Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System

Matt Calder, *Microsoft/USC;* Manuel Schröder, Ryan Gao, Ryan Stewart, and Jitendra Padhye, *Microsoft;* Ratul Mahajan, *Intentionet;* Ganesh Ananthanarayanan, *Microsoft;* Ethan Katz-Bassett, *Columbia University*

https://www.usenix.org/conference/nsdi18/presentation/calder

This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-939133-01-4

Open access to the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX.

Odin: Microsoft's Scalable Fault-Tolerant CDN Measurement System

Matt Calder,[†⋆] *Manuel Schröder,*[†] *Ryan Gao,*[†] *Ryan Stewart,*[†] *Jitendra Padhye*[†]
Ratul Mahajan,[#] *Ganesh Ananthanarayanan,*[†] *Ethan Katz-Bassett*[§]
Microsoft[†] USC[⋆] Intentionet[#] Columbia University[§]

Abstract

Content delivery networks (CDNs) are critical for delivering high performance Internet services. Using worldwide deployments of front-ends, CDNs can direct users to the front-end that provides them with the best latency and availability. The key challenges arise from the heterogeneous connectivity of clients and the dynamic nature of the Internet that influences latency and availability. Without continuous insight on performance between users, front-ends, and external networks, CDNs will not be able to attain their full potential performance.

We describe Odin, Microsoft's Internet measurement platform for its first-party and third-party customers. Odin is designed to handle Microsoft's large user base and need for large-scale measurements from users around the world. Odin integrates with Microsoft's varied suite of web-client and thick-client applications, all while being mindful of the regulatory and privacy concerns of enterprise customers. Odin has been operational for 2 years. We present the first detailed study of an Internet measurement platform of this scale and complexity.

1 Introduction

Content delivery networks (CDNs) are a key part of the Internet ecosystem. The primary function of a CDN is to deliver highly-available content at high performance. To accomplish this, CDNs deploy Points of Presence (PoPs) around the world that interconnect with other Autonomous Systems (ASes) to provide short, high quality paths between content and end users.

While a CDN's goal is to deliver the best performance to all users in a cost-effective manner, the dynamic, heterogeneous, and distributed nature of the Internet makes this difficult. CDNs serve content to users all over the world, across tens of thousands of ASes, using various forms of Internet access and connection quality. User performance is impacted by Internet routing changes, outages, and congestion, all of which can be outside the control of the CDN. Without constant insight into user performance, a CDN can suffer from low availability and poor performance. To gain insight into user performance, CDNs need large-scale measurements for critical CDN operations such as traffic management [1,2,3,4,5], Internet path performance debugging [6, 7], and deployment modeling [8].

Microsoft operates a CDN with over 100 PoPs around the world to host applications critical to Microsoft's business such as Office, Skype, Bing, Xbox, and Windows Update. This work presents our experience designing a system to meet the measurement needs of Microsoft's global CDN. We first describe the key requirements needed to support Microsoft's CDN operations. Existing approaches to collecting measurements were unsuitable for at least one of two reasons:

- **Unrepresentative performance.** Existing approaches lack coverage of Microsoft users or use measurement techniques that do not reflect user performance.
- **Insensitive to Internet events.** Existing approaches fail to offer high measurement volume, explicit outage notification, and comparative measurements to satisfy key Microsoft CDN use cases.

Next we present the design of Odin, our scalable, fault-tolerant CDN measurement system. Odin issues active measurements from popular Microsoft applications to provide high coverage of Internet paths from Microsoft users. It measures to configurable *endpoints*, which are hostnames or IP addresses of remote target destinations and can be in Microsoft or external networks. Measurement allocation is controlled by a distributed web service, enabling many network experiments to run simultaneously, tailoring measurements on a per-use-case basis as necessary. Odin is able to collect measurements even in the presence of Microsoft network failures, by exploiting the high availability and path diversity offered by third party CDNs. Last, we demonstrate that Odin enables important Microsoft CDN use cases, including improving performance.

There are two key insights that make our design distinct and effective. Firstly, first-party CDNs have an enormous advantage over third-party CDNs in gathering rich measurement data from their own clients. Secondly, integration with external networks provides a valuable opportunity for rich path coverage to assist with network debugging and for enabling fault-tolerance.

2 Background

This section provides background about content delivery networks and Microsoft's deployment.

2.1 Content Delivery Networks

Architecture. A content delivery network (CDN) has geographically distributed server clusters (known as *front-ends*, edges, or proxies), each serving nearby users to shorten paths and improve performance [3, 9, 10, 11] (see Figure 1). Many front-ends are in CDN "points of presence" (PoPs), physical interconnection points where the CDN peers with other ISPs. CDNs typically deploy PoPs at major metro areas. Some CDNs also deploy front-ends in end-user networks or in datacenters. A front-end serves cached content immediately and fetches other content from a *back-end*. Back-ends can be complex web applications. Some CDNs operate *backbone* networks to interconnect PoPs and back-ends.

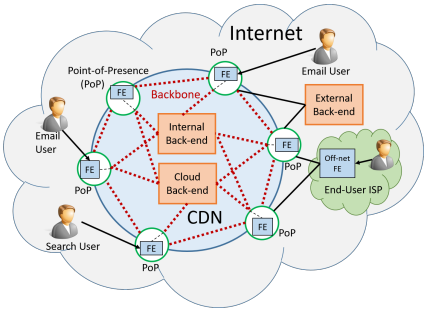

Figure 1: High-level architecture of many CDNs.

Microsoft's CDN is a "hybrid cloud" CDN, i.e., it is used for both its own first-party content as well as for other large third-party customers such as streaming services and online newspapers.

CDN services. Two CDN services are relevant to our paper. Front-ends *cache* static content such as images, JavaScript, CSS, and video. Front-ends also serve as *reverse proxies*, terminating users' TCP connections and multiplexing the requests to the appropriate back-ends via pre-established warm TCP connections [12, 13]. The back-ends forward their responses back to the same front-ends, which relay the responses to the users. Reverse proxies accelerate websites because shorter round-trip times between clients and front-ends enable faster congestion window growth and TCP loss recovery [9].

