peter Dinda have forever burned the importance of statistics into my mind. Fabián Bustamante, the very first systems faculty I met from Northwestern, has happily put up with my frequent requests for Planet Lab slices. In all seriousness, I recognize that in academia, time is particulary precious, and I am thankful that professors in such high demand have made time for me. I owe special thanks to Theophilus Benson, who willingly joined my Committee from across the country before having met me.

I would be remiss to overlook the folks whom I have toiled alongside, through direct collaboration or in spirit. Uri Klarman, who is effectively my entire cohort by himself, lives a life that reminds me to keep dreaming big. To my collaborators, Romain Fontugne and Matteo Varvello, I owe enourmous thanks for their brilliant contributions and insights towards my research. I will

never forget the times I shared with the phenomenal teams at IIJ Innovation Institute and VDMS (EdgeCast). Lastly, I am forever indebted to Marcel Flores and Zachary Bischof, who each, in their own way, took me under their wings and offered invaluable mentorship from day one.

Words cannot express my appreciation for my friends and family: my fellow Army Brat brother, Ameer Jalal, who has consistently been there for me since 6th grade; my Georgia Tech brothers — Andrew Bryant, Austin Keaton, Kaeland Chatman, and Lance Charles — who reinvigorate me with every phone call and gathering; Bruce Lindvall, who has been cheering me on since before I arrived at Northwestern; the Evanston Church of Christ family, who treated me as one of their own as soon as I arrived; Penny Warren, whose warm heart made The Graduate School office a welcoming place; Edison Chen, who has been ever patient with me as a roommate for the past several years; the Black Graduate Student Association, which has been my home away from home; the Keynotes a capella group and the "Anime Night" crew that got me through each week; the trips to Wednesday night Bible study with Ashley Dennis and Bright Gyamfi that always lifted my spirits; the countless other loved ones I can't fit in this document.

Finally, I owe a tremendous shoutout to my incredible parents (still the best in the world): Anthony Warrior, who has always reminded me to do my best, and Michelle Warrior, who swears she always knew I'd be a doctor. Thank you!

Contents

List of Tables

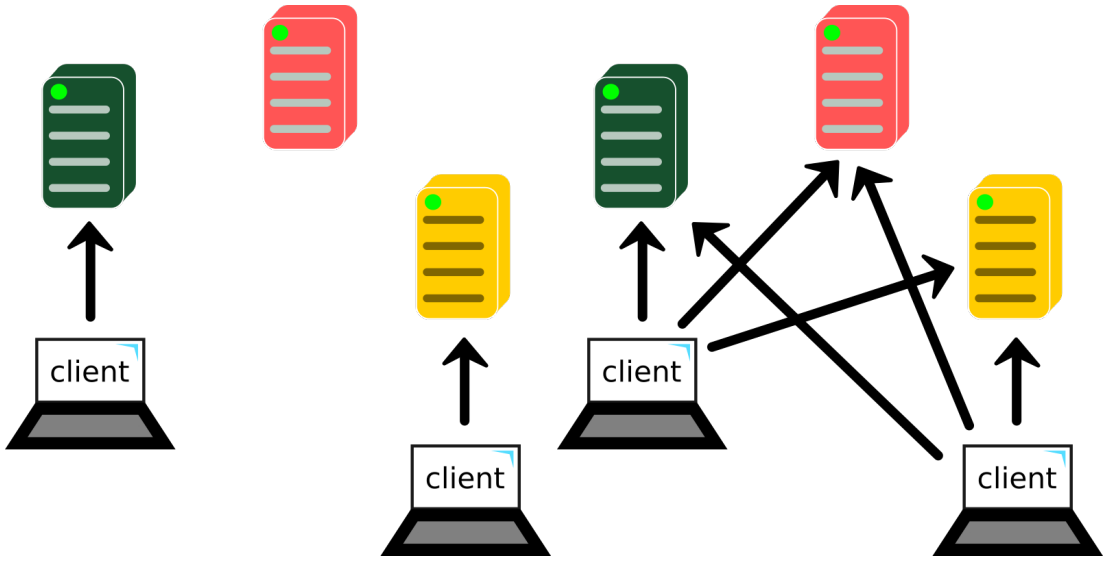
List of Figures

CHAPTER 1

Introduction

Digital content is the lifeblood of the modern Internet. The ease with which content flows from source to consumer can make or break even the most well-established online platforms [**17, 117, 22, 8**]. It is therefore no surprise that both academic and corporate communities alike have invested heavily in the advancement of content delivery techniques. The systems in place to support efficient and effective content distribution have multiplied and grown substantially over the past two decades [**1**], so much so that the very topology of the Internet has evolved to better address this specific, resource intensive concern [**49**]. Content distribution networks (CDNs), the primary component underlying many content delivery models, now host the vast majority of Internet content from most of the top domains used today [**87**].

In contrast, however, technology to *assess* modern content distribution systems has not kept pace with the systems themselves. The daunting task of measuring the performance of CDNs — often requiring a large number of diverse, edge network vantage points — has largely fallen to CDN brokers and so-called "meta-CDNs" which serve as abstractions to multiplex the wide array of CDNs on behalf of third-party, paying content providers [**39, 40**]. Such measurement platforms, comprised primarily of passive techniques silently embedded on their customers' sites and software, are inherently limited in scope and applicability and do very little to directly help with individual CDN optimization. Meanwhile, CDNs, whose private infrastructure may span the globe and employ thousands of machines, are faced with the increasingly difficult technical challenge of assessing and strategically improving themselves [**87, 47**]. On top of all of



(a) conventional approach (b) expansive approach

Figure 1.1. Diagrams of conventional (subfigure 1.1a) and expansive (subfigure 1.1b) approaches to content distribution system measurement.

this, the content itself, served by these ever-compounding systems of systems, has become difficult for content providers to trace and control. For example, thorough analyses of the patterns and implications of illegal distributed content distibution systems are rare, often opaque, and frequently misquoted [**81**].

I assert that the key, underlying fault in the aforementioned concerns is a lack of *breadth*. In this dissertation, I propose the widespread adoption of what I refer to as *expansive* network measurements: client-anchored, cross-sectional measurements designed to enable relative comparison between multiple content distribution resources. I support these claims by demonstrating the feasibility and effectiveness of expansive network measurements at three distinct levels of abstraction in real world content distribution systems.

1.1. Expansive Network Measurements

Expansive network measurements offer a measurement system design approach for breadth capturing, client-side measurements. Expansive network measurements span multiple content resources from a single client. For clarity, I here define "content resources" as any content distribution infrastructure providing entity — a unique network interface, a virtual machine, server, service, *etc.* — that aims to move digital content from some source or provider to content consumers (client machines). It is important that the low level details of resources are allowed to be opaque, as expansive measures span multiple such resources, each potentially stemming from different entities with varying levels of transparency. Figure 1.1 contrasts expansive measurements against the more commonly seen conventional measurement style.

While the concept of cross-sectional experiments may appear intuitive, several factors, which this dissertation aims to debunk, have driven modern industry stakeholders and researchers away from the expansive measurement model. Most notably, there is general consensus that client based measurements should avoid being load intensive, coupled with the assumption that wide-breadth measurements are necessarily high load. I, however, demonstrate that it is both reasonable and effective to design lightweight, expansive measurements that succeed in obtaining relevant, actionable insights that have gone unobserved in the conventional model. It is often tempting for researchers to hold a depth-focused measurement paradigm, in which research pursuits involve investigating one or two *specific* service providers (*e.g.*, a "large CDN"), either attempting to demystify part of an otherwise opaque proprietary system, or, through direct collaboration, working to address concerns specific to the provider of interest [**63, 76, 77, 54, 52, 104**]. As I will show in this document, opting to instead span *multiple* resources, regardless of their opaque nature, sheds light on properties that cannot be observed via a single provider. Lastly,

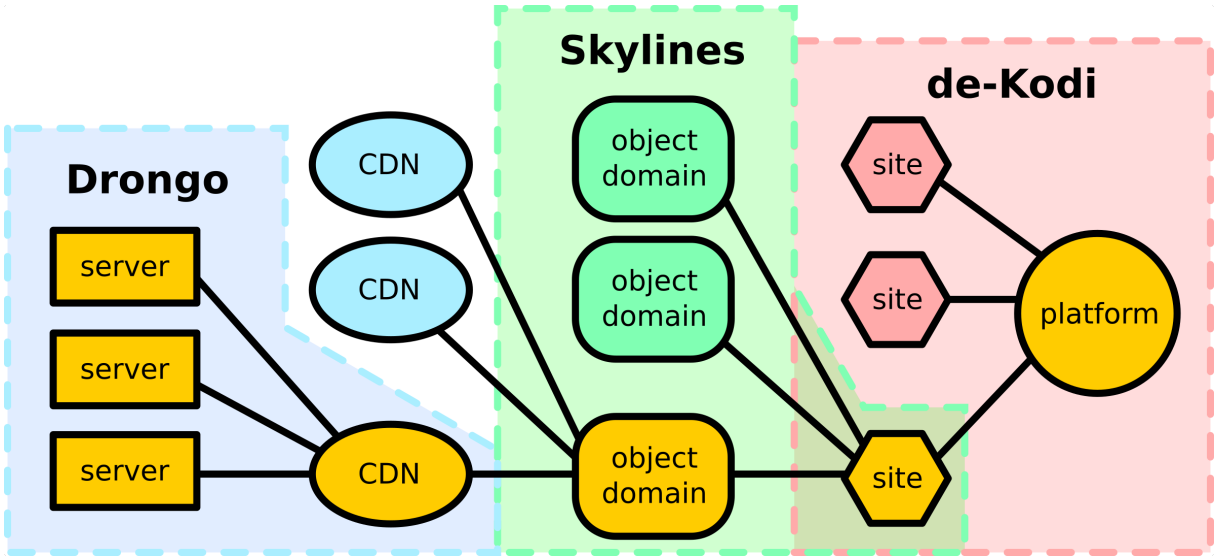


Figure 1.2. High level diagram of abstraction layers in content distribution systems. Layers pertaining to specific portions of this dissertation are highlighted and labeled accordingly.

there are varying concerns regarding the scalability of client-centric measurement paradigms, such as the marginal benefit of per-client adoption and the number of clients required for the system to work. I address these concerns on a per experiment basis, providing the reader with several examples of how to approach this problem in designing expansive network measurements.

1.2. Approach

A typical modern content distribution system can be described in terms of a series of abstractions, each layer multiplexing the one below it. A high level illustration of this is shown in Figure 1.2. Throughout this disseration, I target each abstraction layer with a demonstrational study of expansive network measurements.

My overall approach is as follows: First, I delve into the lowest layer of the provided model — content server selection. CDNs and similar services cache content across many network resources (labeled as servers in Figure 1.2), primarily to achieve low latency for geographically dispersed clients. In the corresponding project, titled Drongo, clients personally measure performance across multiple, strategically discovered replica servers and use the results to immediately enhance server-client mapping. Moving up the layers, I next present Skylines, where I perform the first ever aggregate analysis of DNS redirection catchments across multiple domains, identifying the extent to which clients are exposed to the same content resources. Finally, I perform an in depth case study of the topmost abstraction layer (far right in Figure 1.2), leveraging the scope of a platform whose ecosystem spans many websites and other content resources.

1.3. Mitigation of Suboptimal Client-Server Mapping

Content distribution networks depend heavily on efficient and accurate allocation of their content resources, which may number in the thousands and geographically span the globe. Although other factors, such as security, policy, and compute power, may come into play, low latency remains the chief concern of CDNs and their constituents [**17, 117, 22, 39, 8, 104**]. Ideally, clients should be directed to the replica server that will likely result in the lowest latency achievable between the CDN and that client. In practice, however, this has proven to nontrivial, and optimizating server selection remains an ongoing problem [**105, 107, 54, 102**].

Until now, the burden of assessing and improving an individual CDN's resource allocation scheme rested entirely on that CDN's shoulders. While CDNs have had some success in addressing this problem, I argue that their server-to-client mapping models inherently include

some "unknown unknowns": it is difficult for the CDN, or anyone else, to know whether a given client has been provided the optimal choice (for that client) from the set of servers at the CDN's disposal. In addition, the sheer size of the client address space under consideration — nearly 2^{32} for IPv4 alone — renders a *per client* optimization impractical for the CDN to attempt. Alternatively, as I demonstrate in this dissertation, it is possible for individual clients to participate and fine tune a CDN's mapping scheme. I introduce Drongo, a client-side tool by which clients can 1) identify when they have received a poor server choice, and, 2) in turn, improve resource allocation for *themselves*, on behalf of the CDN of interst. Drongo acheieves this by performing expansive measurements across a small subset of a given CDN's content resources.

1.4. Analysis of Common Network Resource Exposure

The vastness of today's Internet creates an intuitive but often overlooked phenomenon: not everyone is exposed to the same web resources. Even across the set of objects embedded in a single web page, a pair of clients with apparently similar network properties may be assigned to barely overlapping sets of network resources to pull from. While the properties of individual content distribution networks and the like are well explored, there has been, until now, a lack of insight regarding the *aggregate* behavior of these many large networks co-existing from the perspective of the countless domains built on top of them.

I perform a thorough analysis of cross-provider resource allocation patterns and the resulting aggregate mapping of over 9,000 RIPE Atlas clients around the world. To facilitate my research, I introduce common network resource exposure (CNRE) — a measure of the degree to which a pair of clients are exposed to the same content resources as each other across a large set of domains. I explore the implications of high and low CNRE scores, and use the results to identify

catchment anomalies: clients whose latency towards a given set of resources, across hundreds of web object domains, is substantially higher (by several standard deviations) than that of their peers (other clients directed primarily to the same resources). In addition, I identify aggregate catchment *centers*, which effectively approximate where the set of resources assigned to a given aggregate catchment are geographically concentrated.

1.5. Exploration of Open Source Media Platform Infrastructure

Free and open source media centers are currently experiencing a boom in popularity for the convenience and flexibility they offer users seeking to remotely consume digital content. This newfound fame is matched by increasing notoriety — for their potential to serve as hubs for illegal content — and a presumably ever-increasing network footprint [**106, 103**]. As shown in the rightmost section of Figure 1.2, such platforms serve as a gateway to a multitude of sites that act as content resources for consumers. These sites may directly demultiplex toward some physical resource as depicted in the figure, or they may themselves serve as additional intermediaries and content resource aggregators. I leverage the hub-like one such platform — Kodi — to perform a cross-provider analysis of a crowd-maintained content distribution system, reaching across many distinct providers and protocols. The Kodi ecosystem is home to tens of thousands of user-developed add-ons which act as site-like content resources.

To perform this analysis, I introduce de-Kodi, a client-side system with tunable scalability, capable of traversing large cross-sections of the Kodi ecosystem to arbitrary levels of depth. With de-Kodi, I uncover and assess content resources, their interdependencies, and the technology behind them. I further explore the life cycles and popularity trends of Kodi add-ons, and I investigate network traffic patterns for signs of illicit activity often attributed to the platform.

1.6. Thesis Organization

The remainder of this dissertation proceeds as described below. In Chapter 2, I produce my thesis statement, which serves as the foundation and chief motivation of this body of work. Following this, I present the aforementioned projects, each of which building apon the provided thesis and preceding sections. In Chapter 3, I demonstrate the effectiveness of expansive network measurements for empowering individual clients to improve their network performance. Next, Chapter 4 explores the aggregate behavior and implications of content hosting patterns across hundreds of domains. In Chapter 5, I uncover large cross-sections of disparate, user-maintained content resources and analyze their interrelationships and network characteristics. Finally, I offer a brief summary of my findings and contributions in Chapter 6, where I conclude this dissertation.

CHAPTER 2

Thesis

Expansive network measurements — measurements spanning multiple resources or providers from the vantage point of a single client — are essential to understanding and improving the allocation of media over the Internet.

CHAPTER 3

Drongo: Speeding Up CDNs with Subnet Assimilation from the Client

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 [**117**]. Amazon has reported that every 100 ms increase in page load time costs them 1 % in sales [**17**]. Client-server latency is essential for other killer apps such as web search [**8**] and video streaming [**77**]. Reduced latency directly improves throughput, which has been shown both theoretically [**99**] and empirically [**111**]. 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 [**2, 10**]. 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 [**111**]. 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 [**18, 83, 111**], (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 project, I demonstrate that client-server latency is far from optimal. I 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) My 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 the system.

I 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) [**62**] 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] I refer to such a scenario as a *latency valley*. *My goal is not to find lower-latency replicas – I 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, I ensure that any additional policies a CDN may have (such as network access restrictions and load balancing) are unaffected.

[1] It bears emphasizing that all of my measurements are performed between the *client* and a CDN *replica*; no measurements are performed directly from upstream nodes.

By conducting experiments on six major content pro- viders, I 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 I have tested: 26%–76% of routes towards CDN replicas discover at least one latency valley. My 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, I 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.

I 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. My measurements show that Drongo can improve requests' latency by up to an order of magnitude. Moreover, I 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.

I the following contributions:

- I 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.

- I extensively analyze client-side CDN measurements and determine critical time-scales and key parameters that empower clients to leverage their advanced views of CDNs.
- I introduce Drongo, a client-side CDN measurement crowd-sourcing system and demonstrate its ability to substantially improve CDNs' performance in the wild.

3.0.1. Premise

3.0.1.1. 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 [**2, 5, 10, 20**]. 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 [**41, 75, 86, 102**]. This issue was further dramatically amplified in recent years (see [**54**]) with the proliferation of public DNS resolvers, *e.g.*, [**6, 9, 11, 14, 19, 21**].

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) [**97**]. 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.

However, even when the actual client's network location is accurate, numerous factors prevent CDNs from providing optimal CDN replica selection [**83**]. 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.*, [**100**]. As a result, CDN selections could be heavily underperforming (see [**83**] 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 [**69**]. 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.

The 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 my 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.

3.0.1.2. 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. My approach is completely compatible with such arrangements because such a policy can be easily

enforced via access network-level firewalls, which forces my 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. My 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 my approach since it directly improves their performance. While it is certainly the case that CDNs share the same incentives for my system's adoption, this is even more true with the proliferation of *CDN brokers*, *e.g.*, [**40**]. 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 [**92**]. 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 [**92**]. 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 (Section 3.0.1.3), 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 [**15**] and GoogleDNS [**12**] public DNS services, using the actual IP address of the requester. As such, Akamai's CDN is currently not directly usable by my system, as my 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

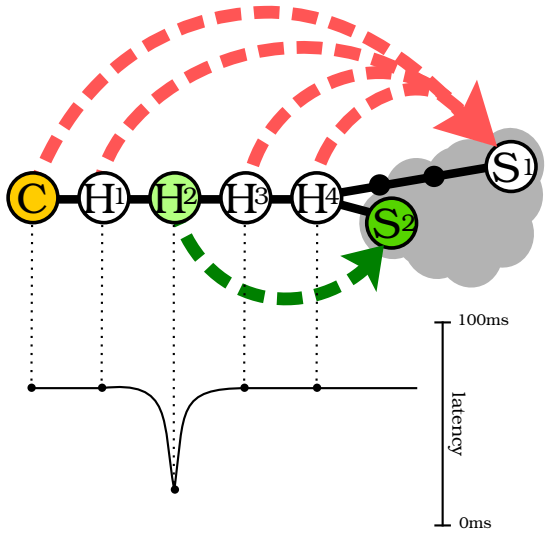


Figure 3.1. Illustration of a latency valley.

to accurately reverse-engineer a CDN's scale and coverage without significant infrastructural resources, *e.g.*, [**53, 110**]. However, that even without unrestricted ECS, Akamai's network has been quite comprehensively analyzed in the past, *e.g.*, [**76, 111, 112**]. One of my 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.

3.0.1.3. 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 thesis I 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 3.1 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 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. I refer to such a scenario as a *latency valley*.

For the remainder of this project, I 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). I refer to the routers along the presumed path from the client to any given CR as *hops*. Next, I refer to the set of replicas suggested to a hop, discoverable via subnet assimilation, as a *hop replica set* (HR-set). I denote a measurement from the client to a CR as a client replica measurement (CRM). Likewise, I 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, I define a latency valley to be any occurrence of the following inequality: $HRM/CRM < v_t \leq 1$. I take $v_t = 1$ until optimizing parameter selection in Section 3.0.4.1.

It is imperative to note that the goal of this project is not to find lower-latency replicas, perhaps included in an HR-set. Instead, I 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.

3.0.1.4. Identifying Latency Valleys. In order to identify latency valleys, I first identify the hops along the path, which I achieve using traceroute. I am 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. I then identify the corresponding

HR-set for each hop, using ECS queries which assimilate the hops' subnets. Lastly, I compare the HRMs to the CRM. Unless otherwise noted, I calculate HRMs and CRMs using ping RTTs (obtained by averaging the results of three back-to-back pings), and I refer to these values as *latencies*. Throughout this project, latency units are milliseconds. For simplicity, I refer to the ratio HRM/CRM, used in the 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, the experiments in Section 3.0.2 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 the system to function. Thus, my proposed technique will use past measurements to predictively choose a good subnet for future assimilation.

3.0.2. Exploring Valleys

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

3.0.2.1. Testing for Valleys. I perform preliminary tests using PlanetLab nodes, a platform which offers a large number of vantage points from around the globe, primarily deployed in

academic institutions [**57**]. I further investigate latency valleys as experienced by a large variety of clients using the RIPE Atlas platform [**16**], as detailed in Section 3.0.4. While PlanetLab lacks the scale and variety offered by RIPE Atlas, the flexibility and freedom it provides made it a prime choice for my preliminary analysis. My PlanetLab experiments use 95 nodes spread across 51 test locations, and I obtain the IP addresses of hops between nodes and CDN replicas via traceroute.

While latency valleys are a common occurrence, as demonstrated 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. I only filter hops that fail these conditions at the *beginning* of the route; once a hop is observed that meets the above three constraints, I stop filtering for the remainder of the route.

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 [**97**]. In order to attain a set of CDNs for my experiments, I have selected those providers which implement an unrestricted form of ECS, as described in Section 3.0.1.2.

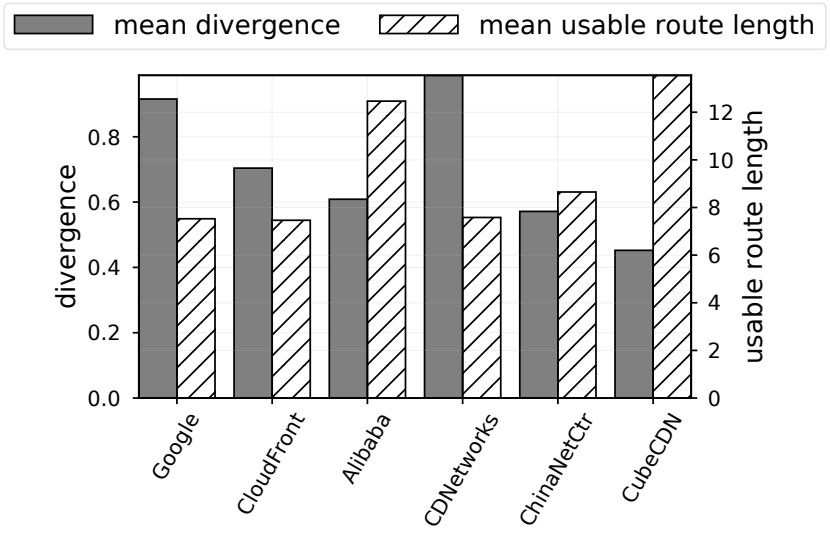


Figure 3.2. Average divergence and average usable route length per CDN

In order to best select the CDNs for my tests, I scraped URLs from over 3000 sites arbitrarily selected from the Alexa Top 1M list [**3**]. Then, I reduced the set of URLs to those that ended in a known file type, in order to ensure my ability to perform download tests on the selected URLs. When applicable, I resolved CNAME domains to their respective CDN domains. I then performed multiple DNS queries for each URL to determine if unrestricted ECS is implemented. From the remaining URLs I have extracted the 6 CDN domains which have appeared most frequently, along with their respective URLs.

Before beginning to seek subnets with *better* mapping-groups, it is first necessary to prove that subnets with *different* mapping-groups exist and are easy to find. I define *usable route length* as the number of hops along the path that fulfill the above filtering. Next, I 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 3.2 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 3.2 that hops are indeed suggested different replicas than their client. As we encounter a wider variety of recommendations, we increase the number of opportunities to find latency valleys. The high divergence shown in Figure 3.2 indicates that other, and possibly lower-latency, replicas can be mapped to clients if they were to assimilate their hops' IP addresses.