Client redirection. This work considers the two most common *redirection* mechanisms CDNs use to direct a client request to a front-end for latency sensitive traffic: anycast and DNS. With both mechanisms, when a user desires CDN-hosted content, it issues a request to its *local DNS resolver (LDNS)* for the hostname of the content. The LDNS forwards the request to the CDN's authoritative DNS resolver, and the authoritative resolver returns a record with an IP address of a front-end that can serve the content. With *anycast*, this IP address is announced by multiple PoPs, and BGP routes a request to a PoP based on BGP's notion of best path. The front-end collocated with that PoP then serves the request. With *DNS*-based redirection, the CDN's authoritative resolver returns an IP address for the particular front-end the CDN wants to serve the user from. Because the request to the authoritative resolver generally includes the LDNS (but not user's) IP address, CDN performance benefits from knowledge of which users the LDNS serves (§8.1.1).

2.2 Microsoft's network

Microsoft provides high performance and availability to its customers using a global network with 100+ PoPs, many datacenters, and a Microsoft-operated backbone network interconnecting them. Microsoft operates two types of datacenters. One set is Microsoft's Azure public cloud compute platform [14] which currently has 36 *regions*. The second consists of legacy datacenters, predating Azure. Third-party cloud tenants only run in the Azure datacenters, whereas first-party services operated by Microsoft run in both types. Figure 1 shows Azure regions as "Cloud Back-ends" and private datacenters as "Internal Back-ends".

Redirection of first-party and third-party clients Microsoft currently runs two independent CDNs. A first-party *anycast* CDN runs Microsoft services such as Bing, Office, and XBox [15, 16]. It has more than 100 front-end locations around the world, collocated with all PoPs and several Microsoft public and private datacenters. The second CDN is an Azure traffic management service offered to Azure customers with applications deployed in multiple regions. Whereas Microsoft's first party CDN uses anycast to steer clients, its Azure service uses DNS to direct users to the lowest-latency region. After receiving the DNS response, users connect directly to an Azure region.

2.3 Comparison to other CDNs

Microsoft's architecture closely mirrors other CDNs, especially hybrid-cloud CDNs from Google and Amazon.
End-user applications. All three have web, mobile, and desktop application deployments with large global user bases. Google's include the Chrome Browser, Android OS, Search, YouTube, and Gmail. Amazon's include the Store, Audible, and Prime Video. Microsoft's include Office, Windows OS, Skype, and XBox.
CDN and cloud services. Like Microsoft, Amazon and Google run multiple types of CDNs. Google runs a first-party CDN [6, 7, 9], a third-party CDN [17], and application load balancing across Google Cloud regions [18]. Amazon's equivalent services are CloudFront [19] and Route 53 [20]. Amazon Web Services [21] and Google Cloud Platform [22] are similar to Microsoft Azure [14]. Amazon [10] and Google [23] also run backbone networks.

Because of these similarities, we believe our goals, requirements, and design are applicable to networks beyond Microsoft.

3 Goals and Requirements

We need measurements to support Microsoft's CDN operations and experimentation, leading to the following goals and resulting requirements.

Goal-1: Representative performance reflecting what users could achieve on current and alternate routes.

Requirement: High coverage of paths between Microsoft's users and Microsoft is critical for traffic engineering, alerts on performance degradation, and "what-if" experimentation on CDN configurations, to avoid limited or biased insight into the performance of our network. In particular, our measurements should cover paths to /24 prefixes that combine to account for 90% of the traffic from Microsoft. In addition, they should cover paths to 99% of designated "high revenue" /24 prefixes, which primarily are enterprise customers.

Requirement: Coverage of paths between Microsoft users and external networks to help detect whether a problem is localized to Microsoft and to assess the performance impact of expanding Microsoft's footprint to new sites. External networks may be any CDN, cloud provider, or virtual private server hosting service.

Requirement: Measurements reflect user-perceived performance, correlating with application metrics and reflecting failure scenarios experienced by production traffic, to enable decisions that improve user experience.

Goal-2: Sensitive and quick detection of Internet events.

Requirement: High measurement volume in order to quickly detect events across a large number of users and cloud endpoints, even if the events impact only a small number. Without high measurements counts, events can be missed entirely, or data quality can be too poor to confidently make measurement-driven traffic engineering choices. A reasonable level of sensitivity is the ability to detect an availability incident that doubles the baseline failure rate, e.g., from 0.1% to 0.2%. Figure 12 in Appendix A shows, if we assume measurements fail independently according to a base failure rate, detecting this change would require at least 700 measurements, and detecting a change from 0.01% to 0.02% would require at least 7000 measurements. For confidentiality reasons, we cannot describe our baseline failure rates, but we consider several thousand measurements within a five minute window from clients served by an ISP within one metropolitan region sufficient for our needs.

Requirement: Explicit outage signals, in order to detect events that impact small groups of clients. Historical trends are too noisy to detect the gray failures that make up the majority of cloud provider incidents [24].

Requirement: Fault tolerance in data collection, to collect operation-critical measurements in the presence of network failures between the client and collector.

Requirement: Comparative measurements in same user session for experimentation, providing accurate "apples-to-apples" comparisons when performing an A/B test and minimizing the chance of changing clients or network conditions coloring the comparison between test and control measurements.

Goal-3: Compatible with operational realities of existing systems and applications.

Requirement: Measurements of client-LDNS associations, which are needed to operate both anycast and DNS-redirection CDNs effectively (§2.1,7.1.1,7.2.1).

Requirement: Minimal updates to user-facing production systems, given that network configuration changes are a common cause of online service outages [25].

Requirement: Application compliance across varying requirements. Each Microsoft application independently determines the level of compliance certifications (FISMA, SOC 1-3, ISO 27018, etc.), physical and logical security, and user privacy protections. Application requirements determine the endpoints that can be measured, set of front-ends that can process the measurements, requirements for data scrubbing and aggregation (e.g., IP blocks), and duration of data retention. These strict security policies stem from Microsoft's enterprise customers. Any cloud provider or CDN that serves enterprises, such as Akamai [26], also need to meet these compliance requirements.

4 Limitations of Existing Solutions

This section describes how existing approaches fail to meet our requirements, summarized in Table 1.

1) Third-party measurements platforms provide insufficient measurement coverage of Microsoft users. Non-commercial measurement platforms such as Planetlab, MLab, Caida ARK, and RIPE Atlas have insufficient coverage, with only a few hundred to few thousand vantage points. The largest, RIPE Atlas, has vantage points in 3,589 IPv4 ASes [27], less than 10% of the number of ASes seen by Microsoft's CDN on a standard weekday.

Commercial measurement platforms also lack sufficient coverage. Platforms including Dynatrace [28], ThousandEyes [29], and Catchpoint [30] offer measurements and alerting from cloud-based agents in tier 1 and "middle-mile" (tier 2 and tier 3) ISPs. Cedexis uses a different approach, providing customers with layer 7 measurements collected from users of Cedexis partner websites [31]. However, none of the platforms provides measurements from more than 45% of Microsoft client

Goals	Requirements	Third-party measurement platforms	Layer 3 measurements from CDN infrastructure	Layer 3, DNS from users	Server-side measurements of client connections	Odin
Representative Performance	Coverage of paths between Microsoft users and Microsoft				✓	✓
	Coverage of paths between Microsoft users and external networks				✓	✓
	Measurements reflect user-perceived performance	✓			✓	✓
Sensitive to Internet Events	High measurement volume		✓	✓	✓	✓
	Explicit outage signal	✓	✓	✓		✓
	Fault tolerance	✓	✓	✓		✓
	Comparative measurements in same user session for experimentation		✓	✓	✓	✓
Compatible with Operational Realities	Measurements of client-LDNS associations	✓		✓	✓	✓
	Minimal updates to user-facing production systems	✓	✓	✓		✓
	Application compliance	✓	✓		✓	✓

Table 1: Goals of Odin and requirements to meet our operational CDN needs. No existing approach satisfies all the requirements.

/24 networks. On top of missing over half the networks, the platform with the best coverage provides 10^+ measurements a day from less than 12% of the networks and 100^+ measurements a day from only 0.5% of them, not enough to meet Microsoft's operational need for sensitivity to Internet events.

2) Layer 3 measurements from CDN infrastructure cannot provide representative coverage of the performance of Microsoft users. A CDN can issue measurements such as traceroutes and pings from its front-ends or datacenters to hosts across the Internet. For example, Entact measures the performance along different routes by issuing pings from servers in datacenters to responsive addresses in prefixes across the Internet [1]. One measurement technique used by Akamai is to traceroute from CDN servers to LDNSes to discover routers along the path, then ping those routers as a proxy for CDN to LDNS or end-user latency [32].

However, these measurements cannot provide a good understanding of user performance. Many destinations do not respond to these probes, so Entact was unable to find enough responsive addresses in the networks responsible for 74% of MSN traffic. Similarly, previous work has shown that 45% of LDNS do not respond to ICMP ping or to DNS queries from random hosts [33], and 40% of end users do not respond to ICMP probes [34]. Routers are more responsive than LDNS, with 85% responding to ping [35], but measurements to routers may not reflect a client's application performance because ICMP packets may be deprioritized or rate-limited [36]. All of the above fail to exercise critical layer 7 behaviors including SSL/TLS and HTTP redirection.

3) Layer 3 and DNS measurement from clients may not reflect user-perceived performance and do not provide sufficient coverage. Many systems perform layer 3 measurements from end user devices [37, 38, 39, 40, 41].[1] These measurements are generally dropped by the strict network security policies of enterprise networks. Further, these measurements generally cannot be generated from in-browser JavaScript and instead require installing an application, keeping them from providing measurements from Microsoft's many web users.

4) Server-side measurements of client connections can satisfy some but not all of our use cases. Google [2, 6, 7, 42], Facebook [3], Microsoft [43], and other content providers and CDNs collect TCP- and application-layer statistics on client connections made to their servers [44]. To measure between users and alternate PoPs or paths, CDNs use DNS or routing to direct a small fraction of traffic or client requests to alternate servers or paths. These measurements are useful for performance comparisons, and DNS redirection could steer some of the measurements to measurement servers hosted in external cloud providers. However, if a user cannot reach a server, the outage will not register in server-side measurements, and so these measurements cannot be used to measure fine-grained availability. There are also several practical challenges with only using server-side measurements. While Table 1 shows that technically server-side measurements can be collected on external networks, there are a number of engineering and operational trade-offs that make client-side measurements a better solution for large content providers. The first is that measuring to external networks would mean hosting alternate front-ends on an external provider which immediately raises serious compliance and production concerns. The second issue is that doing A/B network testing with production traffic is considered too high risk with an enterprise customer base.

[1] Ono [37] and Netalyzr [39] also measure throughput.

5 Design Decisions

To meet our goals (§3) and overcome the limitations of other approaches (§4), Odin uses user-side, application-layer measurements of client connections, combining the explicit outage signaling and fault tolerance of user-side measurements (as with layer 3 measurements from users in §4) with the representative performance and coverage achieved by measuring client connections (as with server-side measurements in §4).

Client-side active measurement from Microsoft users. Odin embeds a measurement client into some Microsoft thick clients and web applications. It directs measurement clients to fetch web objects.

This approach helps achieve a number of our requirements. Odin issues measurements from Microsoft users, achieving coverage important to Microsoft's businesses and (by issuing measurements at a rate similar to the use of Microsoft's applications) sensitivity to Internet events, even events that impact only a small fraction of users or connections. By embedding our measurement client into thick clients, Odin can issue measurements even from users unable to reach a Microsoft web server.

Application layer measurements. Odin clients perform DNS resolutions and fetch web objects, measuring availability and timing of these application-layer actions and reporting the results to Odin. The clients can use `http` and `https`, allowing integration with Microsoft applications that require `https`. Unlike ping and traceroute, the measurements are compatible with enterprise networks that host many Microsoft services and users.

These measurements capture the application-layer user performance that we care about, exercising mechanisms across the network stack that can impact performance and availability, including TLS/SSL, web caching, TCP settings, and browser choice. `http` and `https` measurements also provide status code errors that are useful for debugging. They also suffice to uncover user-LDNS associations [45], a key need for both our anycast and DNS redirection CDNs (§7).

External services and smarter clients. We design the clients to conduct measurements and report results even when they cannot reach Microsoft services, as outage reports are some of the most valuable measurements and measurement-dependent operations must continue to function. To build this fault tolerance, clients that cannot fetch measurement configuration or report results fall back to using third-party CDNs for these operations. We use the third-party CDNs to proxy requests to Microsoft and to host static measurement configuration.