For the remainder of this project, I use the following set of providers: Google, Amazon CloudFront, Alibaba, ChinaNetCenter, 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 [**52**]. As of this writing, Amazon CloudFront offered over 50 points-of-presence, spread throughout the world [**7**]. 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, [**4, 56**]. Finally, CubeCDN is a smaller CDN with locations spread primarily across Turkey [**13**].

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

(1) 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

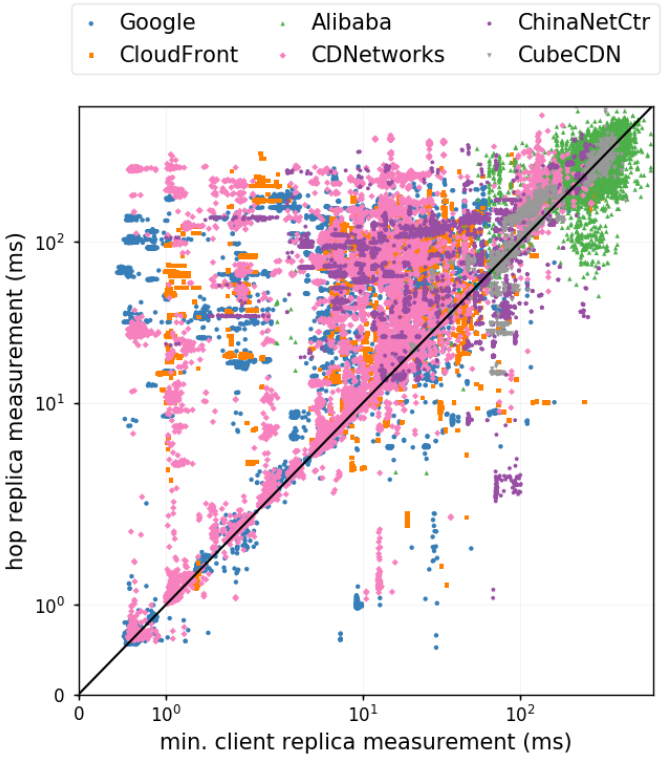


Figure 3.3. Scatter plot of HRMs and minimum CRMs. The area below the diagonal is the *valley region*.

(4) Using subnet assimilation, client retrieves the HR-set for each hop

(5) Client measures CRMs for every CR in its CR-set and HRMs for every HR it has seen (across all HR-sets obtained in that trial)

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

3.0.2.2. Valley Prevalence. In order to establish the prevalence and significance of latency valleys, I now compare HRMs to their respective CRMs. It is important to note that it is possible that a client has multiple CRs, and hence, multiple CRMs, and in practice a client can only choose one replica. Therefore, I compare *all* HRMs to the *minimum* CRM obtained in

Table 3.1. 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%

their trial, thus establishing a *lower bound* for expected performance gain, and allowing us to easily assert any gains or losses provided by the alternative replica choices (the HRs).

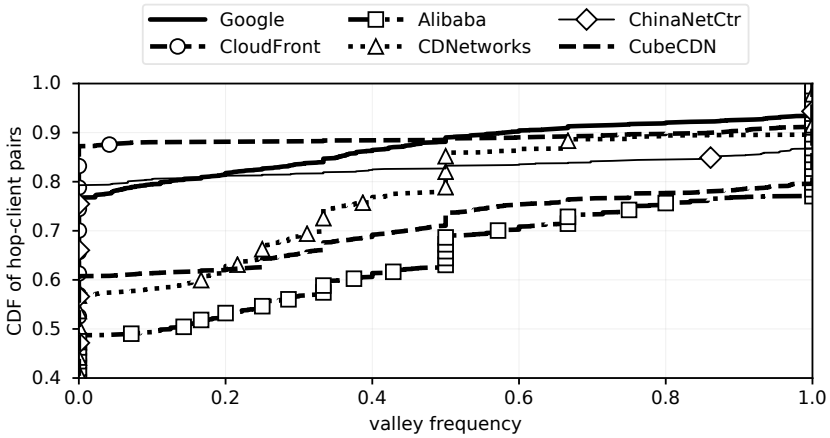
With the PlanetLab dataset, I compared each HRM to the minimum CRM, *i.e*, the best client replica, obtained in the same respective trial. Figure 3.3 shows the results of this assessment. HRM is plotted on the y axis and minimum CRM on the x axis, thus creating a visual representation of the latency ratio. I distinguish between the CDNs using different colors. To better explain the results, the equality line is drawn 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 3.1 shows the results, *i.e.*, Column 2 (% Valleys Overall) lists these percentages for each CDN.

Having established that valleys exist, I wish to determine where we can expect to find valleys. I 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, I now also begin choosing an individual replica from an HR-set. Instead of choosing

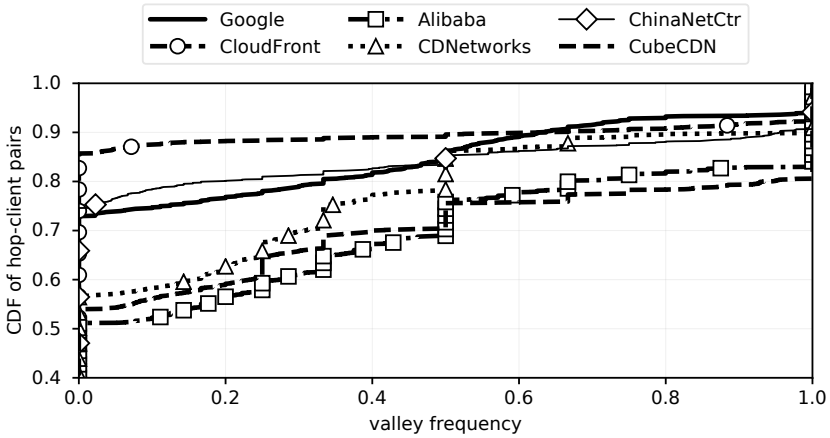
the minimum HRM for a given hop, I conservatively choose the *median*. Each hop's chosen HRM is the median of its HR-set for that trial. I continue this pattern (choosing the client's best CRM from the trial and a hop's median HRM) for the remaining PlanetLab data analysis. The PlanetLab data thus represents a *lower bound* on valleys and their performance implications; I compare the *median* performance of my proposed procedure to the absolute *best* the existing methods have to offer. In Section 3.0.4, using a RIPE Atlas-based testbed, I remove this constraint to demonstrate the real-world performance of the system.

I 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 3.1 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.

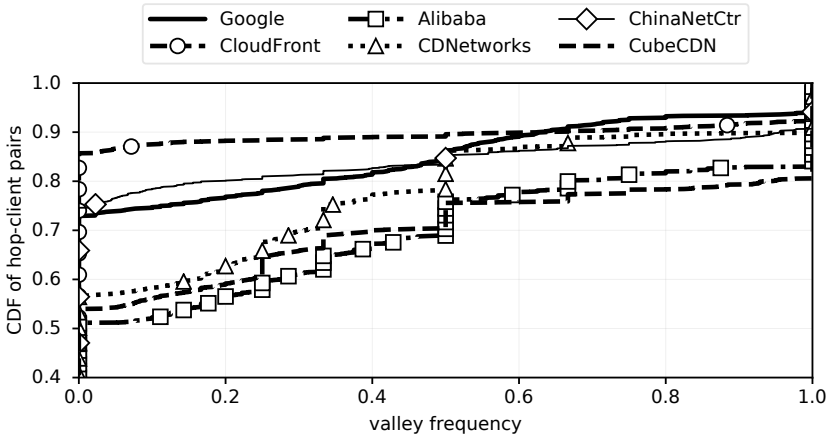
Do Valleys Persist? Next I assess whether subnets are persistently valley-prone. For some number of trials, how often can the client *expect* a valley to occur from some particular hop? To answer this, I to be able to describe how frequently valleys *occurred* for some hop subnet in a given set of trials.



(a) ping



(b) total download time



(c) total post-caching download time

Figure 3.4. CDFs of clients by valley frequency. In 3.4a, the subnet-response measurements are ping times averaged from bursts of 3 back to back pings. In 3.4b, the subnet-response measurements are total download times on first attempts, while in 3.4c 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). Down-

Valley frequency I 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

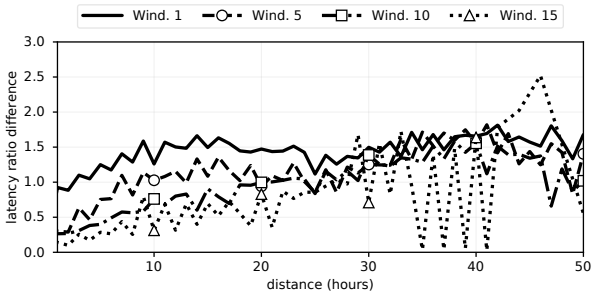
$$v_f = \frac{v}{N}.$$

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

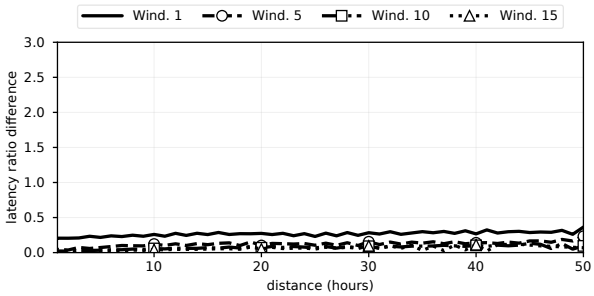
Figure 3.4 shows the results. Note that Figures 3.4b and 3.4c use the total download time and the post-caching total download time, respectively, for the subnet measurements as opposed to pings.[2] *I further note that their results closely follow those obtained using pings, as in Figure 3.4a*. For simplicity, and because the RIPE Atlas platform cannot perform arbitrary web oject downloads, I revert to only using pings for the remainder of this project.

Figure 3.4 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 the system with more opportunities to employ subnet assimilation and, ideally, improve performance. Column 5 (% Hop-Client Pairs with Valley Frequency > 0.5) of Table 3.1 shows the percentage of hop-client pairs that have valley frequencies (calculated across all

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



(a) All client-hop pairs



(b) Valley at least once

Figure 3.5. In both figures, I compare the change in latency ratio between two trial windows to the distance in time between the two windows. In figure a, I use all hop client pairs. In figure b, I restrict the set to pairs that experience at least one valley in at least one of their 45 trials.

45 trials), greater than 0.5, *i.e.*, hop-client pairs that incurred valleys in the majority of their trials.

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 my 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, I 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 I plot this against the time distance between the windows, we can observe the trend in latency ratios as the distance in 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 3.5 plots the latency ratio versus distance in time for several window sizes, ranging from 1 to 15. In these plots, I use the windows' median latency ratio values for comparison. In particular, in Figure 3.5a, I use the 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 3.5a 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. I hypothesize this results from unmapped subnets receiving generic answers, as observed in [**111**]. However, subnet assimilation requires that I have some idea of how a subnet will perform in advance.

Since valley-prone subnets are of particular interest to us, in Figure 3.5b, I reduce the 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 3.5b. 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.

Valley Utility Analysis. Figure 3.6 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 I 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 3.6 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 tested 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 3.6 that ChinaNetCenter's interquartile range spans over 40% of the possible value space (between 0 and 1). With such a wide

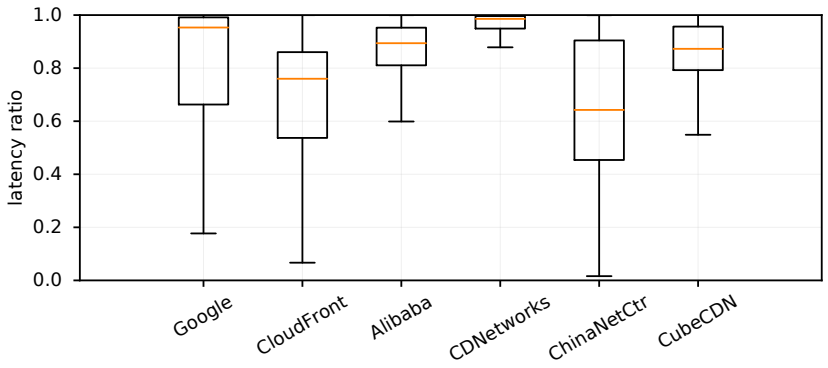


Figure 3.6. Lower bound of latency ratio of all valley occurrences.

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, tightening the definition of what's "good enough" for subnet assimilation, as I evaluate in Section 3.0.4.1.

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 my technique to improve its replica selection. I 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 [**53**]. 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 the system's valley-prone hop selection by filtering out "shallow" valleys from consideration. I demonstrate and discuss the effects of different levels of selectiveness in Section 3.0.4.1.

3.0.3. Drongo System Overview

I 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 its 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).

3.0.3.1. Window Size. I now face the question: What is a sufficient amount of data needed by Drongo to ensure quality subnet assimilation decisions? I 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 observed in Section 3.0.2.2, 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, I set Drongo's window size at 5.

3.0.3.2. Data Collection. Drongo must execute trials, defined in Section 3.0.2.1, in order to fill its window and collect sufficient data. As demonstrated in Figure 3.5b, *when* these trials occur is of little to no significance. In my experiments, I perform trials at randomly sampled intervals;

the trial spacing varies from minutes to days, with a tendency toward being near an hour apart. This sporadic spacing parallels the variety of timings I expect to happen on a real client: the client may be online at random, unpredictable times, for unpredictable lengths of time.

3.0.3.3. Decision Making. Here I 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 3.5b, 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 3.6, 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. I explore the effects of changing the v_f parameter in Section 3.0.4.

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.

3.0.4. Drongo Evaluation

Here, I 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.

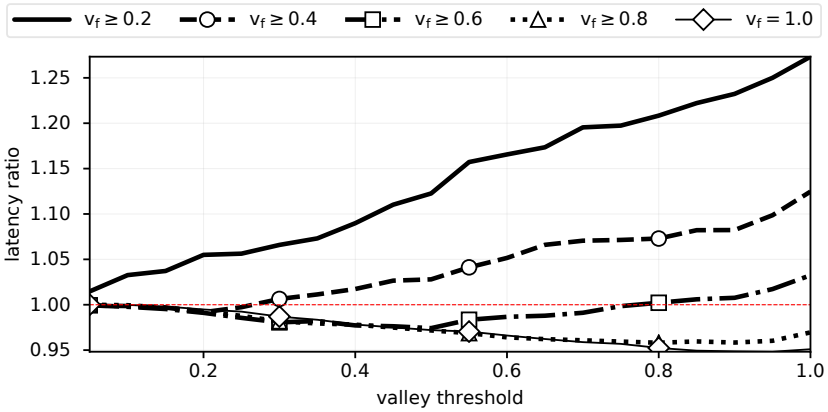


Figure 3.7. Average latency ratio of overall system as I vary v_f and v_t

Using the RIPE Atlas platform [**16**], I collect a trace of 429 probes spread across 177 countries for 1 month. In my experiments, I use the first portion of a client's trials as a training window. Following the completion of the training window, I 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 my 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.

3.0.4.1. Parameter Selection. Figure 3.7 shows the effects of Drongo on my entire RIPE trace's average latency ratio, *i.e.*, I 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 3.0.1.3, 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 3.7 shows the average results, across all tested clients. I draw 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 the selected 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 my prior findings from Figure 3.5b.

Meanwhile, two more characteristics stand out as v_t is varied. First, the valley frequency parameter can completely alter the slope of the latency ratio plotted against v_t. This behavior echoes the observation 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 the valley threshold rises. 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 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.

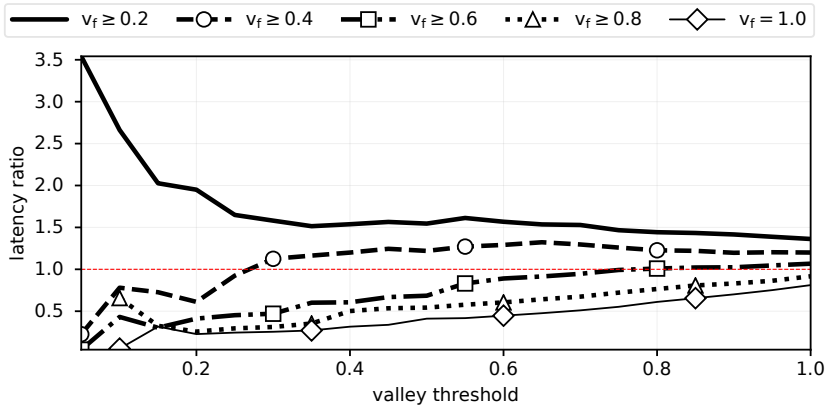


Figure 3.8. Average latency ratio of cases where subnet assimilation was performed

Figure 3.8 plots the average latency ratio in the same manner as Figure 3.7, 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 my knowledge of Figure 3.7, 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 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 3.9 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 3.9, 69.93% of tested 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.

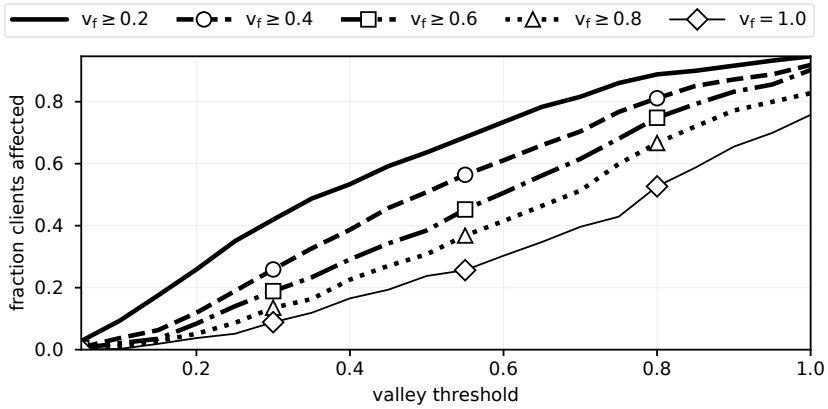


Figure 3.9. Percentage of clients where subnet assimilation was performed

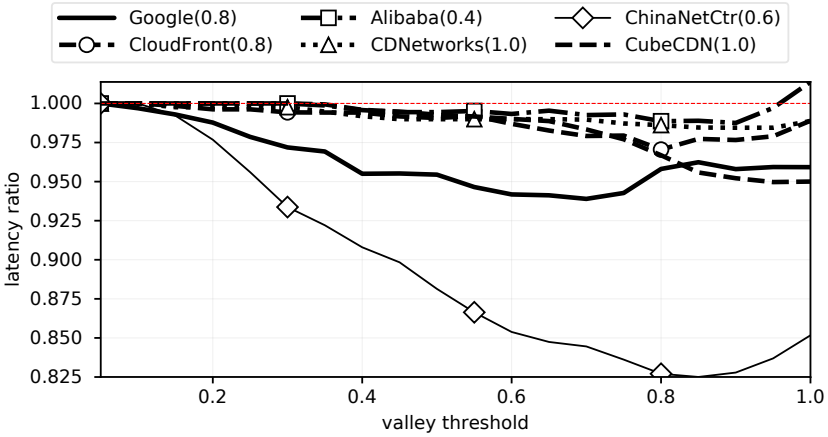


Figure 3.10. Per-provider system performance for all queries. Optimal v_f is set for each provider and noted in parentheses

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

3.0.4.2. Per-Provider Performance. In the previous subsection, I used a constant parameter set across all six providers in my analysis. In this section, I analyze Drongo on a per-provider basis. By choosing an optimal parameter for *each provider*, as done below, Drongo's aggregate

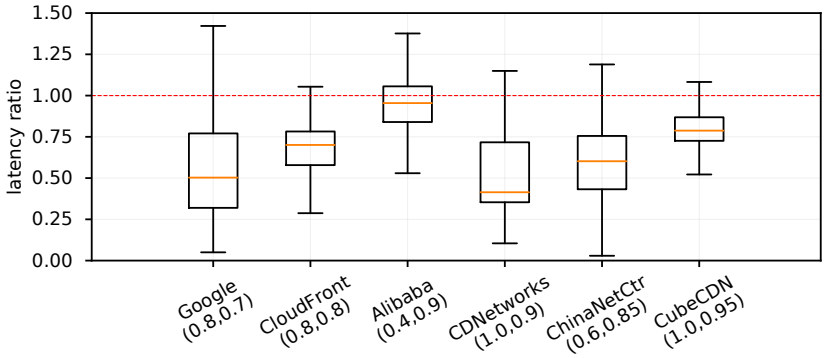


Figure 3.11. Per-provider system performance for queries where subnet assimilation was applied. Parentheses contain optimal values for each respective provider, formatted (v_f, v_t)

performance gains increases to 5.85%. In Figure 3.10, I 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 3.0.4.1 (1.0). While the intricacies of each provider's behavior clearly hinge on opaque characteristics of their respective networks, I offer several explanations for the observed performance. CDNetworks, which sees small gains across all valley threshold values, further validates the hypothesis proposed in Section 3.0.2.2 regarding anycast networks. Meanwhile, Google, the largest provider of my 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 3.11 shows Drongo's performance, per-provider, exclusively in scenarios when it applies subnet assimilation. Comparing to the results shown in Figure 3.6 for the Planet Lab experiments, we observe significant differences. Most notably, the latency valley ratios are

much smaller in Figure 3.11 than in Figure 3.6. For example, the Google's median latency ratio is close to 1.0 in Figure 3.6, indicating insigificant gains. On the contrary, Figure 3.11 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 3.6 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 I used all the valleys under the valley threshold of 1.0, here I use optimal v_t values, such that Drongo successfully filters out most "shallow" valleys that would dilute performance gains.

3.0.5. 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.*, [**39, 71**]. 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 [**51, 94, 117**] in the Web domain, and likewise in streaming, *e.g.*, [**77**]. 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 [**115**], 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 [**55**], 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.

3.0.6. 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 my efforts to respect their mechanisms in Drongo's design. I propose that a broadly deployed form of Drongo could be carefully tuned to effect only the most extreme cases. Figure 3.11 shows that subnet assimilation carries some risk of a loss in performance, so it is in clients' best interests to apply it conservatively. First, I know from Figure 3.9 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 ([**52, 76**]). 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. I leave this modification for future work.

3.0.7. Summary

In this thesis, I 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. I have introduced and explored latency valleys, scenarios where replicas suggested to upstream subnets outperform those provided to a client's own subnet. I have asserted that latency valleys are common phenomena, and I have found them across

all the CDNs I have tested in 26-76% of routes. I 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.

I have introduced Drongo, a client-side system that leverages the performance gains of latency valleys by identifying valley-prone subnets. My 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.

CHAPTER 4

Skylines: Demystifying Network Resource Islands with Virtual Landmarks

4.1. Introduction

You and the person next to you might not be using the same Internet. With ever increasing diversity and interlinking of online services — content distribution networks, cloud computing, CDN brokers, cloud brokers, ad brokers, load balancing, user tracking, geoIP, and more — even the implications of loading a single web page are no longer straightforoard [**50**]. Often, failure to recognize the whole as distinct from the sum of its parts has inhibited progress and hampered performance in networking technology [**47, 80**]. In the same way a city's skyline cannot be anticipated by the artitect of a single building, the "digital skyline" of the Internet can be neither predicted nor fully controlled by any single entity. However, skylines can always be *observed*.

Even across a single network service, client experiences may diverge. In Figure 4.1, I provide a high level illustration of how this can happen. In subfigure 4.1a, a client intending to connect to example.com submits a DNS query. I do not show the minute details of the DNS resolution process, which is itself multi-tierd and possibly involving cooperation from many separate stakeholders. What is important to know is that eventually, the client's request reaches the nameserver responsible for example.com. The nameserver uses what is often internal, proprietary logic to decide which of example.com's network resources the client should be connected

to. In subfigure 4.1b, we are reminded that this client is not the only one from its subnetwork to access example.com. However, as illustrated, the client's peers may not necessarily be directed to the same content resource, despite having carried out essentially the same DNS resolution process and possibly sharing the same edge network. This potential for mismatch between clients only grows as the number of domains considered increases — which it will, often on a single web page.

In this project, I explore the complex combination of independently operating resource allocation schemes and assess their behavior in *aggregate*. To enable my research, I introduce a new similarity measure, common network resource exposure (CNRE), which captures the extent to which a pair of clients are directed to the same network targets as each other across a broad set of domains. CNRE is, to my knowledge, the first ever method to quantify cross-provider DNS redirection patterns and their collective behavior.

I test and assess CNRE using 302 web content hosting domains for each CNRE calculation. To do this, I collect latency and DNS measurements for each domain from each of 9,024 globally distributed clients and perform over 40 million pairwise CNRE calculations between them. My experiments validate common network research exposure as a useful measure and explore its relationship with other client properties.

In order to understand CNRE, I performed an exhaustive set of measurements to frame client experience on a per *site* basis, as opposed to per individual domain. In this work, I capture a snapshot of both DNS resolutions and latency measurements toward the 304 domains that appeared most frequently in the top 2441 most popular webpages. My measurements span over 9,000 unique clients spread across 185 countries and 3637 autonomous systems. I performed

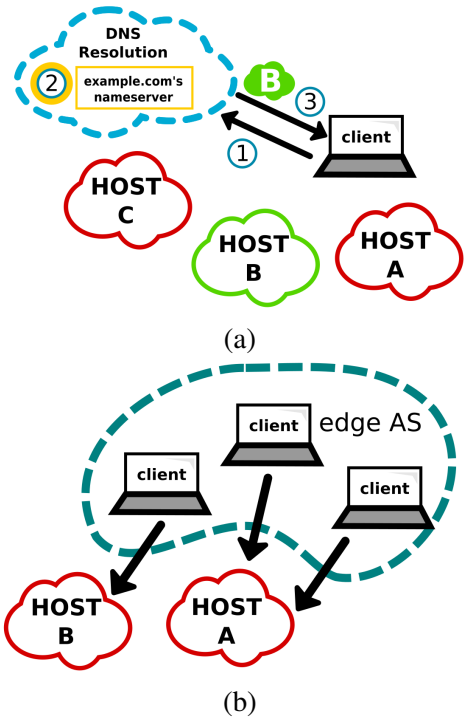


Figure 4.1. Illustration of network resource allocation. Figure 4.1a shows DNS resolution at a high level: 1) The client deploys a DNS query for example.com. 2) This query ultimately reaches nameserver responsible for example.com and decides which of example.com's network resources should serve the client. 3) The nameserver's resource selection is returned to the client. Figure 4.1b shows an example of how clients with similarly described locations may be directed to distinct network resources.

over 52 million pairwise comparisons with the results of these measurements to explore the patterns and implications of common network resource exposure.

This project makes the following contributions:

- I perform a large scale exploration of client network performance on a per webpage level. My raw results are publicly available on the RIPE Atlas platform.
- I introduce the common network resource exposure (CNRE) similarity measure, which quantifies the extent to which clients are directed to the same set of web resources.

- I quantify the degree of alignment between conventional grouping schemes (country, ASN, and BGP prefix) and CNRE.
- I identify clusters of clients that share especially high levels of common network resource exposure and analyze their properties.
- I approximate the effective geographic "centers" of CNRE clusters, where their target network resources are most likely concentrated.

4.2. Problem Space and Related Work

This projects aims to gain an understanding of which clients are directed to the same set of resources across many distinct domains. Its most direct and immediate use case is influencing probe selection in large scale Internet measurements. For researchers, likely unaware of the relatively hidden allocation schemes of the wide array of CDN platforms and other large content distributors, it is difficult to determine, a priori, the degree of similarity between clients. Knowledge of whether there is a high probability that a pair of clients are being directed to altogether different resources may be significant to their experiment design. This approach to experiment design is in line with RIPE Atlas, one of the largest client based measurement platforms, which maintains an exhaustive set of tags on all of their clients in order to help researchers and network operators filter and refine the set selected for their experiment [**16**]. Further, more abstract applications may include, but are not limited to, distributed denial of service mitigation [**90**] and CDN node deployment [**83, 113**].

The most similar body of related work involves anycast CDN catchment analysis, which aims to investigate the set of clients routed towards particular CDN points of presence (PoPs)

[**53, 90, 70**]. My work differs significantly in scope: to my knowledge, I am the first to investigate what I refer to as *aggregate catchments*, the joint behavior of many anycast CDN catchments as well as unicast CDN targets, spread across many content distribution platforms. Conversely, this related body work either focuses on individual platforms or specific services [**53, 90, 70**].

Several authors have attempted to discover the topology of large CDN platforms through large scale measurement studies [**42, 52, 46**]. While their findings are potentially of use in this project, their goals and contributions run parallel to what I aim to accomplish. They seek to identify the properties and locations of CDN resources; conversely, I seek to identify the target pools (sets of clients) of overlapping CDN resource catchments [**42, 52, 46**]. Other work close to this space investigates the performance of a particular CDN deployment scheme [**54**].

To my knowledge, no existing body of work has attempted to quantify the extent to which clients are exposed to the same web resources across domains. However, work concerning security in internationally networked platforms do take network resource exposure into account. For example, some work has identified scenarios where traffic routed through certain neighboring countries is sometimes censored or manipulated by middleboxes in the transit country [**44**]. To the same end, the authors of [**66**] developed a means to guarantee traffic is not exposed to specific territories. In a similar light, many globally distributed platforms employ "geoblocking" to restrict content access to particular regions [**88**]. As a new way to quantify resource exposure in general, my work runs parallel to and may be of use to these areas of study.

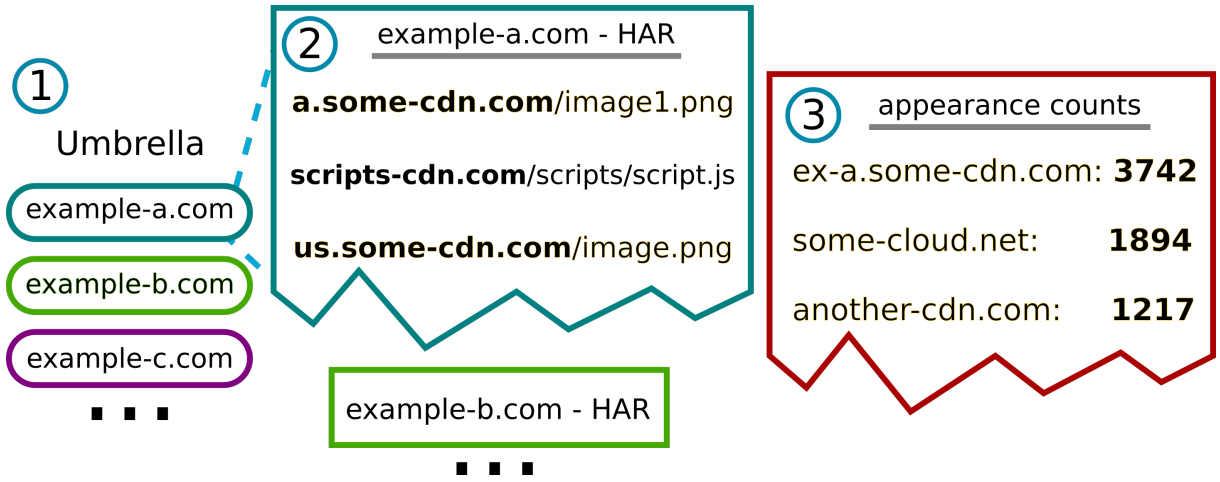


Figure 4.2. Diagram illustrating domain name collection: 1) Domains from the Umbrella top 1-million were loaded via Google Chrome to identify human-targeted websites. 2) For each human-targeted website's landing page, a HAR file was recorded. 3) Domains were extracted from HAR data and ranked by the number of times observed.

4.3. Experiment & Data Collection

The main preliminary steps performed to enable my work are twofold: 1) domain name collection and 2) per-provider performance measurement. The remainder of this section details these steps and the reasoning behind them while providing necessary context to understand the results presented throughout this project.

4.3.1. Definitions

As my aim in this project is to explore the cross-provider behavior of Internet resource-to-client mapping schemes, it is necessary to first establish what qualifies as "cross-provider" and what sort of cross-provider behavior is of relevance. For example, the reader may have observed that, if a pair of providers are not used *together* for a given online experience, there is no reason not to keep their analyses separate. I choose to focus on the providers of webpage

objects, which are known to often span a multitude of providers [**50**]. Previous work has well documented the impact of individual, slow loading objects on page load time [**116**]. To this end, I target domains which I empirically found to co-inhabit large numbers of webpages as web object hosts. Throughout this project, I equate "domain" to "host" or "provider", recognizing, however, that it is often the case that a single provider will use several domain aliases.

Likewise, I also note here that my use of the term "[web] resource" is deliberately ambiguous: the explicit implementation method used by each provider — ranging from a single subnet per geographic point-of-presence to a number of software-partitioned subnets per machine — is opaque and beyond the scope of this study. My chief concern is that an identifiable distinction is made between the set of targets (IP addresses) provided in DNS answers: the sheer fact that they are not labeled as the *same* target indicates that there is likely some difference, performance or otherwise, between them. For simplicity, I treat each /24 IPv4 subnet (generally, the most fine-grained BGP prefix route announcement allowed, by convention) as a potentially distinct resource, noting that it may be the case that larger providers operate with smaller (more coarse grained) prefixes.

4.3.2. Domain Collection

I use the top 10,000 most frequently resolved domains from Cisco's Umbrella Top 1-Million list [**108**] as a starting point. However, as this list is obtained from the perspective of DNS resolution, the relationship *between* these domains is unclear. Further, as there is no complete URL information from such a perspective, there is no indication which domains are used for downloading web content, as opposed to providing some other service or interface. To address this, I attempt to load pages from this list and ultimately use domains providing web objects

discovered on each successfully loaded page. This process is detailed below and illustrated in Figure 4.2.

First I attempt to load each web page from the Umbrella list using Google Chrome. If a page loaded, its source was checked for any indication that the page was not intended for human use (for example, automated server response pages for non-200 HTTP status messages). This filter reduced the size of the domain set from 10,000 pages to 2,441 pages. For each of these pages, an HTTP archive (HAR) file was saved to capture the full set of web objects loaded with the page. By using HAR files instead of just the page source, I avoid missing any dynamically loaded objects that may not appear in the original source. The HAR file provides the full HTTP path of each web object retrieved. Domains used in the remainder of this project come from this resulting dataset.

Due to security related rate limits, my experiment's progression was capped to 15-20 domain measurements per client per day, thus further restricting the number of domains to be used in the experiment. Since the entire domain set obtained was larger than what I could reasonably cover, I *ranked* object hosting domains by how frequently they were observed across the set of HAR files. The most frequently appearing object hosting domains were given priority. Based on this prioritization, I arbitrarily used the top 304 domains from this set.

In Figues 4.3 and 4.4, I show the decreasing marginal impact of each additional domain in the set. As shown, both quantities — the number of visited pages including a URL hosted by a domain from the set and the fraction of URLs on each page covered by domains included in the set — exhibit logarithmic-like growth patterns, beginning to plateau well before 100 domains are reached. We assert that this demonstrates the aggregate behavior of the 304 domains obtained above should sufficiently cover the domain diversity of a "typical" popular web page.

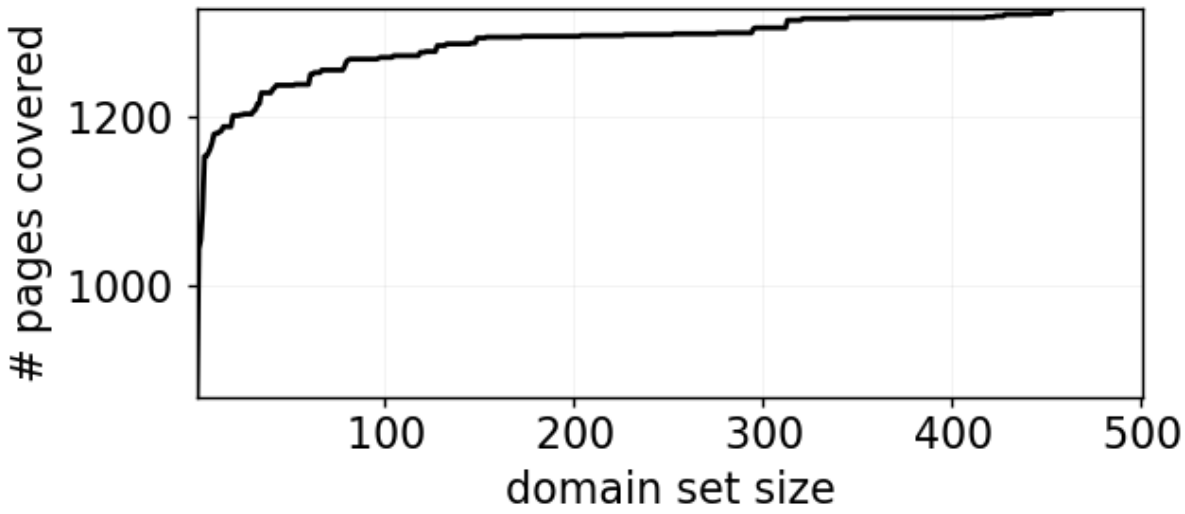


Figure 4.3. The number of sites containing an object hosted by an included domain vs the size of the set of included domains.

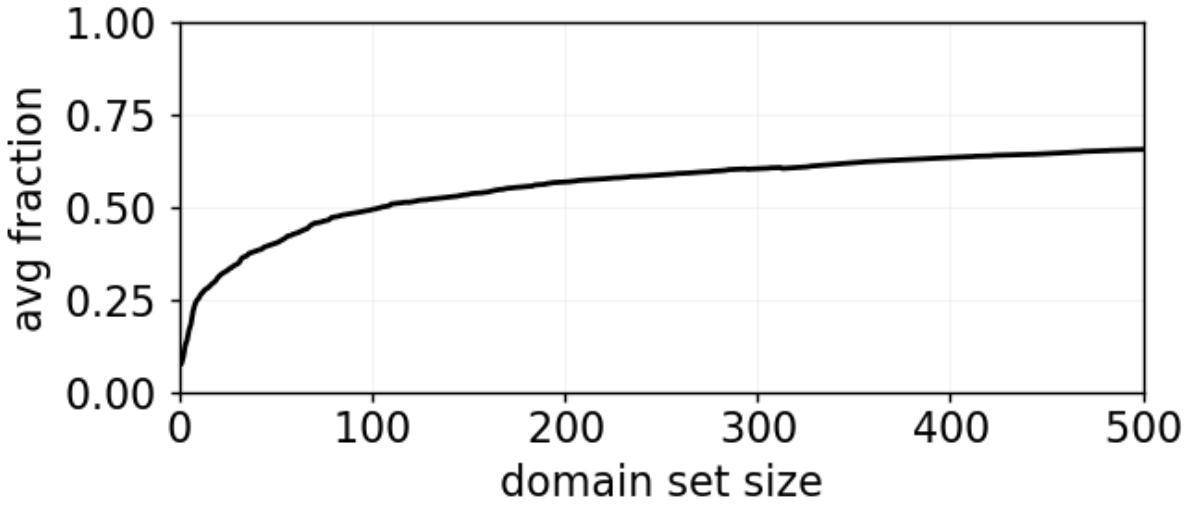


Figure 4.4. Mean fraction of page object links (URLs) covered per site vs the number of domains used.

By inspection after measurements were collected, two of the 304 domains were found to use a single, global IP address for all clients as opposed to performing redirection. Such behavior is 1) not interesting from a technical perspective in the context of this project and 2) evidently rare

(only occurring in two of the 304 tested domains). Further, given my analysis methodology, the inclusion of such domains would essentially only serve to offset CNRE values — which themselves are only meaningful in relative comparisons — by a constant factor, having no impact on results. For this reason, I opt to prune these domains from the list ultimately used, reducing the set size to 302. It is possible that these two domains operated using a fully anycast model as opposed to DNS redirection 4.7.2. I consider the possibility of ammending support for single-IP anycast hosts in Section 4.7.

4.3.3. Per-Provider Performance Measurement

Any attempt to identify the general groups that Internet clients are mapped to requires a dataset with a uniquely broad scope: not only breadth — a diverse set of clients — but also depth — many clients from each, yet to be uncovered, group or cluster. In addition, we are required to minimize the temporal spread of the measurements, as network resource allocation is known to change over time. I utilize the RIPE Atlas platform [**16**] for my measurements. RIPE Atlas offers a large number of globally distributed clients, capable of performing lightweight network measurements, such as pings, on behalf of configurable requests received by the Atlas API. I deployed ping measurements to the previously described 304 domains from 10,274 of RIPE's clients. Each client performed DNS resolution for pings via their local DNS resolver, ensuring that they each targeted the web resource they would ordinarily be directed to.

Unavoidable flux in the availability of individual, voluntarily maintained clients lead to some clients performing only a subset of the given measurements, thus missing some of the domains of interest. To be sure that this does not dramatically affect my findings, I arbitrarily enforce a minimal amount of domain coverage — 160 domains, just more than half of the set

— for use of a given client's data. I show in Section 4.4 the effects of domain quantity on measurement results. Applying this contraint reduced the size of the client set to 9,024 clients.

4.4. Common Network Resource Exposure

This project seeks to explore aggregate network resource catchments — the set of users exposed to the same set of web resources as each other across a given set of domains. I introduce a new similarity measure, which I have coined common network resource exposure (CNRE), to quantify the extent to which two clients are exposed to the same network resources. Inspired by the Jaccard index [**79**], CNRE between two clients, C_1 and C_2, is defined as follows:

$$\text{CNRE}(C_1, C_2) = \frac{\sum_i^D m_i(r_{1_i}^{-1} + r_{2_i}^{-1})}{\sum_i^D (r_{1_i}^{-1} + r_{2_i}^{-1})}$$

where D represents the intersection of measured domains between both clients, and m_i is 1 if C_1 and C_2's i^{th} domain answers match (otherwise zero). The values r_{1_i} and r_{2_i} represent the fraction of all measurements (across the entire dataset) that matched C_1's and C_2's DNS answer for that domain, respectively. For example, if C_1's answer for domain i appeared 900 times across the 9024 times it was tested in the dataset (once by client), $r_{1_i} = 900/9024$, or 0.09973 (*i.e.* roughly 10% of the answers). In other words, r_{n_i} captures the *rarity* of the DNS answer received by client C_n for domain i.

In my calculation of CNRE, I use the inverse of r_{n_i} to add increased weight to the impact of a mismatch on rare answers. CNRE is therefore designed to be higher between clients with more matching rare answers. As it is ultimately a measure of similarity between clients, CNRE values range from 0 through 1, with 1 being the most similar. To ease discussion in the remainder of

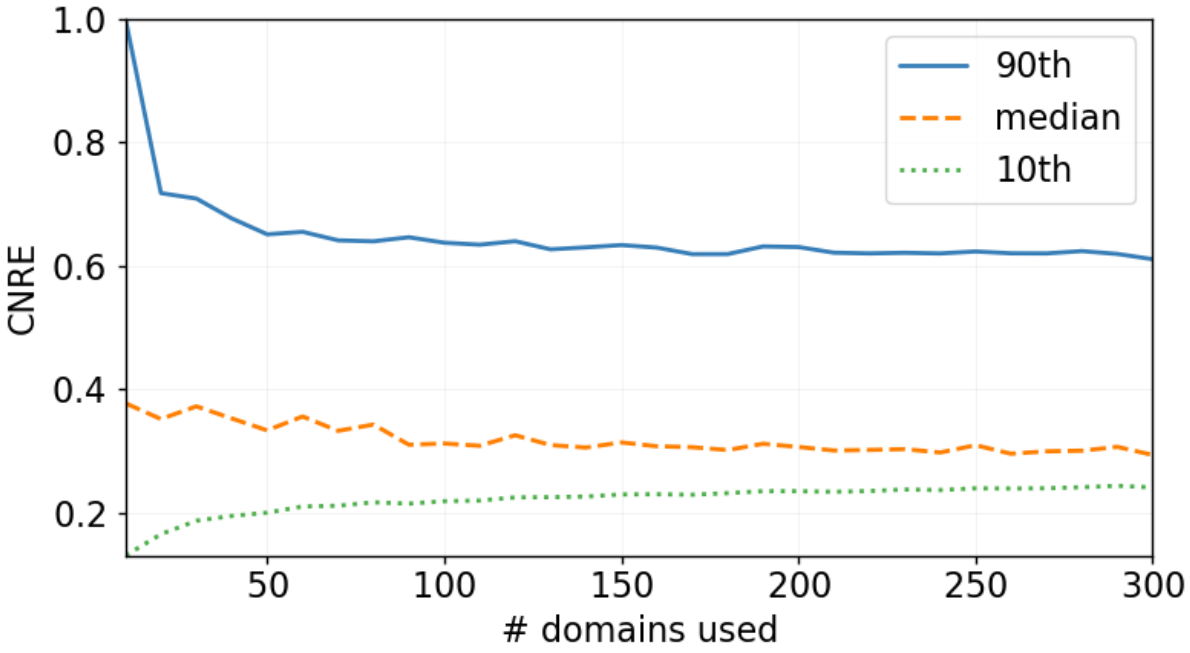


Figure 4.5. CDF showing the effect of the number of domains used for CNRE calculation. The 500 clients and set of domains used for each CNRE calculation were randomized.

this project, here I also define CNRE *distance* as $1-$. Likewise, where ambiguity is likely, I will refer to raw CNRE as CNRE *similarity*.

In Figure 4.5, I plot the effects of the number of domains on CNRE statistics. For each data point, I performed pairwise CNRE calculations between 500, randomly selected clients. For each individual comparison, a random set of domains were selected, matching the quantity being tested. Both upper (90^{th} percentile) and lower (10^{th}) CNRE scores stabilize and plateau significantly by 150 domains, showing no substantial changes as the number of domains increases beyond that. This validates my use of 160 domains as a cutoff for a client's admission into the final dataset.

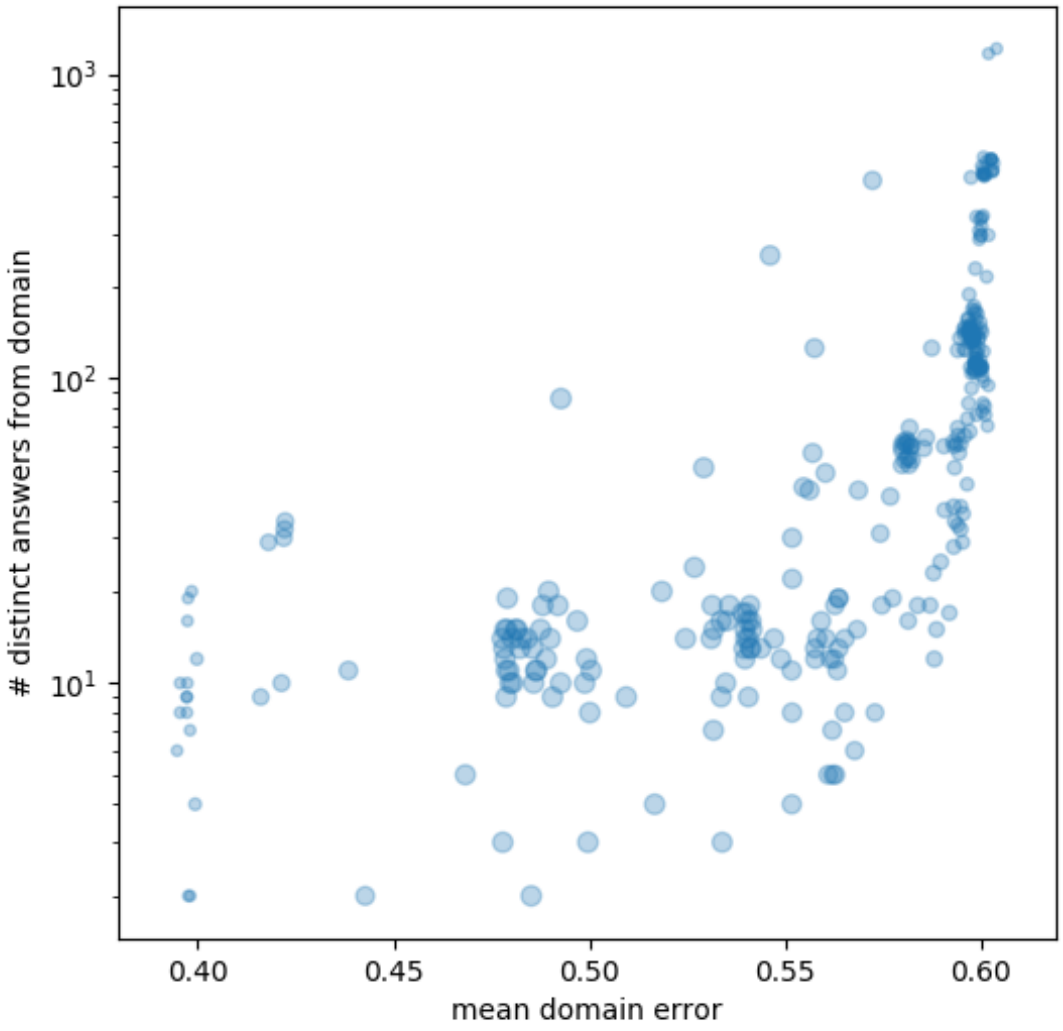


Figure 4.6. Mean domain error vs # of distinct answers observed from domain (one point per domain).