Flexible measurement orchestration and aggregation. We built a measurement orchestration system for Odin that supports parallel experiments with different configurations, helping meet a variety of requirements. To accommodate the unique business constraints and compliance requirements of each application that Odin measures to or from, the system provides control over which endpoints an application's users may be given to measure and which servers they upload reports to. When appropriate, experiments can measure to servers in external (non-Microsoft) networks, and clients conduct multiple measurements in a session to allow direct comparisons. By having clients fetch instructions on which active measurements to run, new experiments generally do not require changes to operational services or to clients, reducing operational risk. We also allow for flexibility in aggregation of the measurements (e.g., in 5 minute buckets) for faster upload to our real-time alerting system.

6 System Design

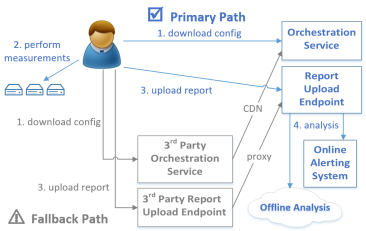

Figure 2: Odin Architecture Overview: CDN clients download measurement config, perform measurements, and upload results. If first-party network sites are unreachable, third-party sites can cache config and relay upload requests.

Figure 2 outlines the Odin measurement process. A number of Microsoft applications embed the Odin client (§6.1). Odin clients in thick applications support a range of measurements. This paper focuses on measuring latency and availability, our highest priorities, supported by thick and web applications.

Step 1: The client uses a background process to fetch a *measurement configuration* from the Orchestration Service (§6.2). The configuration defines the type of measurements and the targets (measurement endpoints).

Step 2: The client issues the measurements. To measure latency and availability, endpoints host a small image on a web server for clients to download. Many Microsoft applications require `https` requests, so measurement endpoints have valid certificates. The endpoints can be in Microsoft front-ends, Microsoft data centers, or third-party cloud/collocation facilities.

Step 3: When the client completes its measurements, it uploads the measurement results to a *Report Upload Endpoint* (§6.3). The Report Upload Endpoint forwards the measurements to Odin's two analysis pipelines.

Step 4: The real-time pipeline performs alerting and network diagnostics, and the offline pipeline enriches measurements with metadata for big data analysis (§6.4).

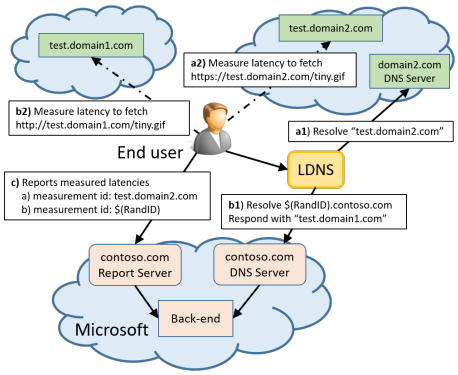

Figure 3: Odin supports two measurement types for latency. Measurement *a* measures the test domain directly. Measurement *b* contacts an Odin authoritative DNS first, which responds with the endpoint to measure. This gives Odin client-LDNS association for a measurement.

6.1 Client

Measurements can vary along 3 dimensions: `http` or `https`, direct or DNS-based, and warm or cold. Figure 3 illustrates direct and DNS-based measurements. The first type has the client performing DNS resolution of `test.domain2.com` in *a1* and fetching the image and recording the latency in *a2*. The measurement to `<randid>.contoso.com` is an example of the second type, which we refer to as a *DNS-based measurement* and which we use to measure web fetch latency *and* client-LDNS association. The domain (`contoso.com`) is one that we control. We design the clients to recognize the `<randid>` scheme and substitute in a unique, transient, random identifier **$(RandID)**.[2] The client then issues a DNS request via the user's LDNS for **$(RandID)**`.contoso.com` (step *b1*). The DNS request goes to our authoritative DNS server, which returns a record for the endpoint Odin wants to measure (`test.domain1.com`) and logs its response associated with the **$(RandID)**. The client then fetches `http://test.domain1.com/tiny.gif`. In step *c*, the client reports its measurements, reporting the ID for the second measurement as **$(RandID)**. The measurement back-end uses **$(RandID)** to join the client's IP address with the DNS log, learning the user-LDNS association.

The Orchestration Service can ask clients to perform "cold" and/or "warm" measurements. A cold measurement initiates a new TCP connection to fetch the image. A warm measurement fetches the image twice and reports the second result, which will benefit from DNS caching and from a warm TCP connection.[3]

[2]Generating the **$(RandID)** at the client rather than at the Orchestration Service lets caches serve the measurement configuration.

[3]The client prevents browser caching by appending a random parameter to the image request (e.g. `tiny.gif?abcde12345`).

Web client vs. thick client measurements. Web clients default to measuring latency using the request start and end times in JavaScript, which is known to be imprecise [46]. If the browser supports the W3C resource-timing API [46], then the client reports that more precise measurement instead, along with a flag that signals that it used the more precise option. If the image fetch fails, the client reports the HTTP error code if one occurred, otherwise it reports a general failure error code. A limitation of in-browser measurements is that low-level networking errors are not exposed to JavaScript. For example, we cannot distinguish between a DNS resolution failure and a TCP connection timeout. Thick clients issue measurements through an Odin application SDK. Unlike web clients, the SDK can report specific low-level networking errors which are valuable in debugging.

6.2 Orchestration Service

The Orchestration Service coordinates and dispatches measurements. It is a RESTful API service that Odin clients invoke to learn which measurements to perform. The service returns a small JSON object specifying the measurements. In the rare case of major incidents with Odin or target Microsoft services, the Orchestration Service has the option to instruct the client to issue no measurements to avoid aggravating the issues.

```
NumMeasurements: 3,
MeasurementEndpoints: [
   {type:1, weight:10, endpoint:"m1.contoso.com"},
   {type:1, weight:20, endpoint:"m2.microsoft.com"},
   {type:2, weight:30, endpoint:"m3.azure.com"},
   {type:3, weight:10, endpoint:"m4.azure.com"},
   {type:2, weight:30, endpoint:"m5.azure.com"},
   {type:1. weight:15, endpoint:"m6.microsoft.com"}],
ReportEndpoints: ["r1.azure.com","r2.othercdn.com"]
```

Listing 1: Example measurement configuration that is served by the orchestration service to the client.