I pause here to address potential bias given toward individual domains. As CNRE calculation gives increased weight to rare answers, there is the possibility that the allocation patterns of large providers (who often have more answer variety related to the scale of their networks) may

dominate the results. This would be detrimental to the main purpose of CNRE — a supposedly aggregate measure — as it would ultimately revert to essentially measuring a single provider, which is a well explored topic. To determine whether this is occurring, I calculate the *domain error*, for a single domain, as follows: Given a pair of clients, first, find their CNRE normally. Next, let us set d to be 1 if the DNS answer for the domain of interest matches between the pair of clients, and otherwise 0. Finally, the domain error is the absolute value of $d-$CNRE. In Figure 4.6, I plot this, with each point representing the mean domain error for a given domain across the entire set of client combinations.

With domain error, I capture how different the CNRE would have been had that domain been the only one used in that client pair's CNRE calculation. A *low* mean domain error — close to zero — implies that the domain is dominating over the CNRE. A *high* mean domain error — close to one — implies that the domain has little impact on CNRE's value. A mean domain error close to the middle — 0.5 — is ideal, as it implies that the domain is neither dominant nor irrelevant. We see in Figure 4.6 that domains with many answers (and hence increased rarity per answer) actually have the highest error. Further, the range of domain error across all domains spans roughly from 0.4 to 0.6, indicative of the absence of substantial bias towards any given domain from my approach.

4.5. Finding High CNRE Clusters

Finding aggregate catchments — pools of clients essentially directed toward the same web resources — necessarily involves finding sets of clients with high CNRE measures between each other. Because CNRE is a measure of similarity, this problem naturally lends itself to hierarchical clustering techniques [**93**]. I employ the complete linkage method to ensure cluster

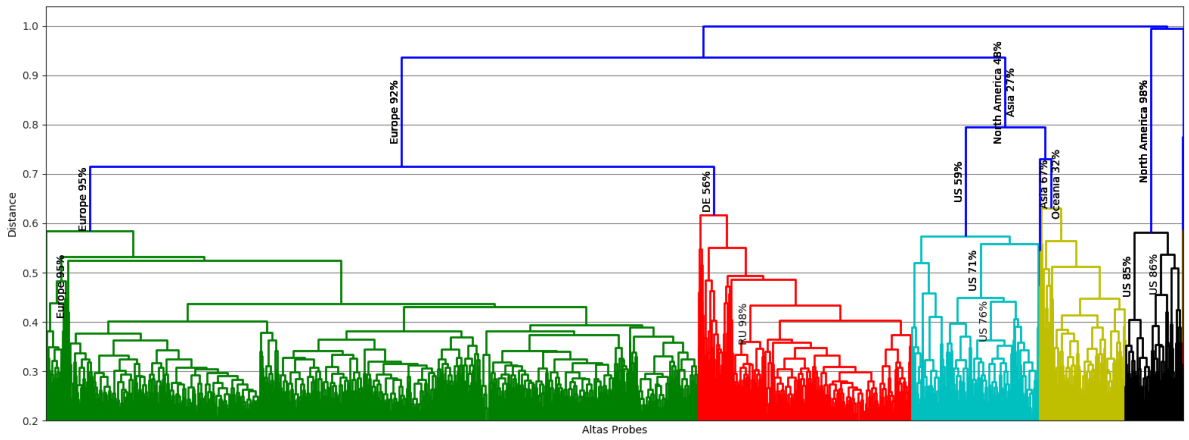


Figure 4.7. Dendrogram of CNRE distance across all client pairs

formation reflects commonalities across all cluster members as opposed to potentially edge-specific properties. Note that in all *clustering* calculations, I opt to use the CNRE *distance* (1−CNRE) as defined in Section 4.4.

Establishing hierarchical clusters requires that we have some definition of what constitutes a *high* or *low* CNRE measure and at what threshold it is appropriate to consider clients sufficiently similar such that they appear in the same cluster. In this section, I explore the implications of various CNRE values, as well as CNRE's relationship with other, well-established client grouping systems: country, ASN, BGP prefix, and /24 prefix subnet.

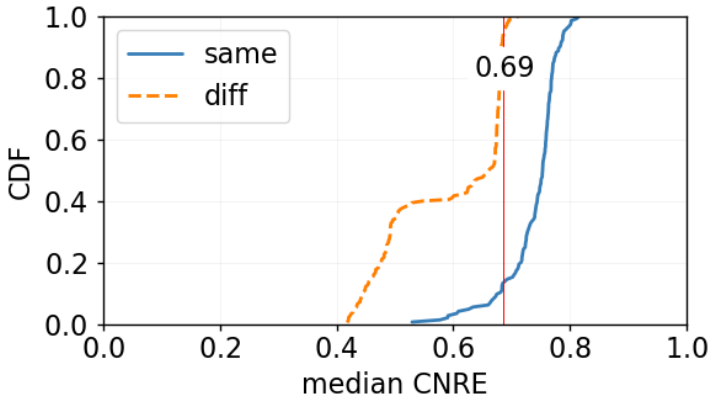
4.5.1. Group Formation Patterns

Figure 4.7 presents a dendrogram derived from pairwise CNRE distances across all clients and highlights two levels of distinct behavior regarding the distribution of CNRE distances. In the uppermost portion of the plot — where CNRE distances are beyond a threshold of 0.65 — we see that distinctions between branches and their implied client groups become well defined. There is a large cluster composed mainly of European and African probes (green cluster labelled

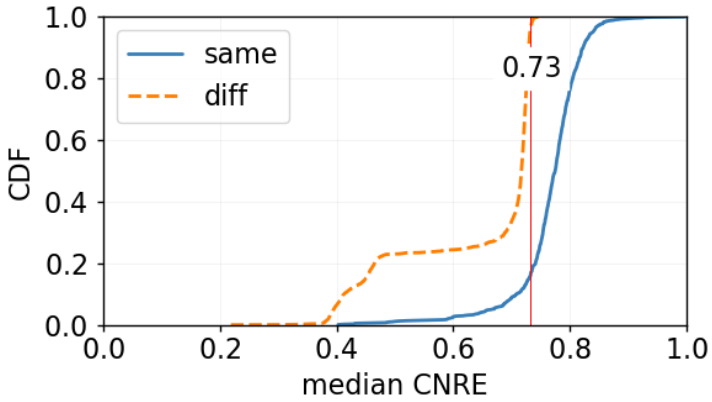
Europe 95%), one representing East Europe (red cluster labelled *DE 56%*), one composed of North and South American probes (blue cluster labelled *US 59%*), a cluster composed of probes from Asia and Oceania (khaki cluster), and one made exclusively with American probes (black cluster labelled *North America 98%*). In the lower region of the tree, where CNRE distances drop below 0.65, we see that branches begin to fork unpredictably with shorter changes in CNRE distance. Some of these branches may map to specific countries, but at finer granularities the geographical location of probes delineate poorly the characteristics of the clusters.

I compare established client group labeling schemes — country, ASN, and BGP prefix — to CNRE similarity between clients with matching (*e.g.*, same country) and differing (*e.g.*, different country) labels. My findings are shown in Figure 4.8. The 95th percentile CNRE between groups with differing labels is marked on each plot by a vertical red line; at this point, differing and matching labels become distinguishable. For example, in Figure 4.8b, a pair of clients with CNRE < 0.73 (see 0.27 in Figure 4.7, which uses CNRE distance) are likely from different ASes, while clients with CNRE > 0.73 are likely from the AS. Note, however, that 4.8 uses *median* values for all points shown. I analyze this further in 4.5.2.

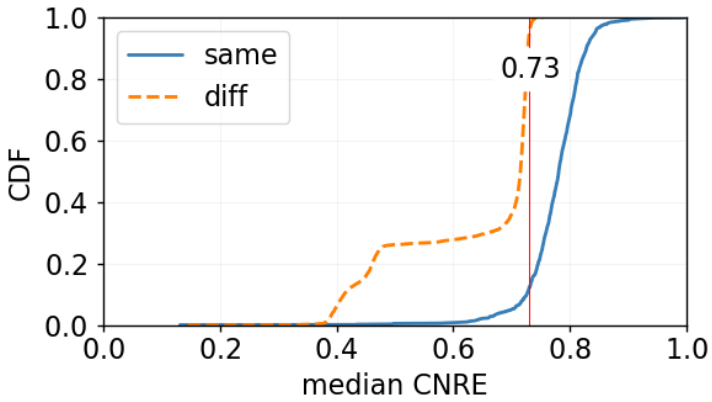
Figures 4.8 and 4.9 together help provide a possible explanation for three partiions observed in Figure 4.7. In Figure 4.8, we see that in all three subplots, the aforementioned middle region of Figure 4.7 appears again, this time as a plateau in both the "Diff" and "Same" curves. In this region, "Diff" and "Same" overlap significantly, rendering them indistinguishable. This transient zone is given further context in Figure 4.9, where I shade each country's median CNRE towards other countries (*i.e.*, outbound comparisons) — the same data used to plot the "Diff" CDF in Figure 4.8a.



(a) country



(b) ASN



(c) BGP prefix

Figure 4.8. CDFs of CNREs across client sets with matching (same) and non-matching (diff) labels. "Same" shows the CDF for the median CNRE distance across all client pairs matching a given label. "Diff" shows the CDF for the median CNRE distance from each label group toward all other labels. The red, vertical line in each subfigure marks the 95th percentile CNRE for for differing labels.

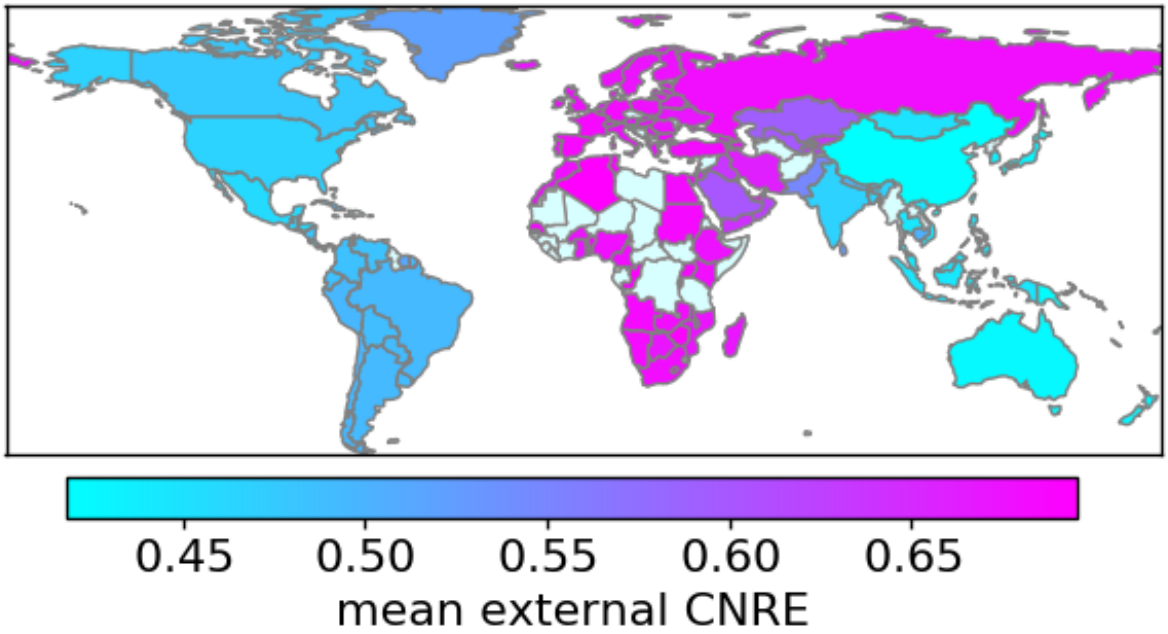


Figure 4.9. Choropleth with each country shaded by its median CNRE distance from all other countries.

If a given country tends to have low CNREs between itself and all other countries, this implies that the country is exposed to a more exclusive set of web resources than its peers. For example, Australia, which, as shown in 4.8a, has a generally low CNRE with other countries, likely utilizes very locale-targeted infrastructure given its relative distance from more more broadly used network resources. Likewise, China, which is well documented as having its Internet infrastructure deliberately disjoint from much of the world [**43**], also has a low CNRE with most other countries. Conversely, we see most that countries within Europe and Africa tend toward having higher CNREs with most other countries, implying that the majority of web resources exposed in those regions are neither exclusive nor fine grained.

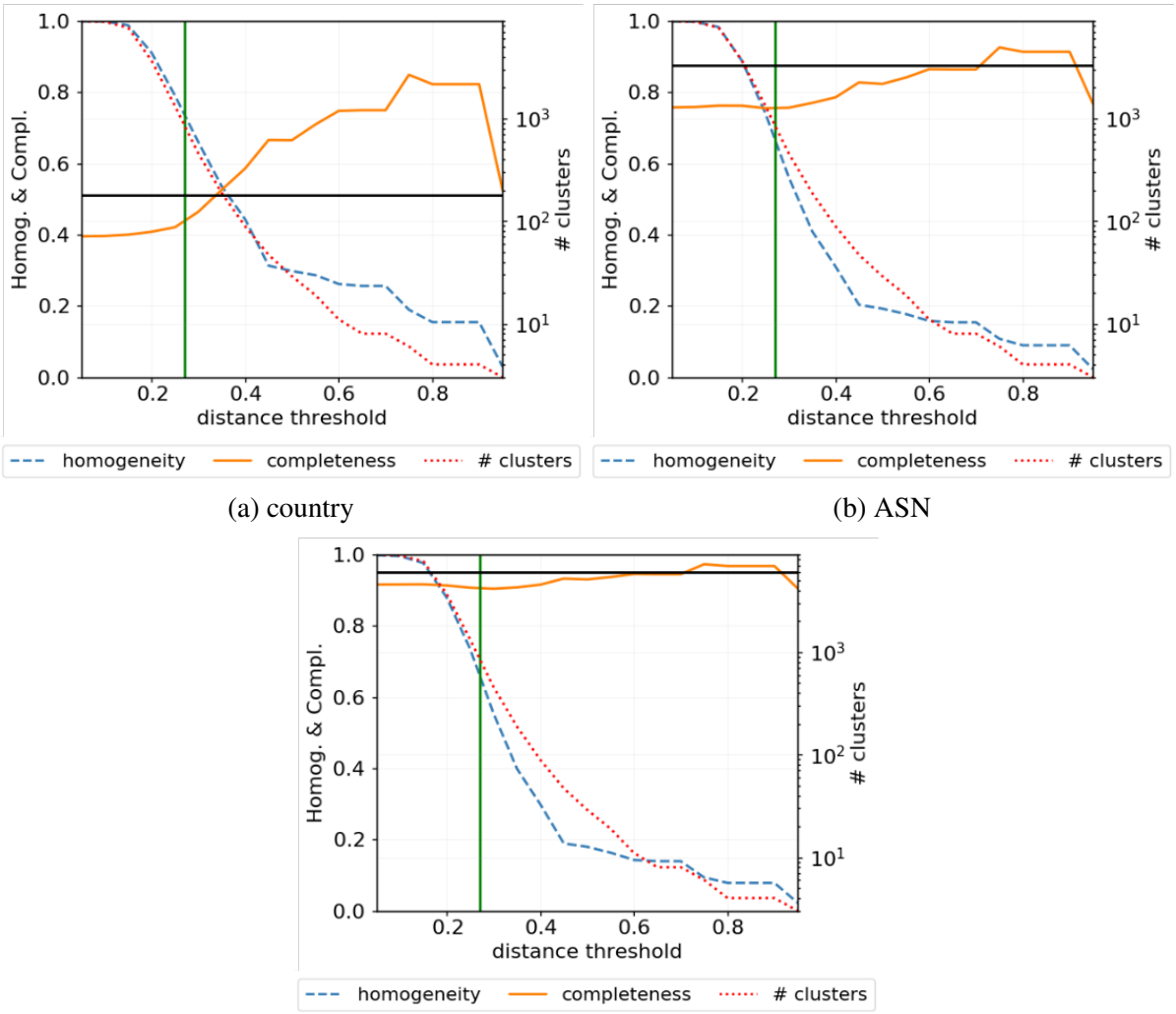
4.5.2. Label Alignment

Having established some concept of what constitutes a "high" or "low" CNRE measure, I further consider CNRE in comparison to country, ASN, and BGP prefix — three labeling schemes commonly used group Internet clients. Specifically, I wish to determine if the information captured by CNRE (the extent to which clients are exposed to the same web resources) is reasonably captured by any pre-existing system. If this were the case, one might argue that the premise of treating CNRE as a separate system would be redundant and arbitrarily complex. Therefore, I treat this subsection as a means of validating and justifying the CNRE as a separate, currently unaddressed concept.

In Figure 4.10, I plot the completeness, homogeneity, and number of clustesr for the aforementioned labeling schemes as I cluster clients in the dataset, varying the CNRE distance threshold used for cluster formation. In addition, I also mark, with a vertical line, the CNRE distance at which labels become distinct (see Figure 4.5.1), and I mark the number of labels (*i.e.*, the number countries, ASes, or BGP prefixes) present with a horizontal line (using the righthand y-axis).

If homogeneity and completeness, which together indicate how well cluster membership aligns with a given labeling scheme, is not high, the CNRE-related implications of a given label become ambiguous. We see in Figure 4.10 that for country and ASN, homogeneity and completeness are never simultaneously high, rendering them unusable for determining CNRE on their own. BGP prefix, however, requires more thorough consideration. In Figure 4.10c

Figure 4.7 depicts the hierarchical clustering dendrogram based on pairwise CNRE distances. This representation highlights the similarity of clients and different possible partitioning. Each node in the tree is a cluster composed of the underneath denser clusters. A horizontal



(a) country

(b) ASN

(c) BGP prefix

Figure 4.10. Completeness, homogeneity, and number of clusters versus clustering distance threshold. The vertical line marks 0.27, the CNRE distance at which clients with differing labels become distinguishable, and the horizontal line denotes (using the right-side y-axis) the number of different real labels (for example, the number of countries) preesent in the data set for the given labeling scheme.

cut in the dendrogram produces a partitioning of clients for which the dissimilarity of two probes in the same partition is not greater than the y-value where the cut is done.

For example, the colored clusters in Figure 4.7 are obtained with a cut at $y = 0.65$. This partitioning produces eight clusters including three small ones composed of outliers. The five large clusters represent a very coarse partitioning of clients. 95% of probes in the green cluster are from Europe and 4% are from Africa (that is almost all African probes), the red cluster consists mainly of German, Italian, and Russian probes (79%), the blue cluster is mainly North and South American probes (resp. 80% and 18%), the khaki cluster has mostly probes from Asia and Oceania (resp. 67% and 32%), and the black cluster is mostly North American probes (98%). As each cluster is composed of smaller and denser clusters, one can again partition these clusters and form groups with clients having more network resources in common.

4.6. Cluster Analysis

Now that I have built up an understanding of how CNRE behaves, I move on to investigate the aggregate catchments of clients using clustering techniques discussed in the previous section. In this section, I use the aforementioned CNRE threshold of 0.73 for cluster formation. This threshold yields 870 clusters. For analysis, in which I compare clients that cohabit the same cluster, I consider only clusters for which the dataset has a representation of at least three clients; this yields 612 clusters from the original 870. The average cluster size from this reduced set is 15.78 members, with a standard deviation of 9.0 and a median size of 14.19. Note that cluster size variety is significantly impacted by the dataset, which has more client representation in western Europe and North America than in the rest of the world where RIPE's influence is more sparse [**16**].

Here I examine each cluster's geographic spread — the closeness, in terms of geographic distance, of members of the same cluster. As clients sharing the same cluster are predominantly

exposed to the same network resources, the geographic spread of a cluster's clients must related to the network performance (latency) they experience. Specifically, if a pair of clients directed to the same network resource are physically "far" apart from each other, it is likewise impossible for *both* clients to simultaneously be near said resource. In such an arrangement, the resource is either near one client and far from the other, or the resource is equidistant from both clients. In the latter case, if the clients are sufficiently far from each other, the resource must also far from both clients.

To calculate these geographic distances, I use coordinates for RIPE Atlas probes — our client machines in the context of this experiment — obtained from RIPE Atlas's API. RIPE acquires probe location information via manual input from volunteers who themselves maintain Atlas probes, and, where necessary, by autmated input from MaxMind [**31**]. Figure 4.11 shows the CDF of the mean geographic distance (in kilometers) between members of a cluster, for each cluster. Members of a cluster with a larger mean geographic distance are farther apart form each other, on average, than are the members of a cluster with a lower mean geographic distance. In the median case, we see an average client distance of 509 km. We observe that for 20% of clusters, members are over 1000 km on average.

Since CNRE potentially spans many, physically distinct resources, it serves as an *aggregate* measurement, and I do not attempt to pinpoint the location of any individual resource. Instead, I identify the effective "center" of each cluster and measure the effect of a member's distance from the center. Figure 4.12 marks a point for the the coordinates of each cluster's center. The disproportionately high number of centers located in Europe is byproduct of the distribution of RIPE Atlas probes, which are most densely concentrated in Europe where RIPE operates.

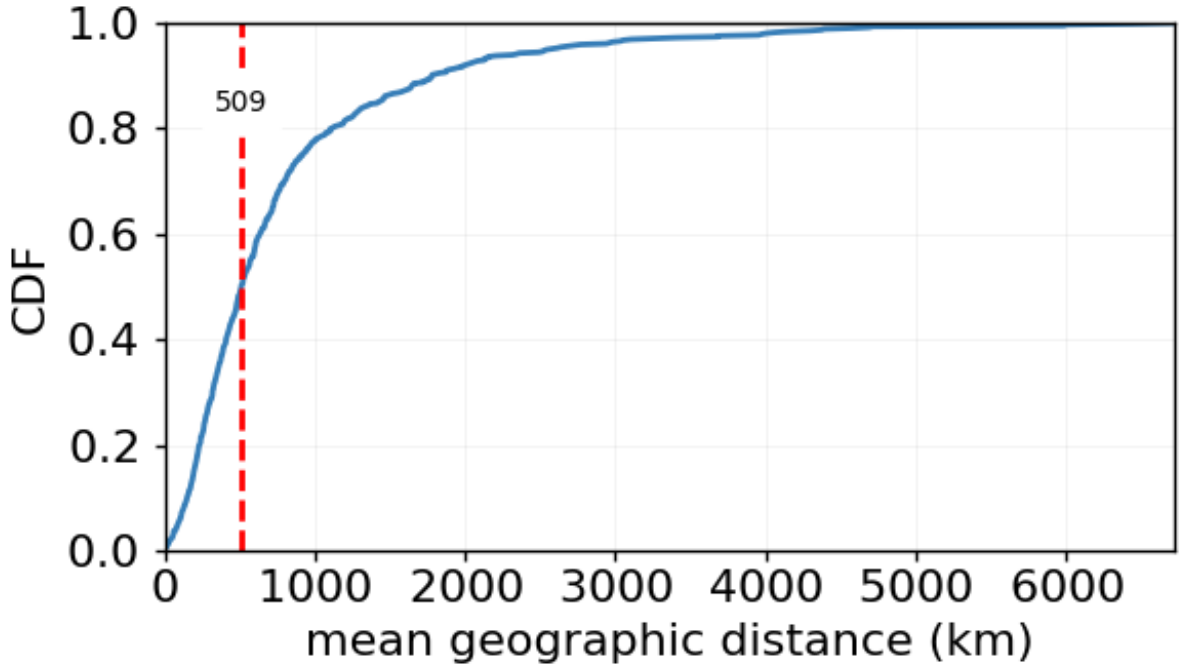


Figure 4.11. CDF of mean geographic distance between cluster members. The dashed vertical line marks the median.