Listing 1 shows an example configuration that specifies three measurements to be run against three out of six potential endpoints. The ability to specify more endpoints than measurements simplifies configurations that need to "spray" measurements to destinations with different probabilities, as is common in CDN A/B testing [16]. The client performs a weighted random selection of three endpoints.

The other component of orchestration is the customized authoritative DNS server for DNS-based measurements (§6.1). When a client requests DNS resolution for a domain such as `12345abcdef.test.contoso.com`, the DNS server responds with a random record to a target endpoint, with the random choice weighted to achieve a desired measurement distribution.

Even a unique hostname used for client-LDNS mapping can generate multiple DNS queries. Our measurements reveal that 75% of unique hostnames result in multiple LDNS requests, and 70% result in requests from multiple LDNS IP addresses. If our authoritative DNS returned different responses for a single hostname, we would be unable to determine from logs which target endpoint the client actually measured. To overcome this issue, we use consistent hashing to always returns the same response for the same DNS query.

The Orchestration Service allocates measurements to clients based on *experiments*. An experiment has an Orchestration Service configuration that specifies the endpoints to be measured, which applications' users will participate, and which report endpoints to use based on compliance requirements of the applications. Experiment owners configure endpoint measurement allocation percentages, and the Orchestration Service converts them into weights in the configuration. The Orchestration Service runs multiple experiments, and experiments may be added or removed at any time.

The Orchestration Service allows Odin to tailor configurations to meet different measurement needs and use cases. For example, the service can generate specialized configuration for clients depending on their geography, connection type, AS, or IP prefix. When experimenting with CDN settings, we tailor Odin configurations to exercise the experimental settings from clients in a particular metropolitan area and ASN. When debugging a performance issue, we tailor configurations to target measurements to an endpoint experiencing problems.

If the Orchestration Service is unavailable, proxies in third-party networks may be used instead. The proxies may act as reverse proxies for the first-party system. Alternatively, if the first-party system is unavailable, a fallback to a cached default configuration can be returned to clients.

6.3 Reporting

Listing 1 shows that the measurement configuration returned by the Orchestration Service also specifies the *primary* and *backup* `ReportEndpoints` for the client to upload measurement results. ReportEndpoints are hosted across the 100+ front-ends of Microsoft's first-party CDN. When a ReportEndpoint receives client measurement results, it forwards them to two Microsoft data pipelines, as shown in Figure 2. If for some reason the Microsoft CDN is unavailable, the client will fall back to using proxies hosted in third-party networks. The proxies forward to a set of front-ends that are not part of the primary set of front-ends.

Fault-tolerant measurement reporting is necessary to support our requirement of an explicit outage signal, since we cannot measure the true availability of Microsoft's first-party CDN if we also report measurements there. Odin's fault-tolerant approach for fetching measurement configuration and uploading results will succeed if the backup reporting channel(s) use a path that avoids the failure and fail if both the primary and backup paths encounter the failure. As long as the client can reach a backup, and the backup can reach at least one of the Odin servers at Microsoft, Odin receives the result, tolerating all but widespread failures that are detectable with traditional approaches and are often outside of Microsoft's control to fix. From operational experience, Odin's handling of faults provides a high level of resilience for our measurement data. We now discuss Odin's behavior in the face of three fault scenarios. We do not consider this an exhaustive treatment of all possible fault scenarios.

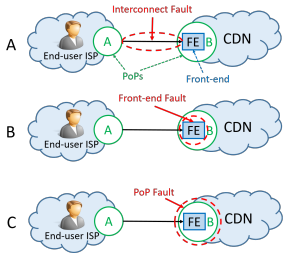

Figure 4: Three types of Internet faults that may occur when fetching measurement configuration or uploading reports.

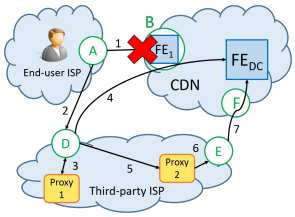

Figure 5: Topology of backup paths when FE_1 is unreachable. FE_1 is a front-end collocated with a PoP while FE_{DC} is a front-end in a datacenter.

Interconnection faults impact an individual link(s) between an end-user ISP and the CDN, caused by issues such as peering misconfiguration or congestion. Connectivity to other ISPs is not impacted. Figure 4(A) shows an interconnection fault between PoPs A and B. Figure 5 shows that, when these faults occur, the client will send a backup request using path $2,3$ to Proxy 1. The proxy then forwards the request back to the CDN by path $3,4$, through D, to a datacenter front-end FE_{DC} instead of FE_1.

Front-end system faults are failures of a front-end due to software or hardware problems, as shown in Figure 4(B). Because the backup proxies connect to a distinct set of front-ends (hosted in datacenters), we gain resilience to front-end system faults, as seen in Figure 5.

PoP-Level faults impact multiple ISPs and a large volume of traffic exchanged at that facility. These faults may be caused by a border router or middle-box misconfiguration or a DDoS attack. In our experience, these faults are rare and short-lived, and so we did not design Odin to be resilient to them. Figure 5 shows that Proxy 1's path to FE_{DC} goes through the same PoP as the client's

path to FE_1, whereas Proxy 2 avoids it. We present a preliminary evaluation of this scenario in Section 8.2.

6.4 Analysis Pipeline

Measurement reports get sent to two analysis pipelines.

Offline Pipeline. The measurement reports include a report ID, metadata about the client (version number of client software, ID of Microsoft application it was embedded in, whether it used the W3C Resource Timing API), and the measurement results, each of which includes a measurement ID, the measurement latency (or failure information), and whether it was a cold or warm measurement. The offline pipeline enriches measurements with metadata including the client's LDNS and the user's location (metropolitan region), ASN, and network connection type. This enriched data is the basis for most operational applications (§7).

Real-time Alerting Pipeline. Many of the target endpoints that Odin monitors are business-critical so must react quickly to fix high latency or unavailability. To ensure fast data transfer to the back-end real-time data analytics cluster, each reporting endpoint reduces data volume by aggregating measurements within short time windows. It annotates each measurement with the client's metropolitan region and ASN, using in-memory lookup tables. Within each window, it groups all measurements to a particular endpoint from clients in a particular ⟨metropolitan region, ASN⟩, then reports fixed percentiles of latency from that set of measurements, as well as the total number of measurements and the fraction of measurements that failed.