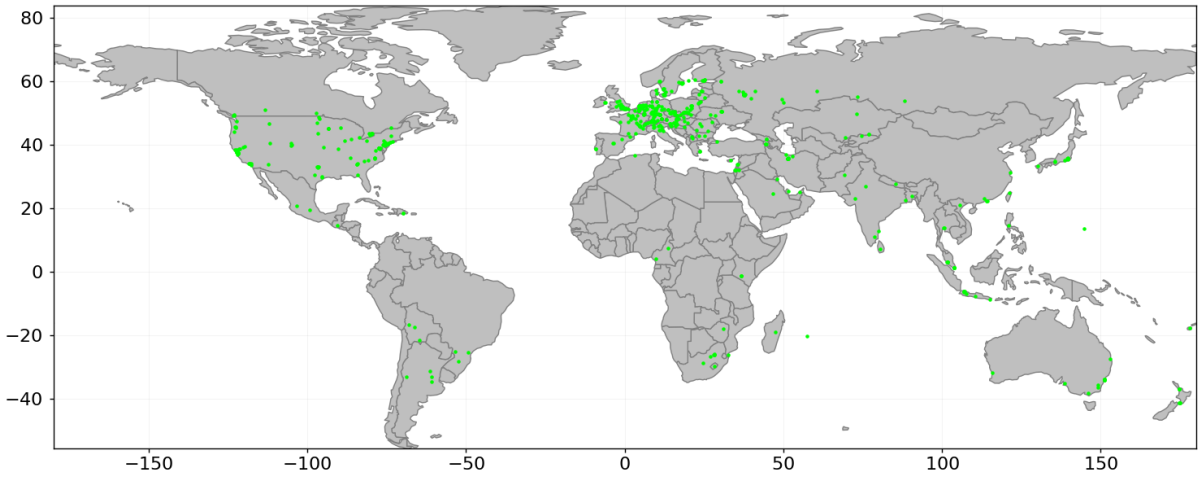


Figure 4.12. Map of world with point for each cluster's geographic center.

Following the intuition of DNS redirection laid out in other work [**52, 110, 45**], I hypothesize that the geographic center of a cluster will sit physically close the location of the most

of that cluster's network resources. By this logic, clients closer to their cluster's center should experience better network performance (*i.e.*, lower latency) than those farther away. To test this, I compared each client's mean latency (taken across all 299 ping responsive domains in our set) to that client's distance from its respective cluster's center. For simplicity, I use a cluster's geometric median as an estimate of its center. The results of this comparison are shown in Figure 4.13 as a scatter plot of mean latency versus distance from cluster center. Each point corresponds to a single client's latency and distance from its respective cluster's center. The figure also includes a best fit line, denoting the overall trend of the points.

Note the positive slope of points in Figure 4.13, indicative of a directly proportional relationship between latency and distance from the cluster's effective center. As a client's distance from its cluster's center increases, so does its latency. Performance for clients relatively near their respective centers — closer than 1000 km — is seemingly noisy and no trend is clearly observable. However, as the geographic distance increases beyond 1000 km, the directly proportional relationship between performance and center distance becomes more apparent.

Also note that the slope of Figure 4.13's best fit line quantifies this relationship as approximately 8.2×10^7m/s, which implies a general data speed of $\frac{1}{4}$th of the speed of light. This is comparable to the transmission speeds of the fastest network communication mediums in use at the time of this writing — approximately $\frac{2}{3}$rds of the speed of light [**72, 114**]. Traffic, routing complexity, and the pressence of lower speed mediums may account for the apparently lower speed in our finding.

As I have demonstrated that one's distance from their cluster's center impacts performance, clients should ideally share resources with the closest cluster center possible. Figure 4.13 raises an additional concern: many clients are very far away — often thousdands of kilometers — from

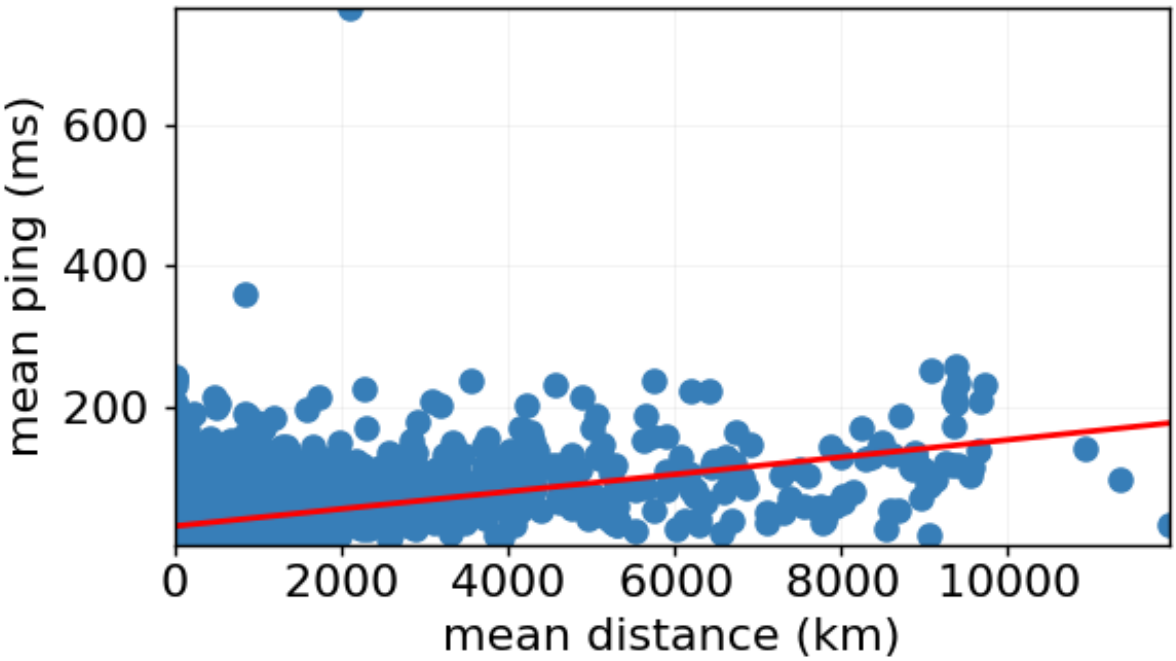
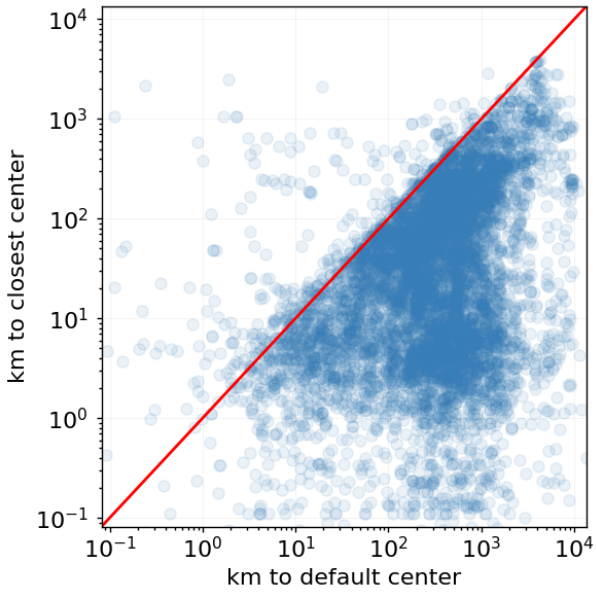


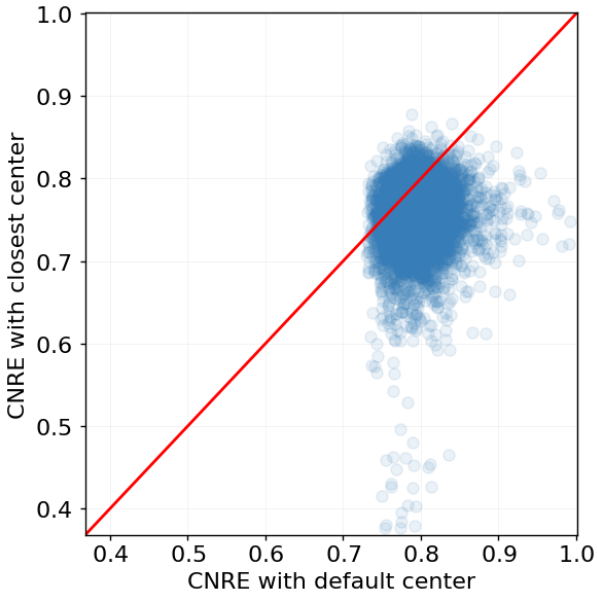
Figure 4.13. Scatter plot where, for each client, I compare the client's mean latency (across all responding sites) to that client's distance from its cluster's center. The line denotes a first order best fit curve for the scatter plot's points.

their respective cluster centers. For perspective, I remind the reader that the circumference of the world is approximately 40,075 km; several clients reside over a fourth of that distance from their cluster's center. With such large geographic distances, however, it is likely the case that there exists some *alternative* cluster whose center is geographically nearer to the client than the client's own cluster's center. For clarity, we will refer to a client's own cluster's center as its "default" center, and the cluster center geographically closest to the client (excluding the "default" center) as the "closest center" or "alternative center". In Figure 4.14, I compare the properties of each client's default and closest centers.

Subfigure 4.14a shows a scatter plot of the geographic distance from each client (in kilometers) to its default and closest centers. The diagonal line dividing the plot indicates where



(a) geographic distance



(b) CNRE similarity

Figure 4.14. Subfigure 4.14a shows a scatter plot of each client's geographic distance from its own ("default") cluster's center location versus its geographic distance to the geographically closest center of another cluster ("closest"). Subfigure 4.14b shows a scatter plot of each client's CNRE similarity with its own ("default") cluster's center location versus its CNRE similarity with the geographically closest center of another cluster ("closest").

the geographic distances are equal. Points beneath the line correspond to clients who are closer to their alternative centers, while clients with points above the line are closest to their default centers. Most clients are closer to their alternative centers, but there are several details to note, discussed below.

First, it is apparent that most clients are geograhpically closer to their alternative cluster centers than their default centers, in some cases by orders of magnitude. Second, I remind the reader that I employed the complete linkage method to form our hierarchical clusters. Because of the behavior of complete linkage, which determines cluster membership by pairwise distance across all members instead of individual members, it is possible that individual clients may have a higher CNRE similarity with their alternative center than with their default center. In other words, it may be the case that I have assigned some clients to the "wrong" cluster. Occurrances of this may account for some of the noise observed in default center distances below 1000 km in Figure 4.13. To test for mismatched clients, I plot a point for each client's CNRE similarity towards its closest center versus its default center in Subfigure 4.14b. The majority of CNRE values in Subfigure 4.14b are concentrated between 0.7 and 0.8. This suggests that geograhpically overlapping resource allocation groups may be responsible for the behavior observed in the ambiguous regions of Figures 4.7 and 4.8.

Although 95.18% of clients had alternative centers geographically closer than their default centers (274.69 kilometers closer in the median case), 16.48% of these geographically closer clients had higher CNRE similarity with their default centers (1.99% higher in the median case). While I have demonstrated that high geographic distance coupled with high CNRE tends to result in higher latency, here I paradoxically observe that this is arrangement is a common occurence. I further explore the implications of this pattern in Section 4.7.

4.7. Discussion

4.7.1. Why Web Object Domains

In Section 4.3.2, I described how the set of domains used in my experiment were selected. The selected domains were extracted directly from web object URLs observed across the set of checked web pages. Note that these domains are often abstractions of more explicit hosting schemes. For example, such domains may resolve to unique CNAMES or directly resolve to third party CDN addresses [**111**]. In contrast to my approach, I could have converted each domain into some lower level representation (such as its CDN) and in turn performed a CNRE-like measurement study using this representation (*i.e.*, ping from each client to each CDN in the set).

While this alternative approach may provide its own insights, I chose leave each discovered domain "as is" for two reasons. First, although a given domain may use CDN hosting, it is worth noting that modern CDN selection techniques are complex and diverse. While some content providers may opt to utilize a *single* CDN for their purposes, it has also become common practice to instead depend on CDN *brokers* or *multi* CDNs to dynamically (by cost, performance, geography, *etc.*) make use of a set of CDNs. In other words, the "less abstract" representation of a given content provider is, in many cases, far from comprehensive with regard to a diverse set of clients. Second, even if a domain or content provider could adequately be reduced to a lower level description as described, the performance of a given CDN "in general" is not necessarily representative of the performance of one of its customers. The reasons for this are plentiful, ranging from customer specific policies and agreements (for example, the customer might purchase region specific support) to caching algorithms and load balancing.

4.7.2. Anycast

I opted for this project, as the first ever exploration of aggregate content resource catchments, to focus on naturally occuring diversity in DNS redirection across domains and clients. However, many large Internet platforms, although serving from globally distributed points of presence, operate with as few as one primary public IP address for all clients, regardless of the client's location by employing *anycast* techniques as opposed to DNS redirection for server-to-client mapping [**82**]. In anycast, a provider will advertise different BGP route announcements for the same IP address in different regions. While, in its current implementation, CNRE is dependent on differing DNS answers, it is in some cases possible to distinguish differing anycast hosts independently of their IP address [**53, 68, 48**]. With this approach, it would be trivial to treat distinct anycast destinations in the same way as distinct IP answers, enabling them to be incorporated into CNRE's calculation.

4.7.3. Bettering Catchments

In Section 4.6, I exposed the existance of many clients that likely experience poor mapping between themselves and content resources: the groups with whom they share a high proportion of network resources are physically much farther away than the centers of other groups that may offer the unfortunate client better performance. In related work, researchers found that many clients experience significantly lower latency toward a specific CDN by simply slightly by posing as a member of a different subnet [**118**]. While their results are on a per-CDN scale, I propose that it may be possible to apply their techniques *across* providers to outlier clients discovered in this project. Rather than identify an optimal alternative subnet for each CDN, it may be the case that a client can choose a *single* alternative subnet from an aggregate catchment

to use for all of its DNS resolution, across domains. I leave in depth analysis of this potential avenue for future work.

4.7.4. Client Labels

A key contribution of this project is the observation that established client labeling paradigms — country, ASN, and BGP prefix — fail to describe common network resource exposure, especially at arbitrary levels of granularity. These finds may warrant the introduction of a new, heirarchical labeling model to this end. Related work demonstrated that there is much incentive to decouple catchment labels from IP prefix matching systems, but failed to offer an adequate and intuitive alternative client naming convention [**118, 45**]. I argue that this project establishes CNRE as an appropriate basis for such a new convention. Rather than arbtrarily building catchment groups and corresponding labels from scratch, CNRE highlights *existing* resource allocation trends, from which appropriate insights can be drawn.

4.8. Summary

In this chapter, I performed a large scale analysis of cross-domain DNS redirection for 9,024 globally distributed clients. My experiments spanned 302 content hosting domains, shown to often coexist within the same popular web pages. To quantify my findings, I introduced common network resource exposure, a similarity measure that captures the extent to which clients are directed to the same content resources. I validate CNRE as a necessary new measure by formally demonstrating that existing alternative labeling schemes fail to adequately capture the same information.

Having established CNRE, I investigate the properties of high and low CNRE measures between clients, and, from my findings, I derive CNRE *clusters*, representative of aggregate content resource catchments. Finally, I show that a client's geogrpahic relationship with its cluster's center is directly proportional to that client's average performance across hundreds of domains. In summary, my work in this project provides a baseline by which we can better understand content resource allocation patterns, independent of the case specific concerns of individual hosting implementations.

CHAPTER 5

de-Kodi: Understanding the Kodi Ecosystem

5.1. Introduction

It is no secret that video streaming constitutes a substantial proportion of the data moving across today's global networks. Cisco Systems predicts that video traffic will make up 82% of *all* Internet traffic by the year 2021, amounting ot 17,000 hours of video content traversing the Internet every *second* [**59**]. This raises the following question: How can one begin to digest and analyze the vast landscape of streamable content, from its many content sources to its network infrastructure and disparate stakeholders? Kodi, an extremely popular home media center platform estimated to have millions of users [**106, 103**], may hold the key.

The Kodi software platform, compatible with a wide range of consumer electronics (including Windows, Mac, Android, Linux machines), serves as a framework for tens of thousands of user-made *add-ons* which each extend Kodi's feature set. A screenshot of the home menu screen of a Kodi instance containing a number of installed add-ons is shown in Figure 5.1. Many of these add-ons serve as interfaces that conveniently connect Kodi to various third-party streaming platforms. Collectively, such add-ons effectively transform Kodi into a powerful *hub*, a "one stop shop" for all things streaming. This large ecosystem, consisting of millions users and countless user-developed add-ons, presents a uniquely wide, cross-sectional view of the modern video streaming landscape. I argue that by obtaining a snapshot of the Kodi ecosystem,

one has, in effect, obtained a sampled snapshot of online digital content distribution systems across the Internet at large.

The goal of this work is to perform an in depth analysis of Kodi's ecosystem and, in so doing, obtain a snapshot of the infrastructure behind modern, free video streaming. Very little is known regarding the nature of the thousands of third party applications, or *add-ons* at the disposal of Kodi users, although alleged illicit activity (*e.g.*, pirated video streaming and botnet proliferation) attributed to user-made Kodi add-ons has garnered notoriety for the platform [**103, 106, 64, 58**]. Accordingly, I wish to investigate the plethora of Kodi add-ons and their respective content and providers. In particular, I aim to uncover 1) where add-ons are coming from, 2) the degree to which said add-ons are maintained, 3) the extent of observeable illicit or otherwise unwanted activity supposedly coupled with Kodi add-ons and 4) the technical properties of the free content made accessible by Kodi add-ons.

To facilitate this study, I have built a powerful system, DE-KODI, capable of navigating and assessing Kodi's vast array of add-ons in an automated fashion; in other words, a tool capable of "*decoding*" the complexities of the Kodi ecosystem. DE-KODI leverages the Kodi platform itself to enable our exploration. I have designed DE-KODI's performance be both *tunable* and *scalable* so as to better support the diverse range of use cases in which it may be applied.

From this work, my key contributions are as follows:

- I introduce DE-KODI, a system designed to exhaustively traverse the Kodi ecosystem with arbitrarily set levels of depth, breath, and speed. The source code for DE-KODI has been made publically available at [**24**].
- I use DE-KODI to perform the first transparent analysis of Kodi's add-on ecosystem.
- I quantify the prevelance of apparently malicious traffic across Kodi add-ons.

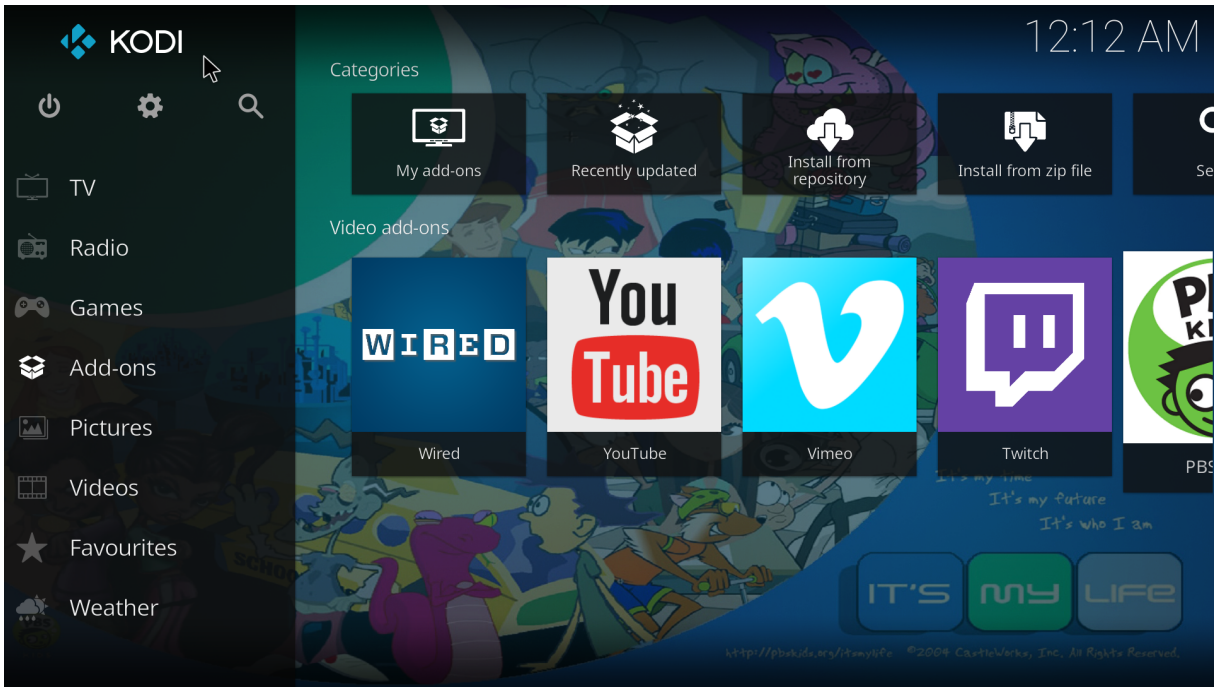


Figure 5.1. Screenshot of Kodi's home menu.

- I discover 744 distinct second-level domains (SLDs) apparently hosting or having hosted freely accessible streamed content, and I assess their network characteristics.

The remainder of this project is organized as follows: First, in Section 5.2, I provide the reader with further context regarding the Kodi platform as well as related work. I then outline the design of DE-KODI in Section 5.3. Section 5.4 validates DE-KODI's speed and scalability. I provide a brief overview of dataset obtained through DE-KODI's snapshot of Kodi in Section 5.5. Next, in Sections 5.6 & 5.7, I perform an in depth analysis of the Kodi ecosystem. Finally, I summarize this project's findings in Section 5.8.

5.2. Background & Related Work

Kodi – Among the countless media center platforms available — Plex, OSMC, Emby, MediaPortal, and MrMC, to name a few [**101, 98, 67, 89, 91**] — Kodi has begun to capture the attention of news media and groups interested in digital copyright protection. Kodi, an extremely popular open source project from which several other popular platforms forked, dates back to 2004 under its previous moniker, Xbox Media Center (XBMC). Originally designed for Xbox, Kodi is now readily available for easy installation across a wide range of consumer devices such as personal computers, Amazon Fire TV Sticks, and Raspberry Pis.

Kodi's community touts thousands of add-ons for video consumption alone, in addition to a plethora of add-ons for other forms of media, such as photos, music, and games [**119**]. Add-ons form the foundation of media consumption on Kodi. Many of the add-ons available to Kodi users were made not by official Kodi affiliates, but by third-party developers leveraging the convenience of the Kodi platform. It has been well established that a number of these third-party add-ons engage in piracy; in fact, Kodi's official wiki site bans a set of add-ons, primarily consisting of pirated content aggregators [**120**].

The Motion Picture Association of America (MPAA) claims that about 26 million (68.5%) of Kodi's alleged 38 million users are accessing pirated content through the Kodi platform. These numbers, representative of Kodi's global user-base, reflect similar findings in a previous North America specific study performed by Sandvine [**106**]. Meanwhile, neither the MPAA nor Sandvine have produced any clear information regarding how these numbers were obtained, thus leaving the legitimacy of their claims up for debate — especially when several entities, including the makers of Kodi, have stated that there exists no mechanism to acquire such measurements [**103**]. While I do not directly challenge the validity of these claims, the lack of

transparency and clear conflict of interst from such studies have served in part as motivation for this project. Kariiti et. al [**81**] observed that studies of piracy and its resulting impact are often grossly exaggerated and readily misquoted by media industry stakeholders. Rather than attempt to explicitly classify specific content as copyrighted or not, I instead, throughout this project, highlight the 1) popularity of distribution of Kodi add-ons and 2) the availability of working, streamable content. By this approach, I drastically lower (by orders of magnitude) the size of the space warranting investigation from concerned parties, who can further utilize DE-KODI for an in-depth analysis of each add-on. Most importantly, by offering DE-KODI as a freely available and open source tool, I present, for the first time, an opportunity for *transparent* and *repeatable* analysis of Kodi's ecosystem, including the wide range of third-party content sources upon which Kodi add-ons depend.

The main contribution of this paper is an in depth study of the Kodi ecosystem. To the best of my knowledge, no previous research paper has investigated this ecosystem yet. Conversely, researchers have directed their attention in understanding the potential security and privacy threats of the Kodi application [**95**] as they allow arbitrary code from unknown sources to be executed with little control. The authors show, for instance, how addons and video subtitles can be used as backdoors to gain control on the client device. In this project, I check network traffic observed during DE-KODI's crawl for suspicious activity, detailed in Section 5.7.

Copyrighted Video Distribution – More related work can be found in the area of copyrighted video distribution, a well explored topic over the last 10 years. Since my work also comments on the legality of content distributed over Kodi, I here summarize the main research papers in this area.

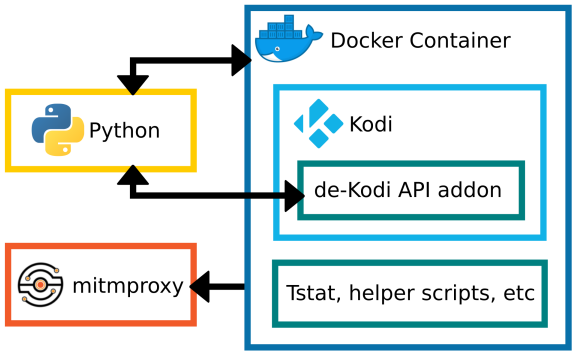
Back in 2007-2011, platforms like YouTube and Vimeo were mostly used for redistributing illegal content [**73**], [**60**]. Even when legal, the majority of the uploaded content was copied rather than user-generated [**65**]. Video platforms implemented several technical solutions to prevent copyrighted materials, which in turn triggered ingenious evasion techniques such a reversing of the video (particular used in sports), covering of TV logos, etc.

To avoid dealing with copyright detections, uploaders directed their attention to *cyberlockers*, or services offering remote file storages, sometimes even for free [**85**]. In [**84**], the authors scraped popular cyberlockers, e.g., MegaUpload and RapidShare, and show that 26-79% of the content infringed copyright. More recently, Ibosiola et al. [**78**] study *streaming cyberlockers*, or illegal websites which distribute pirated content (mostly video). The paper looks at both cyberlockers and the content they serve. Overall, it finds a centralized ecosystem composed of few countries and cyberlockers.

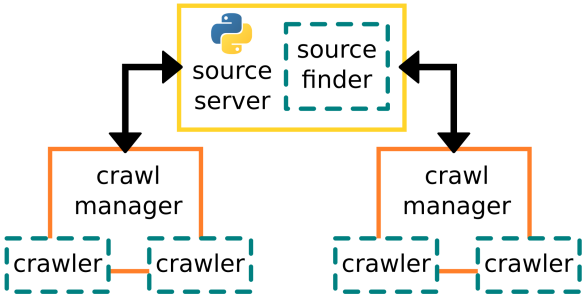
An interesting new angle was explored in [**74**]. In this paper, the authors investigate a very intuitive question: *why are illegal streaming services free?*, or what is in it for the streamers? They focus on illegal sports streaming and show a huge extent of user tracking – much more than what done in legitimate streaming services. I investigate Kodi network chatter for signs of tracking in Section 5.7.

5.3. de-Kodi System Overview

This section details the DE-KODI system I have developed to explore the Kodi ecosystem. I first discuss the key challenges in building DE-KODI. I then present the detail of DE-KODI's key components, namely the *crawler* and *source finder*. Finally, I describe the overall working flow of DE-KODI.



(a) a single crawler



(b) crawling system

Figure 5.2. A visual overview of the DE-KODI system. Figure 5.2a shows the structure of an individual *crawler*. The *crawlers* in 5.2b are instances of the *crawler* shown in 5.2a, but in the case of 5.2b, I use one instance of *mitmproxy* per machine to capture traffic from all *crawlers*.

5.3.1. Challenges

Platform level software changes: The official Kodi team actively updates and improves the platform's main software base; Kodi 18.0, the most recent major revision as of this writing, was released on Jan 29, 2019. As the platform's base software changes, ill-maintained add-ons inevitably fall into obsolescence. Well-maintained add-ons release newer, versions compatible with the latest versions of its dependencies and the Kodi platform itself, old, deprecated versions of Kodi add-ons persist on the web. As a result, identifying currently functional Kodi add-ons

compatible with the latest version of Kodi is nontrivial and often requires some from of "duck testing".

Visual dependent interaction: Although Kodi offers a built-in API for generalized operations (such as moving up and down on a menu), much of the relevant information one may be inclined to use in a crawling procedure — such as the text displayed in a non-default menu or dialog — is completely inaccessible. As a result, any automation I attempt must be able to function with an extremely limited and fragmented view of the system state.

Decentralized add-ons: My primary interests lie not in Kodi's own software, but in the wide array of community developed *addons* at Kodi's disposal, as well as the disparate content resources they pull from. Although there are many community maintained *repositories* that aggregate and distribute popular add-on collections, there is no single "app-store-like" database from which one can reliably obtain a comprehensive list of all Kodi add-ons. Therefore, the size of Kodi's overall ecosystem is unknown, and any attempt to explore it must take into account the potentially large size of the space.

Diverse developer community: Kodi's international fame has drawn an equally diverse add-on developer community. Kodi does offer some conventions for the structure of each type of add-on, but adherence to these convensions are completely arbitrary. Further, different add-ons serve varied purposes and often require distinct methods of interaction. Paired with the above issue of visual dependent interaction, it is important that my approach is both robust and resilient to getting "stuck".

Malicious add-ons: Previous work and news media have well established the (realized) potential for Kodi add-ons to carry dangerous malware. Out of respect and concern for both lab

equipment and the instituion at which my experiments were deployed, I needed to ensure any potential threats were sufficiently *isolated* from institutional and lab resources resided outside of the scope of my work.

5.3.2. The DE-KODI Crawler

The *crawler* component of DE-KODI serves three purposes: 1) installing add-ons and add-on sources, 2) browsing add-ons and add-on sources for new add-ons or playable media, and 3) isolating and documenting the behavior and effects of each individual add-on. A high level view of the crawler's design is shown in Figure 5.2a. A Python script, which contains most of the crawler's logic and state, launches a docker [**23**] container derived from an Ubuntu image. I primed the image to include Kodi's software platform with DE-KODI, which greatly extends the functionality of Kodi's API, add-on pre-enabled. Tstat [**37**] and additional helper scripts mounted to the container help capture network statistics and system state changes. Much of Kodi's traffic is encrypted, so I also directed the container's HTTPS traffic through Mitmproxy [**32**] (a "man in the middle" proxy server) to expose the contents of such messages. Each crawler instance runs Kodi headlessly via a virtual screen (Xvfb) [**38**]. The main python script communicates directly with DE-KODI's addon script via RPCs.

As mentioned above, many significant aspects of Kodi's state cannot be explicitly obtained programmatically. This renders DE-KODI's crawler blind in scenarios where an add-on interaction requirements deviate from the most basic conventions. I ultimately settle for a brute force approach, which I found works well for most add-ons that do no require explicit human intervention (*e.g.*, no required log-in or pay wall). At a high level, I treat each add-on's respective menu screens as trees, where non-leaf buttons are assumed to lead to towards child branches

(other screens or menus) and streamable content may reside on the leaf-node buttons. In cases where Kodi's built-in API falls short of my needs, the DE-KODI crawler attempts to traverse the tree via a random walk, using one of many case specific implementations designed to avoid infinitely deep branches and other progress inhibiting phenomena commonly encountered in browsing Kodi add-ons.

5.3.3. The de-Kodi Source Finder

A key requirement for these experiments is the ability to perform add-on discovery. While there exists no single comprehensive list of add-ons for the Kodi platform, I was able to leverage two properties of the Kodi ecosystem to facilitate an expansive crawl.

First, I draw the reader's attention to what the Kodi community calls *sources*. Sources are simply paths — local or remote — that point to files to be used by Kodi. By default, only local sources are listed within a fresh instance of Kodi. However, the Kodi community often deploys remote, third-party sources which host zip files of add-ons ready to be installed. By convention, many of these source derived add-ons are themselves *repositories*, which aggregate and ease the installation of some set of add-ons. Sources are remotely maintained and updated by their owners. DE-KODI's *source finder*, which itself contains an instance of a DE-KODI crawler, can add known sources to Kodi and traverse their sub-directories for add-ons in the form of zip files.

Second, I take advantage of Kodi's popularity. Due to the lack of a single centralized aggregator, avid Kodi users are forced to socialize to exchange add-ons and sources. This narrows the scope of my search: rather than scavenge the Internet extensively, I enable the DE-KODI source finder to browse popular social platforms for potential Kodi add-ons.

In these experiments, I ultimately use three such pointers, which I will from here on refer to as "search seeds", as starting points for add-on discovery: 1) LazyKodi, a wellknown and actively maintained Kodi add-on source which aggregates collections of add-on repositories and add-ons into a convenient, single location [**30**], 2) Reddit, a large online social platform with many publically accessible communities [**35**], and 3) GitHub, a large online software development platform often used for hosting, maintaining, and distributing open source code [**28**].

I further verify the content collected by DE-KODI source finder by checking if it is in fact a Kodi add-on. When possible, I check the contents of a given web obect for mandatory Kodi add-on files and formatting, e.g., an `addon.xml` file in advance of attempting to install an add-on. It is also worth noting that crawling *redundant* copies of an add-on is a common and difficult to avoid occurence. Due to Kodi's design, add-on repositories each contain their own instances of their constituent add-on, and therefore popular add-ons are encountered quite frequently. On top of this, outmoded versions of many add-ons are often frequently encountered in a crawl. I mitigate the impact of redundancy by hashing unique add-on properties and only proceeding with add-ons whose hash I have not yet seen. Different versions of the same add-on hash to different values and therefore do not interfere with each other's crawls.

5.3.4. The de-Kodi System

In this subsection, I document the relationships between the aforementioned components of DE-KODI and describe DE-KODI's overarching control flow and structure, which depicted in Figure 5.2b. First, I start a *source server*, which contains a list of known add-on sources discovered via the DE-KODI's search seeds. The source server can actively attempt to discover new sources via its included *source finder* if this feature is enabled. Adjacent to this, I start some number

of *crawl managers*. The source server serves as a centralized point of contact for all crawl managers, who periodically query the source server for relevant state information, such as the status of a given add-on as previously hashed or new. The following precedure then occurs repeatedly indefinitely:

(1) A manager queries the source server, which replies with an item (either a source path or a link to an add-on zip file) to download and test.

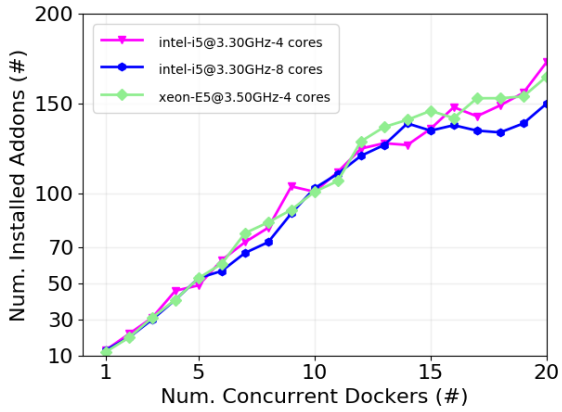
(2) The manager then attempts to install the item using a single instance of a crawler. If the installation is unsuccessful, it returns to the previous step.

(3) If the installation was successful, the manager checks to see if any previously unhashed add-ons have been installed or appeared within Kodi as options *to be* installed. If so, it continues to the next step. Otherwise, it returns to the first step.

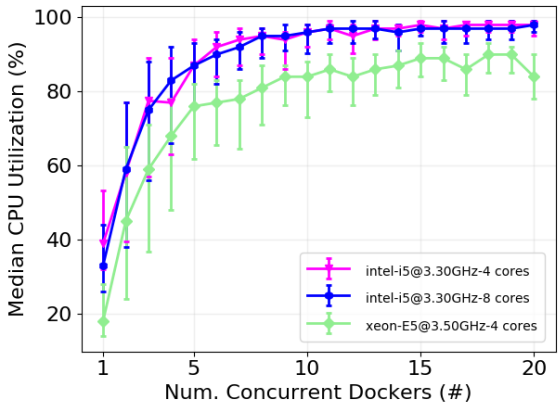
(4) Since new add-ons have been discovered as a result of the item's installation, the manager makes a new Docker image from the container of the crawler that successfully installed the item.

(5) The manager spins off new crawlers, using the newly created image, to test and crawl each newly discovered add-on (one per crawler). To more quickly traverse the set of new add-ons, managers may run multiple crawlers in parallel. Note that in some cases, an installed add-on may itself be a repository, pointing to many *other* new add-ons to test. In this scenario, new, previously unhashed add-ons are queued to be tested along with the set currently under examination in this step.

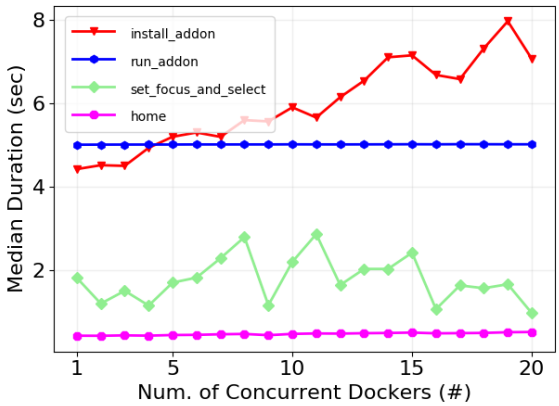
(6) Once the manager has completed its exploration of the given item, it returns to step 1, repeating the cycle on the next time provided.



(a) Number of installed addons.



(b) CPU utilization.



(c) DE-KODI primitives benchmarking.

Figure 5.3. DE-KODI benchmarking ; $N_{docker} = [1 : 20]$; Crawling-duration: 30 minutes.

Because the distribution of items to crawl is handled entirely by the source server, I am able to run an arbitrary number of crawl managers in parallel, thus covering the set of known items more quickly.

5.4. deKodi Benchmarking

One of the main goals of this project is for DE-KODI to be sufficiently lightweight for use on commodity hardware and readily scalable for arbitrarily large snapshots. To this end, DE-KODI was designed to be easily parallelizable, both in term of docker instances and number of machines where it can run. I setup three machines[1] at a university campus connected to the Internet via a shared Gigabit connection (both in download and upload). Next, I instrument each machine to run DE-KODI for 30 minutes while crawling the same set of addons. Note that in a real crawl, each machine would focus on a different set of addons, but the goal here is to compare their performance while working on the same set of addons. I repeat each crawl 20 times while increasing the number of docker instances (N_{docker}) used per machine from 1 to 20. Kodi's default addon repo was used for this benchmark.

Figure 5.3a shows the number of successfully crawled Kodi addons as a function of the number of docker instances used and the machine where the crawler ran. When $N_{docker} \leq 10$, the number of crawled addons grows linearly (between 10 and 100 addons) and no major difference is observable across machines. When $N_{docker} > 10$, I start observing a sublinear growth in the number of crawled addons and more "noise" in the results. This suggests that, eventually, the overhead of running more docker instances on a single machine does not pay off in term of crawling "speed".

To further understand the previous result, I investigate the CPU utilization during the above benchmarking. Figure 5.3b shows the median CPU utilization as a function of the number of docker instances and machine used. Error-bars relate to 25th and 75th percentiles. Note

[1]Two machines mount an Intel i5-4590 (3.30GHz, quadcore); one machine mounts an Intel Xeon E5-1620 (3.50GHz, quadcore). All machines are equipped with 8GB of RAM.

that the CPU utilization is indeed a distribution since I sample it every 5 seconds during the benchmarking. The trend mimics the one observed above, *i.e.,* linear increase followed by a saturation as Iapproach exhaustion of available CPU. It can be observed how the distance between percentiles becomes more tight as N_{docker} increases. This implies that the machines are under higher CPU utilization for a longer time as the overall load increases (higher N_{docker}). The figure shows an overall lower CPU utilization on the (slightly) more powerful machine (xeon-e5) which saturates at 80% versus 90-95% for the other machines.

To understand the latter results, I set out to benchmark low level DE-KODI "primitives", *i.e.,* functions like "install_addon" or "run_addon" which are composed together to enable crawling. Figure 5.3c shows the average duration of key DE-KODI primitives as a function of N_{docker}. These results refer to one of the machines, but they are representative of all machines. The figure shows how most DE-KODI primitives are not impacted by N_{docker}, *i.e.,* their durations are limited by Kodi's internal rather than the machine resources. The primitive "install_addon" is the only one impacted by the machine resources. This happens because this primitive is more complex and requires network operations (to pull the addon), and CPU usage (to perform its installation). However, this operation only constitutes a small fraction of DE-KODI operations which are instead dominated by constant operations.

No significant difference was instead reported in term of memory consumption. Across the machines, DE-KODI requires a minimum of 500MB ($N_{docker} = 1$) and a maximum of 4GB ($N_{docker} = 20$). Based on these empirical results, I set for the crawler a conservative $N_{docker} = 8$ which should allow me to crawl, in total, about 11,000 addons per day while not overloading the test machines.

Total links tested	2,662
Fruitful links discovered	1,151
Total repositories discovered	653
Total add-ons discovered	7,362
Banned add-ons discovered	421
Add-ons with media discovered	662
Total media URLs discovered	15,167
Total media domains discovered	1,600
Total media second-level domains discovered	744
Total add-on providers discovered	2,885

Table 5.1. Crawl summary

5.5. Dataset Analysis

In this section I briefly discuss the execution of the experiment and the nature of the dataset. A synopsis of this discussion can be found in Table 5.1.

5.5.1. Data Collection

I ran DE-KODI across the three machines used for benchmarking in the previous section, enabling each machine to deploy up to eight docker instances in parallel. I then crawl Kodi over one week in May 2019.

In addition to Kodi's official default repository, the crawl used three addon seeds as starting points: LaziKodi (a 3rd-party Kodi source), which returned 809 potential links; Reddit (a social platform), which returned 1,250 potential links; and GitHub (a software development platform), which returned 603 potential links. After link verification (see Section 5.3.3), 1,151 of the initially discovered 2,662 links were determined to point to real Kodi add-on zips. Note that this does not mean that the discovered add-ons were also working or able to be installed successfully, but merely that Kodi was able to identify the obtained content as an add-on, and present us with the opportunity to attempt installation.

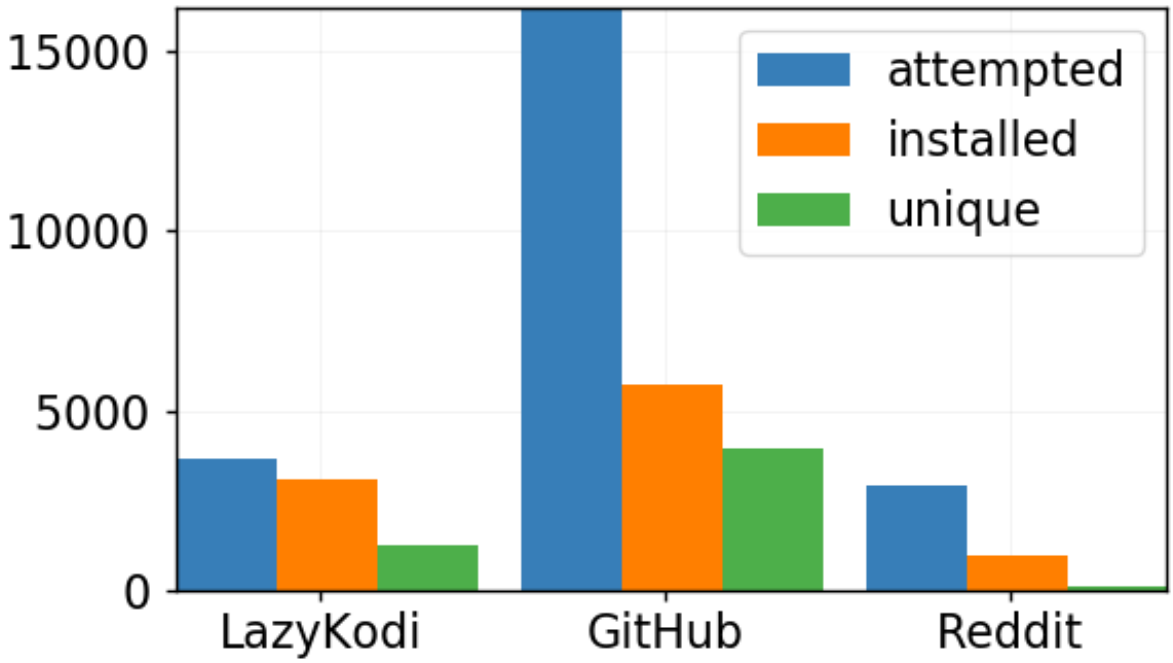


Figure 5.4. Comparison of the fruitfulness of search seeds used in this experiment.

5.5.2. Add-on Discovery Success Rate

The results of this experiment hinge on the breadth and quality of my search for Kodi add-ons. As there is no "ground truth" against which one can measure the comprehensiveness of the data, I pause here to analyze the distribution of Kodi add-ons across the three search seeds and consider its implications with regard to my findings.

Figure 5.4 shows the number of add-ons DE-KODI *attempted* to install, the number of add-ons successfully *installed*, and the *unique* add-on discoveries (relative to the other search seeds), for each seed (LazyKodi, GitHub, and Reddit). First, I observe that Reddit, although initially having provided the largest number of potential links, yielded limited results in comparison to the other search seeds. This is likely because the Reddit search seed used is inherently more open-ended than the other two seeds. LazyKodi specifically aggregates Kodi add-ons

for easy installation; it is therefore unlikely to yield false positives outside of ill-maintained or otherwise dysfunctional add-ons. Likewise, for GitHub, I am able to specifically search for GitHub repositories (not to be confused with Kodi repositories) containering Kodi-specific files and keywords necessarily present in *all* Kodi add-ons. In contrast, links acquired via Reddit provide no information regarding the contents of a linked zip file (unless they point to files hosted on GitHub), in which case the relevant traits can be tested for. Next, I note that despite the large number of Kodi add-ons identified via GitHub, very few of them successfully installed. This stems at least partially from *versioning*. By inspection, it became clear that many GitHub located repositories archived entire many past, outmoded versions of each add-on.

Finally, I draw attention to Reddit's low number of *unique* add-ons — that is, add-ons discovered via the Reddit search seed that did not also appear on the GitHub or LazyKodi results. Although 961 Reddit-discovered add-ons were successfully installed, only 116 add-ons of that installed set were unique to Reddit. This highlights the practicality of the dataset. As documented in Table 5.1, the global space of Kodi add-ons is potentially very large or incalculable. However, I can leverage the fact that the number of users of an add-on is indicative of the add-on's popularity. It is well established that popularity follows a power law distribution — for example, although there exist well over 1 billion websites, the top 0.1% dominates most of the traffic. Therefore, I assert that the most accessible and shared add-ons (*i.e.*, those that propagate social media) are the most descriptive of the Kodi ecosystem at large. As the GitHub and LazyKodi search seed results span the majority of what I uncovered via Reddit, I argue that this dataset is sufficiently expansive for the purposes of this paper.

5.5.3. Banned Add-ons

Although commonly associated with piracy by the public, the official makers of Kodi (the Kodi Team), actively pushes against the use of their software for flagrantly illicit activities, such as piracy. Kodi's official community forum has formally banned any promotion or discussion of any add-on, add-on repository, or Kodi version included in their manually maintained blacklist [**120**]. In the crawl, I discover 421 add-ons associated with entries from this list.

5.5.4. Other Content

The Kodi platform, initially developed to extend the functionality of the original Xbox game console, is capable of more than only streaming. Kodi has long supported a wide variety of other media types, including local video and audio, as well as photos, games, radio, and IPTV. Moreover, even for streaming, a number of Kodi add-ons involve complex additional requirements. For example, the content of some legitimate services — such as Netflix — available in the Kodi ecosystem are only accessible paid accounts. By inspection, I also discovered add-ons which require accounts with special third-party intermediary services known as "multihosters" or "debrids" [**27, 34, 33**] in order to access the add-on's content. While *in depth* explanation of such add-ons lie outside of the scope of this project, I have made DE-KODI available as an open-source project so that its features can be extended to address cases such as these, as more, in future work. Note however that, although contained content is not necessarily directly identified or accessible for some add-ons in this space, the add-ons themselves are still captured and analyzed accordingly in the crawl results.