7 Supporting CDN Operations with Odin

We use Odin to support key operational concerns of CDNs – performance and availability, plus CDN expansion/evolution and how it impacts the other concerns. The two CDNs we support have sizes of over a hundred sites (which is more than most CDNs) and few dozen sites (which is common for CDNs [47]).

7.1 Directing users to the CDN front-ends

Low latency web services correlate with higher user satisfaction and service revenue. A central proposition of a CDN is that distributed servers can serve users over short distances, lowering latency, but deploying the servers alone does not suffice to achieve that goal.

Odin continuously monitors client performance for both of Microsoft's CDNs. Previous work demonstrated the value of comparing performance of our CDN to another to guard against latency regressions [15]; of comparing performance from one client to multiple CDN servers [16], and of comparing the performance from a CDN to multiple clients in the same city [6]. Odin provides measurements for all these analyses, which can uncover performance problems stemming from circuitous routing in either direction or from poor server selection. This section describes how we use Odin to create high-performance redirection maps for DNS redirection (§7.1.1) and to identify cases in which Internet routing selects poor performing anycast routes (§7.1.2).

7.1.1 Generating low latency DNS redirection maps

Azure's traffic manager service (§2.2) directs a user to a particular Azure region [14] by returning a DNS record for that region. When determining which DNS record to return, the traffic manager knows which LDNS issued the request but not which user.[4] We refer to an instance of the DNS redirection policy as a *map* (from LDNS to IP addresses of Azure regions).

To achieve low latency for users, we need to understand which use each LDNS and their performance to the various regions. Microsoft constructs maps using Odin data as the primary data source, as follows:

(1) Data Aggregation. The offline pipeline annotates each DNS-based measurement with the LDNS the client used (§6.4). We use this associate to group the measurements directed by each LDNS to each Azure region and calculate the median latency to each region from each LDNS. (In practice, before finding the median latency, we aggregate all LDNS within the same /26 IP prefix, which we found balances precision because of IP localization and statistical power from measurement aggregation.)

(2) Filtering. Next, we filter out LDNS-region pairs which do not have enough measurements. Our minimum threshold was chosen using statistical power analysis. If we filter the region that was lowest latency for the LDNS in the currently-deployed map, we do not update the map for the LDNS, to prevent us from making the best decision from a set of bad choices.

(3) Ranking Results. For each LDNS, we rank the regions by latency. At query resolution time, the traffic management authoritative DNS responds to an LDNS with the lowest latency region that is currently online.

(4) Applying the Overrides. The final step is to apply the per-LDNS changes to the currently deployed map, resulting in the new map. The map generation process takes care of prefix slicing, merging, and aggregation to produce a map with a small memory footprint.

7.1.2 Identifying and patching poor anycast routing

Microsoft's first-party CDN uses anycast (§2.2). Anycast inherits from BGP an obliviousness to network performance and so can direct user requests to suboptimal front-ends. We identify incidents of poor anycast routing in Microsoft's anycast CDN by using Odin to measure

[4] Except for the few LDNS that are ECS-enabled [48, 49].

performance of anycast and unicast alternatives from the same user. Our previous study used this methodology for a one-off analysis using measurements from a small fraction of Bing users [16]. Odin now continuously measures at a large scale and automatically generates daily results. As with our earlier work, we find that anycast works well for most–but not all–requests. The traditional approach to resolving poor anycast routing is to reconfigure route announcements and/or work with ISPs to improve their routing towards the anycast address.

While Microsoft pursues this traditional approach, announcements can be difficult to tune, and other ISPs may not be responsive, and so we also patch instances of poor anycast performance using a hybrid scheme that we proposed (but did not implement) in our previous work [16]. The intuition is that both DNS redirection and anycast work well most of the time, but each performs poorly for a small fraction of users. DNS redirection cannot achieve good performance if an LDNS serves widely distributed clients [32], and anycast performs poorly in cases of indirect Internet routing [16]. Since the underlying causes do not strongly correlate, most clients that have poor anycast performance can have good DNS redirection performance. We use Odin measurements to identify these clients, and a prominent Microsoft application now returns unicast addresses to the small fraction of LDNS that serve clients with poor anycast routing.

7.2 Monitoring and improving service availability

The first concern of most user-facing web services is to maintain high availability for users, but it can be challenging to quickly detect outages, especially those that impact only a small volume of requests.

Odin's design allows it to monitor availability with high coverage and fast detection. By issuing measurements from the combined user base of a number of services, it can detect issues sooner than any individual service. By having a single client session issue measurements to multiple endpoints, sometimes including an endpoint outside of Microsoft's network, it can understand the scope of outages and differentiate client-side problems from issues with a client contacting a particular service or server. By providing resilient data collection even in the face of disruptions, Odin gathers these valuable measurements even from clients who cannot reach Microsoft services. Anycast introduces challenges to maintaining high availability. This section discusses how Odin helps address them.

7.2.1 Preventing anycast overload

Monitoring a front-end's ability to control its load. Previous work from our collaborators demonstrated how Microsoft's anycast CDN prevents overload [50]. The approach works by configuring multiple anycast IP addresses and organizing them into a series of "rings" of front-ends. All front-ends are part of the largest ring, and then each subsequent ring contains only a subset of the front-ends in the previous one, generally those with higher capacity. The innermost ring contains only high capacity data centers. Each front-end also hosts an authoritative nameserver. If a front-end becomes overloaded, its authoritative nameserver "sheds" load by directing a fraction of DNS queries to a CNAME for the next ring. These local shedding decisions work well if anycast routes a client's LDNS's queries and the client's HTTP requests to the same front-end, in which case the authoritative nameserver can shed the client's requests.

The previous work used measurements from Odin to evaluate how well HTTP and DNS queries correlate for each front-end [50], a measure of how controllable its traffic is. Odin now continuously measures the correlations and controllability of each front-end, based on its measurements of client-to-LDNS associations.

Designing rings with sufficient controllability. We use Odin data on per front-end controllability to design anycast rings that can properly deal with load. The data feeds a traffic forecasting model that is part of our daily operation. The model predicts per front-end peak load, broken down by application, given a set of rings.

Two scenarios can compromise a front-end's ability to relieve its overload. First, the above approach sheds load at DNS resolution time, so it does not move existing connections. This property is an advantage in that it does not sever existing connections, but it means that it cannot shed the load of applications with long-lived TCP connections. Second, if a front-end receives many HTTP queries from users whose DNS queries are not served from the front-end, it can potentially be overwhelmed by new connections that it does not control, even if it is shedding all DNS requests to a different ring.

We use Odin measurements in a process we call *ring tuning* to proactively guard against these two situations. For the first, we use measurements to identify a high-correlation set of front-ends to use as the outermost anycast ring for applications with long-lived connections. The high-correlation allows a front-end that is approaching overload to quickly shed any new connections, both from the long-lived application and other applications it hosts on other anycast addresses. To guard against the second situation, we use measurements to design rings that avoid instances of uncontrollable load, and we configure nameservers at certain front-ends to shed all requests from certain LDNS to inner rings, to protect another front-end that does not control its own fate.

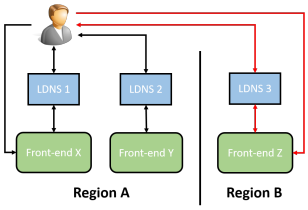

Figure 6: In a regional anycast scenario, if a user's LDNS is served by a front-end in the user's region, the user's performance is unaffected. If the user's LDNS is served by a front-end in a different region, then the user will be served from the distant region, likely degrading performance.

7.2.2 Monitoring the impact of anycast route changes on availability

Long-lived TCP connections present another challenge to anycast: an Internet route change can shift ongoing TCP connections from one anycast front-end to another, severing the connections [51, 52, 53, 54]. Odin measurements address this concern in two ways. First, by having clients fetch an object from both an anycast address and a unicast address, we can monitor for problems with anycast availability. Second, we use Odin to monitor the availability of different candidate anycast rings in order to identify subsets of front-ends with stable routing.

7.3 Using measurements to plan CDN evolution

7.3.1 Comparing global vs regional anycast

In a basic anycast design, all front-ends share the same global IP address. However, this address presents a single point of failure, and a routing configuration mistake at Microsoft or one of our peers has the potential to blackhole a large portion of our customer traffic. An alternate approach is to use multiple regional anycast addresses, each announced by only a subset of front-ends. Such an approach reduces the "blast radius" of a potential mistake, but it can also change the performance of the CDN. A user's request can only end up at one of the front-ends that announces the anycast address given to its LDNS, which might prevent the request from using the best performing front-end . . . or prevent Internet routing from delivering requests to distant front-ends.

Figure 6 shows the three scenarios that can occur when partitioning a global anycast into regions. A user's LDNS may be served by the same front-end as the user or by a different one. If different, the front-ends may be assigned to the same or different regions. If they are assigned to different regions, then the user will be directed away from its global front-end to a different one, likely degrading performance.

In a use case similar to anycast ring tuning, we used Odin to collect data, then used a graph partitioning algorithm to construct regions that minimize the likelihood that a user and their LDNS are served by front-ends in different regions. We construct a graph where vertices represent front-ends and edges between vertices are weighted proportional to the traffic volume where one endpoint serves the DNS query and the other serves the HTTP response. We use an off-the-shelf graph partitioning heuristic package to define 5 regions, each with approximately the same number of front-ends, that minimizes the number of clients directed to distant regions. We compare the performance of regional versus global anycast in Section 8.3.

Country	P75 Imp.	P95 Imp.	Country	P75 Imp.	P95 Imp.
Spain	30.68%	10.79%	Switzerland	10.67%	22.18%
Italy	29.92%	17.95%	Netherlands	7.22%	24.94%
Japan	28.14%	32.02%	France	6.60%	18.14%
Australia	20.05%	16.82%	Norway	5.61%	14.93%
Canada	19.17%	5.10%	U.K.	4.44%	12.39%
Sweden	14.14%	24.02%	Germany	2.82%	5.49%
U.S.A.	14.04%	8.81%	Finland	1.56%	12.97%
South Africa	13.97%	6.33%	Brazil	0.68%	6.18%
India	13.97%	6.08%			

Table 2: The performance improvement in the 75th and 95th percentile from a 2 month roll-out using the Odin-based mapping technique over May and June 2017.

8 Evaluation and Production Results

Odin has been running as a production service for 2 years. It has been incorporated into a handful of Microsoft applications, measuring around 120 endpoints.

8.1 Odin improves service performance

8.1.1 Odin's DNS redirection maps reduce latency

In May 2017, the Azure traffic manager began directing production traffic using maps generated as described in Section 7.1.1, replacing a proprietary approach that combined geolocation databases with pings from CDN infrastructure. We evaluated the performance of the two maps by running an Odin experiment that had each client issue two measurements, one according to each map. Table 2 shows the latency change at the 75th and 95th percentile for the countries with the most measurements. Finland and Brazil saw negligible latency increases (1.56%, 0.68%) at the 75th percentile, but all other high traffic countries saw reductions at both percentiles, with a number improving by 20% or more.

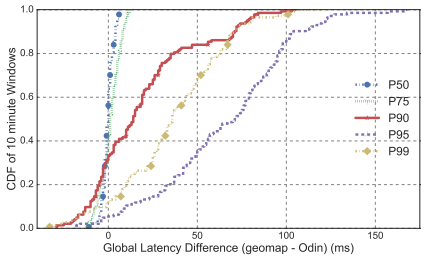

Figure 7: Difference in global performance over one day between a Odin map and a map generated from LDNS locations (geomap). Values less than 0 show the fraction of time that the geomap performed better.

Comparison with alternative DNS mapping techniques. A simple approach to generating a redirection map for a CDN is to use the locations of LDNSes. To test the performance difference between this and the Odin approach, we generated a map using a proprietary Microsoft geolocation database that aggregates locations from many sources. For every IPv4 address, we find the geographically closest front-end and choose that for the map entry. We aggregate neighboring IP addresses with the same map entry and convert this into a set of DNS records. We then configured Odin to measure both this map and the current Odin-based map for 24 hours on Sept. 21, 2017. We bucketed measurements into 10-minute bins. For each bin, we calculated the latency differences at different percentiles. Figure 7 depicts a CDF over all 10 minute bins. Most of the time there is no real difference at the median. The difference is also small at the 75th percentile, although Odin is better around 65% of the time. The real improvement of using Odin comes at the 90th, 95th, and 99th percentile. At P95, Odin's map is better by 65ms half the time.