5.6. Kodi Ecosystem Analysis

in this section we perform a detailed analysis of the Kodi ecosystem, using the data discussed in Section 5.5

5.6.1. Add-on Repositories

Although Kodi lacks a strong central resource from which to pull add-ons, Kodi add-ons are not only distributed in a fully ad-hoc fashion. Rather, Kodi encourages users to group add-ons into "repositories", special add-ons which themselves facilitate the easy installation of a predetermined set of *other* add-ons. This has evolved into a tree-like network of add-on relationships, a significant factor in enabling us to cover such broad space with so few seeds.

In Figure 5.5, I plot the CDF of the number of add-ons provided by each repository. The vast majority of add-on repositories — many of which are decendents of the distribution "trees" rooted at the largest repositories — offer less than 100 add-ons each; the median case provides less than 10. Further, the majority of add-ons provided by smaller repositories (below the 50^{th}) are of the Kodi-defined type *pluginsource*, which indicates they are designed to provide the user with some form of consumable media.

5.6.2. Add-on Providers and Origins

The Kodi platform strongly encourages community driven add-on development. The Kodi Team, the group responsible for the development of the Kodi platform itself as well as its "official" add-ons set, consists entirely of volunteers, takes credit for 216 Kodi add-ons [**36**]. In general, it is unclear how many developers are behind any given "provider" title. I observed 2885 different add-on providers (including the Kodi Team) in our crawl. Over 90% of the

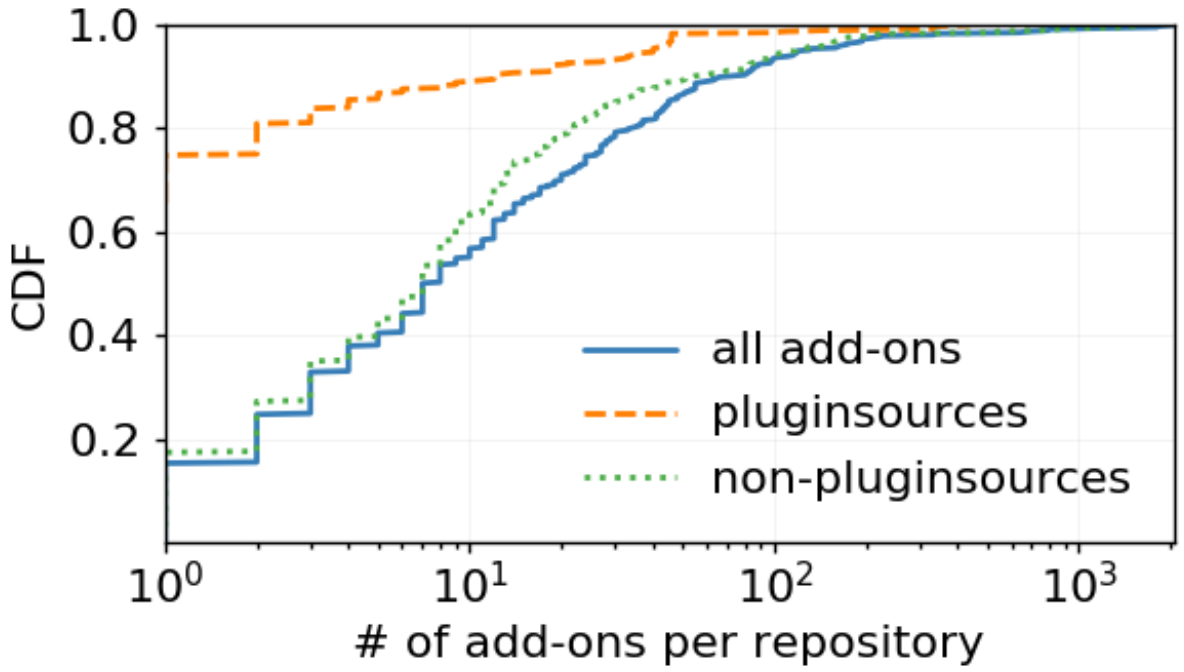


Figure 5.5. CDF of add-ons per repository.

providers encountered were each responsible for only a single add-on in our dataset. It is worth noting that completion of add-on metadata, such as the provider-name field, is arbitrary and unregulated.

Besides the authors of add-ons, I also investigate the target *audience* of Kodi add-ons. I acheive this in part by examining the content hosting strategies of Kodi add-ons. Note that, in addition to existing limitations of IP geolocation [**121**], our results here may be skewed to North America, where our hostname resolutions were performed from. While some add-on content is hosted by globally distributed services (*e.g., content distribution networks (CDNs)*). Figure 5.6 shows a bar plot of the number of hosts discovered in the top Kodi content hosting countries.

Here, I estimate the popularity of each add-on by observing the frequency with which the add-on appears across the dataset resulting from DE-KODI's crawl. I remind the reader that

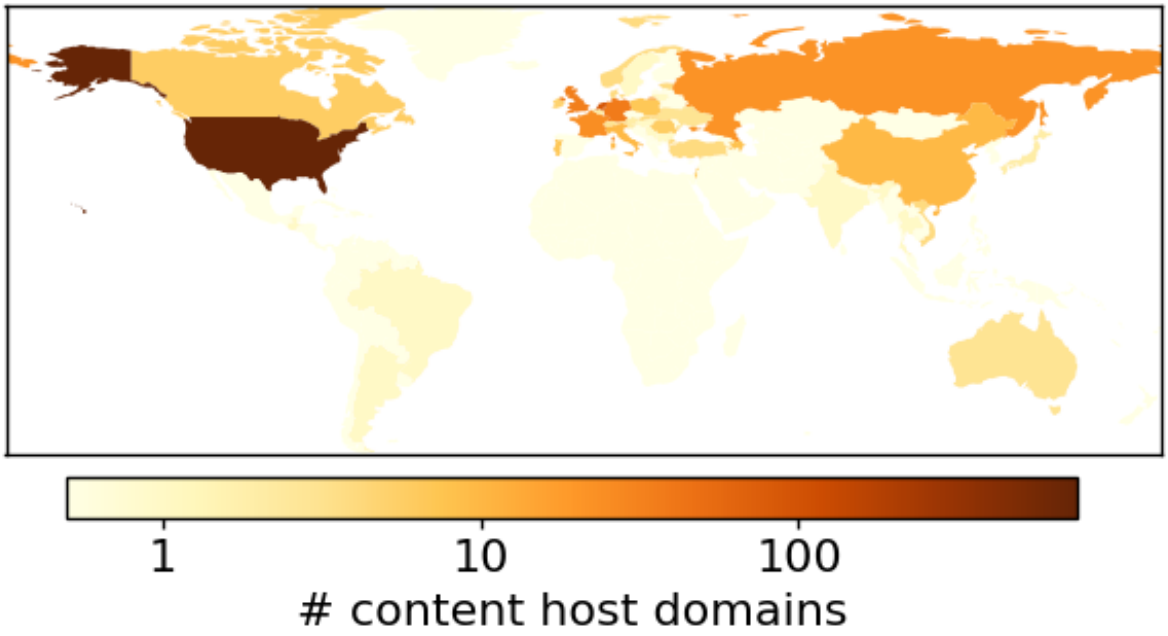


Figure 5.6. World map of Kodi content host locations. Country color shading indicates the number of content hosting domains discovered there in our experiment.

Kodi repositories contain bundled sets of add-ons for convenient installation, and that the Kodi Team actively encourages users to create and share repositories on their own. Also note that an officially endorsed repository, curated and maintained by the Kodi Team, is included with fresh installations of Kodi. Figure 5.7 shows the number of distinct repositories that included each add-on. In the figure, I separate so-called "banned" add-ons (defined in Section 5.5.3) from the set of all other add-ons discovered during this crawl. While a low number of repositories including a given add-on may not necessarily imply low popularity, it stands to reason that an add-on referenced by many repositories must necessarily be popular, as more users must have arbitrarily opted to include the add-on in their custom repository.

From Figure 5.7, it is clear that the majority of add-ons discovered in this crawl appear in very few repositories — only two repositories in the median case. When restricting our scope to only banned add-ons, this trend becomes omewhat more pronounced, suggesting that illicit add-ons may not necessarily cominate the Kodi community as much been suggested by Kodi's opponents. It is also worth noting that the pattern seen in Figure 5.7 reflects that observed in Figure 5.6, where most repositories discovered contain less than 10 add-ons. Moreover, we see that the CDFs of both plots are *long tailed*, with data points trailing into values over an order of magnitude larger than that of the majority. These trends are similar to what has been established in literature as "popularity laws", where attention is exponentially more concentrated towards the most popular of a given group [**96**]. With this assumption, in combination with Figures 5.6 & 5.7, we can draw the following insights. First, it is apparent that banned add-ons are approximately *as* popular as the general population of Kodi add-ons we discoved, as the curves of their distribution across repositories are nearly identical. Second, I propose that such patterns in popularity conveniently narrow the required scope for expansive measurements of this nature. What are likely the most popular repositories — and in turn, the easiest to find — contain the *most* add-ons. It would appear that in the absence of an official and exhaustive "app-store" for the Kodi ecosystem, something similar naturally manifests in its stead: large, well known repositories that contain the majority of what a given Kodi user could want. Meanwhile, although the least popular add-ons may be more difficult to find, their lack of popularity likewise renders them less significant: their footprint is inherently orders of magnitude smaller than that of that have attained popularity.

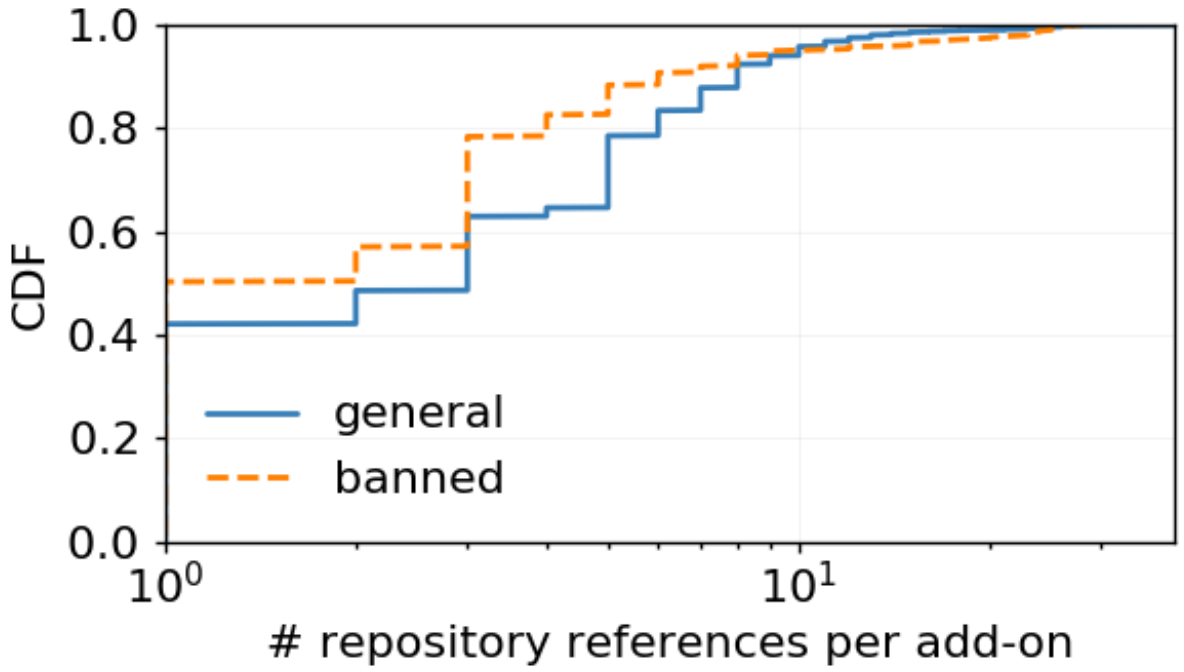


Figure 5.7. CDF of the number of found repositories that included a discovered add-on.

5.6.3. Background Network Chatter

In general, browsable Kodi add-ons — a group which consists primarily of add-ons typed "pluginsource" — perform a considerable amount of network chatter in the background as the user browses the application. This can occur for a variety of reasons, ranging from remotely loading stored resources such as video thumbnails and live content lists to performing user tracking and malicious activities. I consider the latter in detail in Section 5.7. In Figure 5.8, I plot the amount of network chatter per browseable add-on. To facilitate direct comparison between add-ons, I normalize byte counts to bytes per minute of the add-on's crawl sessions. Note that the resulting values also include bytes exchanged to initially download the add-on. Banned add-ons, which Figure 5.8 plots separately, exibits overall similar network performance to unbanned add-ons.

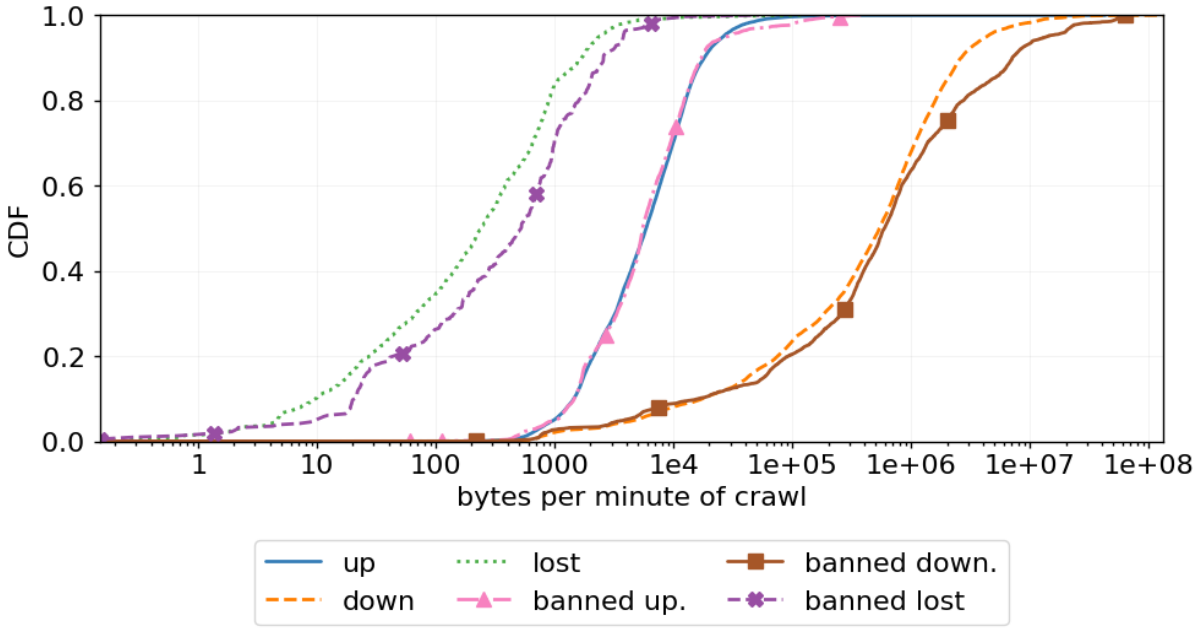


Figure 5.8. CDF of mean bytes per minute for each add-on's crawl session.

Figure 5.9 shows the top most frequently used port numbers observed in our experiment. In this case, we see a notable distinction between banned and unbanned add-ons. While unbanned add-ons tend towards well known ports, such as ports 80 and 443, banned add-ons also make heavy use of lesser known ports, especially those associated with torrenting.

5.7. Suspicious Activity

Despite the best interests of the official Kodi team, the inherently open nature of the Kodi platform ultimately renders it ever susceptible to unintended uses, spanning from piracy to malware. In this section, I quantify the extent to which I discovered suspcious add-on behavior over the course of the experiment.

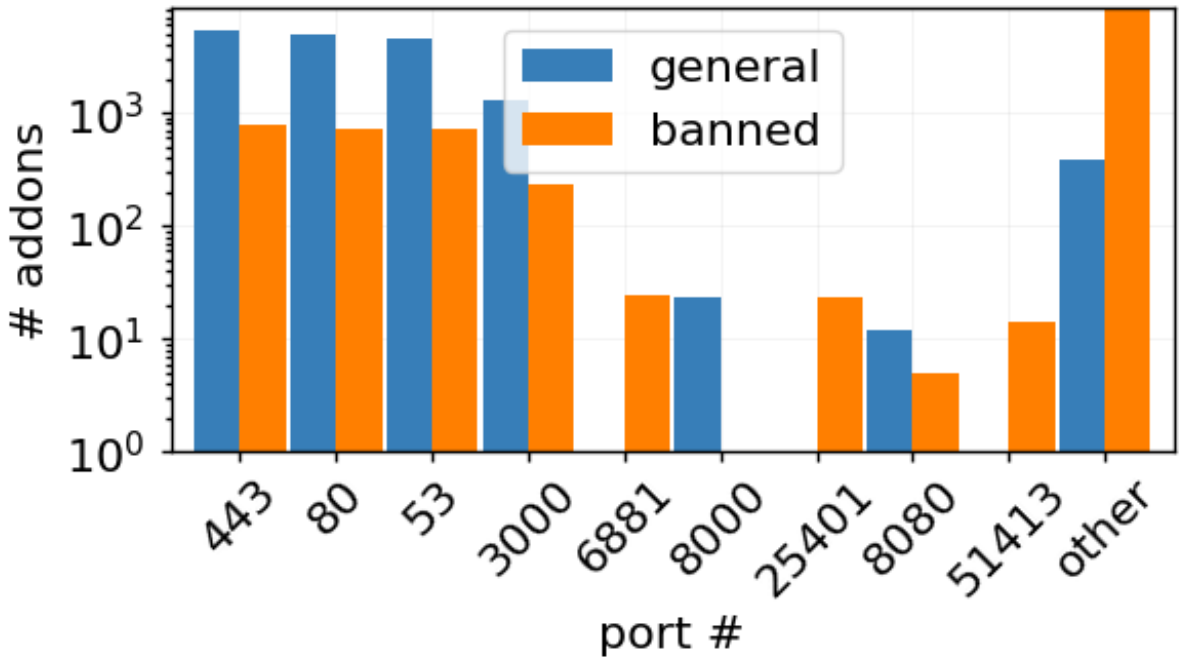


Figure 5.9. Bar plot of the number of add-ons using each port number used for background downloads (stack it to show TCP/UDP). The final bar (labeled "other") is cumulative across all port numbers not shown (*i.e.*, some add-ons may be counted multiple times for "other" — once for each port number).

5.7.1. Banned Add-ons and Piracy

Here I consider the list of add-ons banned from discussion on Kodi's official forums and the implications of my findings with regard to them. Although add-ons found to violate any of Kodi's official policies are banned from their forums, the also provided general rule of thumb in determining whether a currently unlisted add-on is "allowed" points users to specifically avoid piracy. Likewise, per their language, it is safe to assume that the majority of add-ons included on Kodi's banned list is or was at some point engaging in flagrant piracy.

Explicit copyright infringment analysis is beyond the scope of this paper. However, some of my findings indirectly suggest — but do *not* confirm — that banned add-ons in the dataset

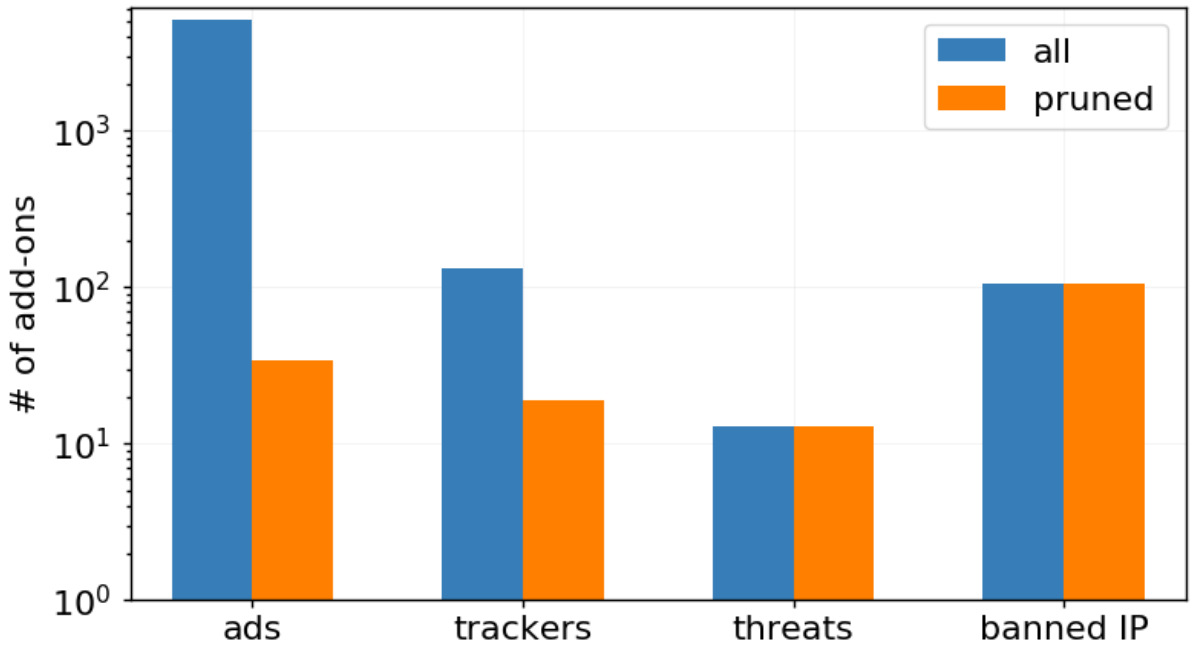


Figure 5.10. Bar plot of the number of add-ons found to exchange forms of potentially undesirable traffic.

are likely designed to engage in piracy. Most notably, we return our attention to Figure 5.9 in Section 5.6.3. Banned add-ons were found to more heavily use lesser known ports often associated with torrenting clients. While not inherently indicative of illicit activity, it is well known that many torrenting communities have long served as havens for content piracy [**109**].

5.7.2. User Tracking and Ad Chatter

Online user tracking, due to its potential to erode privacy, has drawn increased public concern and sparked new protective legislation [**61**]. In addition, authors of [**74**] have shown that free pirated content accessed via streaming has served as a trojan horse for excessive user tracking and advertisement. Therefore, although not necessarily indicative of illicit activity, background

tracking and ad traffic warrant investigation. Here I investigate the amount of user tracking and ad traffic background connections and streamable content locations across the dataset.

In order to identify suspicious traffic, I compare all full URLs against well-maintained URL blacklists. From EasyList [**25**], a group that aims to minimize unwanted online advertisement, I use the primary EasyList, which matches advertisement related URLs, and EasyPrivacy, which matches known tracking URLs. By routing port 80 and 443 traffic through the man in the middle proxy, I am able to use full URLs — including, for example, parameters — for these checks. Figure 5.10 plots the number of distinct advertisement and tracking URLs seen by de-Kodi for add-on in the dataset. Note that throughout this analysis, results shown count distinct domains as opposed to specific URLs including path information. However, full URLs were used against all employed filters.

As seen in Figure 5.10, the number of advertisement related traffic initiall obsserved was suspiciously high — approaching the number of add-ons crawled. Manual inspection revealed that EasyList's blacklisting rules filter aggresively and included what could debatably be called false positives. Most notably, all forms of "www.google.com/adsense/*", "www.google-analytics.com/*", and "https://raw.githubusercontent.com/mash2k3/MashupArtwork/master/skins/vector/adv2.png" (where "*" is a wildcard) were all marked as sources of advertisement or tracking. In light of this, I also plot the number of add-ons containing links from the remaining set if the listed questionable URLs are removed. I reserve a description of the *threats* bars, as well as final commentary with regard to this figure, for the following subsection.

5.7.3. Social Engineering

Several Kodi add-ons have garnered media attention for hosting malicious software, such as botnets and background cryptominers. I attempted to identify such behavior across the dataset by testing observed full URLs against the latest Google Safe Browsing hash, which matches against current known threats and malware [**29**]. These are plotted in Figure 5.10 as "threats". Note that Google's Safe Browsing hash targets malicious URLs generally encountered via web browsing and may not necessarily address threats that operate outside of that space, such as botnets. To increase coverage, I also compare each observed IP against FireHOL, an automatically updated aggregator of several actively maintained IP banlists [**26**], labled in Figure 5.10 as "banned IP". Detailed results for all information shown in Figure 5.10 are presented in Table 5.2.

While an in depth analysis of the uncovered suspicious links is beyond the scope of this paper, here I provide a cursory overview of the properties that stand out. Most notably, all "threats" observed in this data were marked by Google Safe Browsing as social engineering. I also highly that, surprisingly, the amount of advertisement present (having pruned out likely false positives) is much lower that other identified sources of suspicion — both in its domain diversity and its spread across add-ons. Finally, I note that the banned IPs warrant further investigation, as FireHOL [**26**] spans a number of different IP ban lists, each addressing a different type of known threat.

5.8. Summary

A large, active user-base and extensive add-on ecosystem uniquely positions Kodi, an immensely popular home media center, as a "window" through which we can capture an expanive

Total suspcious domains	182
Total ad-flagged add-ons	5186
Pruned ad-flagged add-ons	34
Total tracking-flagged add-ons	132
Pruned tracking-flagged add-ons	19
Total threat-flagged add-ons	13
Total banned IP-flagged add-ons	105
Total ad-flagged domains	18
Total threat-flagged domains	13
Total tracking-flagged domains	25
Total banned IP-flagged domains	126

Table 5.2. Suspician summary

view of disparate content distribution systems across the Internet. In this chapter, I introduced DE-KODI, a system designed to enable transparent and repeatable exploration of the Kodi. I designed DE-KODI to be lightweight and readily scalable to meet the varied needs of those interested in exploring Kodi, and, having empirically validated these claims, I made my code available to the public.

Using DE-KODI, I perform the first large scale crawl of the Kodi ecosystem. From my results, I determine the network characteristics of content streaming resources currently supported by Kodi add-ons. By using a man-in-the-middle proxy, my analysis demystifies even Kodi's *encrypted* traffic. I further examine add-on traffic for signs of malicious acivity, where I find urls currently flagged for social engineering, user-tracking, and advertisement. In parallel to this, I assess the popularity, distribution, and behavior of add-ons flagged for actively engaging in illicit activity — chiefly, piracy — in comparison add-ons that have not been flagged. Over the course of DE-KODI's crawl, I discover 744 distinct second-level domains hosting freely streamable content accessible via the Kodi platform, demonstrating the breadth of Internet snapshots captured using Kodi's field of view.

A key takeaway from this chapter is the practical tractability of arbitrarily large and complex content ecosystems. While it is infeasible to declare that one has *comprehensively* covered such a space, I show that the most *relevant* portions of that space (*e.g.*, the most popular) are easily discovered.

CHAPTER 6

Conclusion

The Internet is big. Although the allure of cross-sectional measurements is tainted by load and scaling concerns, I demonstrate that such client-based, breadth-intensive measurements are not only feasible, but necessary. Through the work I have presented, I have shown that clients offer opportunities for arbitrarily fine grained and easily scaled network measurements. I firmly establish that expansive client perspectives expose relative network properties that otherwise go unseen in conventional measurement techniques.

First, with Drongo, I showed the existence of suboptimal server-client mappings by enablings clients to test performance across multiple, ordinarily unexposed content replica servers. I further demonstrated that a poorly mapped client can, on its own, correct for these poor-mapping scenarios in a manner that still allows CDNs to make dynamic allocation decisions.

In the next section, I introduced a measure for common network exposure between clients. Measured across multiple domains, this previously completely unexposed dimension revealed cross-provider, global resource allocation patterns that ultimately play a significant role in client network performance.

Finally, I presented DE-KODI, designed to capture cross-sectional snap-shots of streamable content distribution infrastructure from the perspective of a popular home media center. I showed that DE-KODI is capable of spanning the breadth of a large content platform (Kodi)

in an automated fashion, covering the most popular and relevant portions of the space. I further analyzed the Kodi's network characteristics and quantified the prevalence of suspicious and illicit network activity present across portions of the Kodi ecosystem.

The above projects serve to strengthen the thesis presented at the beginning of this document. I have shown that client-anchored, expansive network measurements are both *feasible* and *necessary* for fully understanding the complexities of content distribution in today's Internet.

Bibliography

[1] The history of content delivery networks (cdn), Dec. 2012. https://www.globaldots.com/the-history-of-content-delivery-networks-cdn/.

[2] Akamai, 2016. http://www.akamai.com/.

[3] Alexa top sites, 2016. https://aws.amazon.com/alexa-top-sites/.

[4] Alibaba cloud cdn, 2016. https://intl.aliyun.com/product/cdn.

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

[6] Amazon Route 53, 2016. https://aws.amazon.com/route53/.

[7] Aws global infrastructure, 2016. https://aws.amazon.com/about-aws/global-infrastructure/.

[8] The cost of latency, 2016. http://perspectives.mvdirona.com/2009/10/the-cost-of-latency/.

[9] GoDaddy: Premium DNS, 2016. https://www.godaddy.com/domains/dns-hosting.aspx.

[10] Google CDN Platform, 2016. https://cloud.google.com/cdn/docs/.

[11] Google Cloud Platform: Cloud DNS, 2016. https://cloud.google.com/dns/.

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

[13] Netdirekt: Content Hosting - CDN, 2016. http://www.netdirekt.com.tr/cdn-large.html.

[14] Neustar DNS Services, 2016. https://www.neustar.biz/services/dns-services.

[15] OpenDNS, 2016. https://www.opendns.com/.

[16] Ripe atlas - ripe network coordination centre, 2016. https://atlas.ripe.net/.

[17] 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/.

[18] Sweden's Greta wants to disrupt the multi-billion CDN market, 2016. https://techcrunch.com/2016/08/30/greta/.

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

[20] Verizon Digital Media Services, 2016. https://www.verizondigitalmedia.com/.

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

[22] 2016 Q2 Mobile Insights Report, 2016 (accessed May 2019). https://resources.mobify.com/2016-Q2-mobile-insights-benchmark-report.html.

[23] Docker, 2018. https://www.docker.com/.

[24] de-Kodi Crawler, 2019. https://github.com/mwarrior92/dekodicrawler.

[25] EasyList, 2019. https://easylist.to/.

[26] FireHOL IP Lists, 2019. http://iplists.firehol.org/.

[27] Gaia - Kodi Streaming, 2019. https://gaiakodi.com.

[28] GitHub, 2019. https://github.com/.

[29] Google Safe Browsing, 2019. https://safebrowsing.google.com/.

[30] LazyKodi, 2019. http://lazykodi.com/.

[31] MaxMind, 2019. https://www.maxmind.com/en/geoip2-databases.

[32] mitmproxy, 2019. https://mitmproxy.org/.

[33] Orion Media Index, 2019. https://orionoid.com.

[34] Real-Debrid, 2019. https://real-debrid.com/.

[35] Reddit, 2019. https://www.reddit.com/.

[36] Top Authors — Kodi, 2019. https://kodi.tv/addon-top-authors.

[37] Tstat - TCP STatistic and Analysis Tool, 2019. http://tstat.polito.it/.

[38] XVFB, 2019. https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

[39] Citrix Intelligent Traffic Management, (accessed May 2019). https://www.citrix.com/products/citrix-intelligent-traffic-management/.

[40] Conviva: Video AI Platform, (accessed May 2019). https://www.conviva.com/.

[41] AGER, B., MÜHLBAUER, W., SMARAGDAKIS, G., AND UHLIG, S. Comparing dns resolvers in the wild. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (2010), IMC '10, ACM, pp. 15–21.

[42] AGER, B., MÜHLBAUER, W., SMARAGDAKIS, G., AND UHLIG, S. Web content cartography. In *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference* (New York, NY, USA, 2011), IMC '11, ACM, pp. 585–600.

[43] ANDERSON, D. Splinternet behind the great firewall of china. *Queue 10*, 11 (Nov. 2012), 40:40–40:49.

[44] ANONYMOUS. The collateral damage of internet censorship by dns injection. *SIGCOMM Comput. Commun. Rev. 42*, 3 (June 2012), 21–27.

[45] BENCHAÏTA, W., GHAMRI-DOUDANE, S., AND TIXEUIL, S. Stability and optimization of dns-based request redirection in cdns. In *Proceedings of the 17th International Conference on Distributed Computing and Networking* (2016), ACM, p. 11.

[46] BENSON, K., DOWSLEY, R., AND SHACHAM, H. Do you know where your cloud files are? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop* (New York, NY, USA, 2011), CCSW '11, ACM, pp. 73–82.

[47] BLAIR, G., BROMBERG, Y.-D., COULSON, G., ELKHATIB, Y., RÉVEILLÈRE, L., RIBEIRO, H. B., RIVIÈRE, E., AND TAÏANI, F. Holons: Towards a systematic approach to composing systems of systems. In *Proceedings of the 14th International Workshop on Adaptive and Reflective Middleware* (New York, NY, USA, 2015), ARM 2015, ACM, pp. 5:1–5:6.

[48] BOOTHE, P., AND BUSH, R. Dns anycast stability. *19th APNIC,'05* (2005).

[49] BÖTTGER, T., ANTICHI, G., FERNANDES, E. L., DI LALLO, R., BRUYERE, M., UHLIG, S., AND CASTRO, I. The elusive internet flattening: 10 years of IXP growth. *CoRR abs/1810.10963* (2018).

[50] BUTKIEWICZ, M., MADHYASTHA, H. V., AND SEKAR, V. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 313–328.

[51] BUTKIEWICZ, M., WANG, D., WU, Z., MADHYASTHA, H. V., AND SEKAR, V. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 439–453.

[52] CALDER, M., FAN, X., HU, Z., KATZ-BASSETT, E., HEIDEMANN, J., AND GOVINDAN, R. Mapping the expansion of google's serving infrastructure. In *Proceedings of the 2013 Conference on Internet Measurement Conference* (2013), IMC '13, ACM, pp. 313–326.

[53] CALDER, M., FLAVEL, A., KATZ-BASSETT, E., MAHAJAN, R., AND PADHYE, J. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference* (2015), IMC '15, ACM, pp. 531–537.

[54] CHEN, F., SITARAMAN, R., AND TORRES, M. End-user mapping: Next generation request routing for content delivery. In *Proceedings of ACM SIGCOMM '15* (London, UK, Aug. 2015).

[55] CHIESA, M., RETVARI, G., AND SCHAPIRA, M. Lying your way to better traffic engineering. In *Proceedings of the 2016 ACM CoNEXT* (2016), CoNEXT '16, ACM.

[56] CHINANETCENTER. Chinanetcenter - network, 2016. http://en.chinanetcenter.com/pages/technology/g2-network-map.php.

[57] CHUN, B., CULLER, D., ROSCOE, T., BAVIER, A., PETERSON, L., WAWRZONIAK, M., AND BOWMAN, M. Planetlab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun. Rev. 33*, 3 (July 2003), 3–12.

[58] CIMPANU, C. Windows and Linux Kodi users infected with cryptomining malware, Sept. 2018. https://www.zdnet.com/article/windows-and-linux-kodi-users-infected-with-cryptomining-malware/.

[59] CISCO SYSTEMS, I. Cisco visual networking index: Forecast and methodology, 2016-2021, 2017. https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html.

[60] CLAY, A. Blocking, tracking, and monetizing: Youtube copyright control and the downfall parody. Institute of Network Cultures: Amsterdam, 2011.

[61] COMMISSION, E. 2018 reform of EU data protection rules, 2018. https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en.

[62] CONTAVALLI, C., VAN DER GAAST, W., LAWRENCE, D., AND KUMARI, W. Client subnet in dns queries. RFC 7871, RFC Editor, May 2016.

[63] DA SILVA, D. V. C., DE A. ROCHA, A. A., VELLOSO, P. B., AND DOMINGUES, G. D. M. A first look at mobile-live-users of a large cdn. In *Proceedings of the 23rd Brazilian Symposium on Multimedia and the Web* (New York, NY, USA, 2017), WebMedia '17, ACM, pp. 81–84.

[64] DER SAR, E. V. Popular Kodi Addon 'Exodus' Turned Users into a DDoS 'Botnet', 2017. https://torrentfreak.com/popular-kodi-addon-exodus-turned-users-into-a-ddos-botnet-170203/.

[65] DING, Y., DU, Y., HU, Y., LIU, Z., WANG, L., ROSS, K., AND GHOSE, A. Broadcast yourself: understanding youtube uploaders. In *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference* (2011), ACM, pp. 361–370.

[66] EDMUNDSON, A., ENSAFI, R., FEAMSTER, N., AND REXFORD, J. A first look into transnational routing detours. In *Proceedings of the 2016 ACM SIGCOMM Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 567–568.

[67] EMBY. Emby - The open media solution., 2019. https://emby.media/.

[68] FAN, X., HEIDEMANN, J., AND GOVINDAN, R. Evaluating anycast in the domain name system. In *2013 Proceedings IEEE INFOCOM* (2013), IEEE, pp. 1681–1689.

[69] FLACH, T., KATZ-BASSETT, E., AND GOVINDAN, R. Quantifying violations of destination-based forwarding on the internet. In *Proceedings of the 2012 ACM Conference on Internet Measurement Conference* (2012), IMC '12, ACM, pp. 265–272.

[70] FLORES, M., AND MCQUISTIN, S. Seeing the world with ripe atlas, 2017. https://labs.ripe.net/Members/verizon_digital/seeing-the-world-with-ripe-atlas.

[71] GANJAM, A., SIDDIQUI, F., ZHAN, J., LIU, X., STOICA, I., JIANG, J., SEKAR, V., AND ZHANG, H. C3: Internet-scale control plane for video quality optimization. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)* (Oakland, CA, May 2015), USENIX Association, pp. 131–144.

[72] GUEYE, B., ZIVIANI, A., CROVELLA, M., AND FDIDA, S. Constraint-based geolocation of internet hosts. *IEEE/ACM Transactions on Networking (TON) 14*, 6 (2006), 1219–1232.

[73] HILDERBRAND, L. Youtube: Where cultural memory and copyright converge. *FILM QUART 61*, 1 (2007), 48–57.

[74] HSIAO, L., AND AYERS, H. The price of free illegal live streaming services. *CoRR abs/1901.00579* (2019).

[75] HUANG, C., BATANOV, I., AND LI, J. A practical solution to the client-ldns mismatch problem. *SIGCOMM Comput. Commun. Rev. 42*, 2 (Mar. 2012), 35–41.

[76] HUANG, C., WANG, A., LI, J., AND ROSS, K. W. Measuring and evaluating large-scale cdns (paper withdrawn at microsoft's request). In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement* (2008), IMC '08, ACM, pp. 15–29.

[77] HUANG, T.-Y., JOHARI, R., MCKEOWN, N., TRUNNELL, M., AND WATSON, M. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM Conference on SIGCOMM* (2014), SIGCOMM '14, ACM, pp. 187–198.

[78] IBOSIOLA, D., STEER, B., GARCIA-RECUERO, A., STRINGHINI, G., UHLIG, S., AND TYSON, G. Movie pirates of the caribbean: Exploring illegal streaming cyberlockers. In *Proc. INTERNATIONAL AAAI CONFERENCE ON WEB AND SOCIAL MEDIA* (2018).

[79] JACCARD, P. New research on floral distribution. *Bull. Soc. Vaud. Sci. Nat. 44* (1908), 223 to 270.

[80] KAKHKI, A. M., JERO, S., CHOFFNES, D., NITA-ROTARU, C., AND MISLOVE, A. Taking a long look at quic: an approach for rigorous evaluation of rapidly evolving transport protocols. In *Proceedings of the 2017 Internet Measurement Conference* (2017), ACM, pp. 290–303.

[81] KARIITHI, N. K. Is the devil in the data? a literature review of piracy around the world. *The Journal of World Intellectual Property 14*, 2 (2011), 133–154.

[82] KATABI, D., AND WROCLAWSKI, J. A framework for scalable global ip-anycast (gia). In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication* (2000), SIGCOMM '00, ACM, pp. 3–15.

[83] KRISHNAN, R., MADHYASTHA, H. V., JAIN, S., SRINIVASAN, S., KRISHNAMURTHY, A., ANDERSON, T., AND GAO, J. Moving beyond end-to-end path information to optimize cdn performance. In *Internet Measurement Conference (IMC)* (Chicago, IL, 2009), pp. 190–201.

[84] LAUINGER, T., ONARLIOGLU, K., CHAABANE, A., KIRDA, E., ROBERTSON, W., AND KAAFAR, M. A. Holiday pictures or blockbuster movies? insights into copyright infringement in user uploads to one-click file hosters. In *Proceedings of the 16th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 8145* (New York, NY, USA, 2013), RAID 2013, Springer-Verlag New York, Inc., pp. 369–389.

[85] MAHANTI, A., CARLSSON, N., ARLITT, M., AND WILLIAMSON, C. Characterizing cyberlocker traffic flows. In *37th Annual IEEE Conference on Local Computer Networks* (2012), IEEE, pp. 410–418.

[86] MAO, Z. M., CRANOR, C. D., DOUGLIS, F., RABINOVICH, M., SPATSCHECK, O., AND WANG, J. 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* (Berkeley, CA, USA, 2002), ATEC '02, USENIX Association, pp. 229–242.

[87] MATIC, S., TYSON, G., AND STRINGHINI, G. Pythia: A framework for the automated analysis of web hosting environments. In *The World Wide Web Conference* (New York, NY, USA, 2019), WWW '19, ACM, pp. 3072–3078.

[88] MCDONALD, A., BERNHARD, M., VALENTA, L., VANDERSLOOT, B., SCOTT, W., SULLIVAN, N., HALDERMAN, J. A., AND ENSAFI, R. 403 forbidden: A global view of cdn geoblocking. In *Proceedings of the Internet Measurement Conference 2018* (New York, NY, USA, 2018), IMC '18, ACM, pp. 218–230.

[89] MEDIAPORTAL, T. MEDIAPORTAL - a HTPC Media Center for free - MEDIAPORTAL, 2017. `https://www.team-mediaportal.com/`.

[90] MOURA, G. C., SCHMIDT, R. D. O., HEIDEMANN, J., DE VRIES, W. B., MULLER, M., WEI, L., AND HESSELMAN, C. Anycast vs. ddos: Evaluating the november 2015 root dns event. In *Proceedings of the 2016 Internet Measurement Conference* (New York, NY, USA, 2016), IMC '16, ACM, pp. 255–270.

[91] MRMC. MrMC – MrMC Media Center, 2017. `https://mrmc.tv/`.

[92] MUKERJEE, M. K., BOZKURT, I. N., MAGGS, B., SESHAN, S., AND ZHANG, H. The impact of brokers on the future of content delivery. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (2016), HotNets '16, ACM, pp. 127–133.

[93] MURTAGH, F. A survey of recent advances in hierarchical clustering algorithms. *The computer journal 26*, 4 (1983), 354–359.

[94] NETRAVALI, R., GOYAL, A., MICKENS, J., AND BALAKRISHNAN, H. Polaris: Faster page loads using fine-grained dependency tracking. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, Mar. 2016), USENIX Association.

[95] NIKAS, A., ALEPIS, E., AND PATSAKIS, C. I know what you streamed last night: On the security and privacy of streaming. *Digital Investigation 25* (2018), 78–89.

[96] NÉDA, Z., VARGA, L., AND BIRÓ, T. S. Science and facebook: The same popularity law! *PLOS ONE 12* (07 2017), 1–11.

[97] OPENDNS. A faster internet: The global internet speedup, 2016. `http://afasterinternet.com`.

[98] OSMC. OSMC, 2019. `https://osmc.tv/`.

[99] PADHYE, J., FIROIU, V., TOWSLEY, D., AND KUROSE, J. 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* (1998), SIGCOMM '98, ACM, pp. 303–314.

[100] PAXSON, V. End-to-end routing behavior in the internet. *SIGCOMM Comput. Commun. Rev. 26*, 4 (Aug. 1996), 25–38.

[101] PLEX. Media Server — Plex media server allows you to stream video smarter., 2019. `https://www.plex.tv/`.

[102] POESE, I., FRANK, B., AGER, B., SMARAGDAKIS, G., AND FELDMANN, A. Improving content delivery using provider-aided distance information. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement* (2010), IMC '10, ACM, pp. 22–34.

[103] PRITCHARD, T. Nearly 70 Per Cent of Kodi Users are Pirates, Claims MPAA, 2017. http://www.gizmodo.co.uk/2017/11/nearly-70-per-cent-of-kodi-users-are-pirates-claims-mpaa/.

[104] PUZHAVAKATH NARAYANAN, S., NAM, Y. S., SIVAKUMAR, A., CHANDRASEKARAN, B., MAGGS, B., AND RAO, S. Reducing latency through page-aware management of web objects by content delivery networks. In *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science* (New York, NY, USA, 2016), SIGMETRICS '16, ACM, pp. 89–100.

[105] RULA, J. P., AND BUSTAMANTE, F. E. Behind the curtain: Cellular dns and content replica selection. In *Proceedings of the 2014 Conference on Internet Measurement Conference* (2014), IMC '14, ACM, pp. 59–72.

[106] SANDVINE. Global Internet Phenomena Spotlight - Kodi, 2018. https://www.sandvine.com/hubfs/downloads/archive/2017-global-internet-phenomena-spotlight-kodi.pdf.

[107] SARKAR, D., RAKESH, N., AND MISHRA, K. K. Problems in replica server placement (rsp) over content delivery networks (cdn). In *Proceedings of the Sixth International Conference on Computer and Communication Technology 2015* (New York, NY, USA, 2015), ICCCT '15, ACM, pp. 175–179.

[108] SCHEITLE, Q., HOHLFELD, O., GAMBA, J., JELTEN, J., ZIMMERMANN, T., STROWES, S. D., AND VALLINA-RODRIGUEZ, N. A long way to the top: significance,

structure, and stability of internet top lists. In *Proceedings of the Internet Measurement Conference 2018* (2018), ACM, pp. 478–493.

[109] SMITH, M. D., AND TELANG, R. Competing with free: The impact of movie broadcasts on dvd sales and internet piracy. *MIS Quarterly 33*, 2 (2009), 321–338.

[110] STREIBELT, F., BÖTTGER, J., CHATZIS, N., SMARAGDAKIS, G., AND FELDMANN, A. Exploring EDNS-client-subnet adopters in your free time. In *Proceedings of IMC '13* (2013), IMC '13.

[111] SU, A.-J., CHOFFNES, D. R., KUZMANOVIC, A., AND BUSTAMANTE, F. E. Drafting behind akamai (travelocity-based detouring). *SIGCOMM Comput. Commun. Rev. 36*, 4 (Aug. 2006), 435–446.

[112] SU, A.-J., AND KUZMANOVIC, A. Thinning akamai. In *Proceedings of the 8th ACM SIGCOMM Conference on Internet Measurement* (2008), IMC '08, ACM, pp. 29–42.

[113] TARIQ, M., ZEITOUN, A., VALANCIUS, V., FEAMSTER, N., AND AMMAR, M. Answering what-if deployment and configuration questions with wise. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication* (New York, NY, USA, 2008), SIGCOMM '08, ACM, pp. 99–110.

[114] TOOMEY, F. Data, The Speed Of Light And You, 2015. `https://techcrunch.com/2015/11/08/data-the-speed-of-light-and-you/`.

[115] VISSICCHIO, S., TILMANS, O., VANBEVER, L., AND REXFORD, J. Central control over distributed routing. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication* (2015), SIGCOMM '15, ACM, pp. 43–56.

[116] WANG, X. S., BALASUBRAMANIAN, A., KRISHNAMURTHY, A., AND WETHERALL, D. Demystifying page load performance with wprof. In *Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13)* (2013), pp. 473–485.

[117] WANG, X. S., KRISHNAMURTHY, A., AND WETHERALL, D. Speeding up web page loads with shandian. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, Mar. 2016), USENIX Association, pp. 109–122.

[118] WARRIOR, M. A., KLARMAN, U., FLORES, M., AND KUZMANOVIC, A. Drongo: Speeding up cdns with subnet assimilation from the client. In *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies* (2017), ACM, pp. 41–54.

[119] XBMC. Kodi — Open Source Home Theater Software, 2019. `https://kodi.tv/`.

[120] XBMC. Official:Forum rules/Banned add-ons, 2019. `https://kodi.wiki/view/Official:Forum_rules/Banned_add-ons`.

[121] ZANDER, S. On the accuracy of ip geolocation based on ip allocation data.