Dispelling mistaken conventional wisdom. Prior work on CDN performance sometimes exhibited misconceptions about DNS redirection, because operational practices were not transparent to the research community. We distill some takeaways from our work that contradict prior claims and elucidate realities of modern CDNs.

- *For many CDNs, measurements of user connections suffice as the key input to map generation,* whereas previous work often describes mapping as a complex process requiring many different types of Internet measurements [4], including measurements from infrastructure to the Internet [6, 55]. This reality is especially true for CDNs that host popular first-party services, as the CDN has measurement flexibility and control over first party services.
- *The geographic or network location of an LDNS does not impact the quality of redirection*, even though redirection occurs at the granularity of an LDNS. Previous work claimed that redirection decisions were based on the location of or measurements to the LDNS [55], or that good decisions depending on users being near their LDNS [45, 56, 57]. In reality, techniques for measuring associations between users and LDNS have been known for years [45], allowing decisions based on the performance of the users of an LDNS to various front-ends, which provides good performance as long as the users of an LDNS experience good performance from the same front-end as each other.
- *Most redirection still must occur on a per LDNS basis,* even though EDNS client-subnet (ECS) enables user prefix-granularity decisions [32, 48, 55, 58]. Our measurements reveal that, outside of large public resolvers, almost no LDNS operators have adopted ECS.

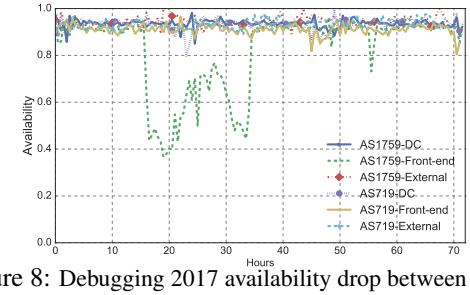

Figure 8: Debugging 2017 availability drop between Helsinki front-end and AS1759 users in Finland.

8.1.2 Odin patches anycast performance

Due to space constraints, we summarize the results from our earlier work [16]. Anycast directed 60% of requests to the optimal front-end, but it also directed 20% of requests to front-ends that were more than 25ms worse than the optimal one. Today we use Odin measurements to derive unicast "patches" for many of those clients.

8.2 Using Odin to identify outages

An outage example. Figure 8 visualizes Odin measurements showing an availability drop for Finnish users in AS1759 during a 24 hour period in 2017. The availability issue was between users in that ISP and a Microsoft front-end in Helsinki. Because Odin measures from many Microsoft users to many endpoints in Microsoft and external networks, it provides information that assists with fault localization. First, we can examine measurements from multiple client ISPs in the same region towards the same endpoint. For readability, we limit the figure to one other ISP, AS719, which the figure shows did not experience an availability drop to the front-end. So, the front-end is still able to serve some user populations as expected. Second, the figure indicates that AS1759 maintains high availability to a different endpoint in Microsoft's network, a nearby datacenter. So, there is no global connectivity issue between Microsoft and AS1759. Last, the figure indicates that availability remains high between clients in AS1759 and an external network. The rich data from Odin allows us to localize the issue to being between clients in AS1759 and our Helsinki front-end.

Reporting in the presence of failures. Odin successfully reports measurements despite failures between end-users and Microsoft. Figure 9 shows the fraction of results reported via backup paths for representative countries in different regions, normalized by the minimum fraction across countries (for confidentiality). During our evaluation period, there were no significant outages so the figure captures transient failures that occur during normal business operations. All countries show a strong

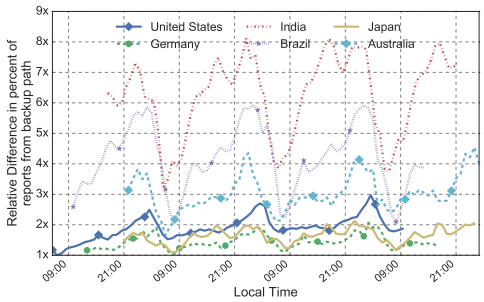

Figure 9: Relative difference per hour in percentage of reports received through the backup path across four weekdays.

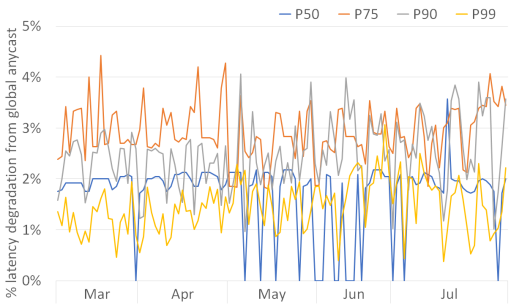

Figure 10: Latency degradation of 5-region vs. global anycast.

diurnal pattern with peaks around midnight and valleys around 8 a.m. local time. Interestingly, the peaks of highest failover reports occur well outside of working hours, when Microsoft's traffic volume is low. This is consistent with previous work which found that search performance degraded outside business hours, because of an increase in traffic from lower quality home broadband networks relative to traffic from well-provisioned businesses [43].

The percentage of reports uploaded through third-parties varies significantly across countries. For example, at peak, Brazil have 3x and 4x the percentage of backup reports as compared to Germany and Japan. Another distinguishing characteristic across countries is the large difference in range between peaks and valleys. India ranges from $\approx 3\times$ to $\approx 8\times$ the baseline, Australia from $\approx 2\times$ to $\approx 4\times$, and Japan from $\approx 1\times$ to $\approx 2\times$

Backup path scenarios. Backup proxies forward report uploads to datacenter front-ends instead of front-ends collocated with PoPs (§6). To illustrate why this is necessary, we allocated a small fraction of Odin measurements to use an alternate configuration in which the third-party proxies instead forward traffic to an anycast ring consisting of front-ends at the same PoPs as the primary `ReportEndpoints`. The third party CDN has roughly the same number of front-end sites as Microsoft. Out of 2.7 billion measurements collected globally over several days in January 2018, around 50% were forwarded to the same front-end by both the third-party proxies and the primary reporting pathway, meaning that the reports could be lost in cases of front-end faults.

Fault-tolerance for PoP-level failures. Figure 4(C) shows an entire PoP failing. It is likely that the nearest front-end and nearest backup proxy to the end-user are also relatively close to each other. When the proxy forwards the request, it will likely ingress at the same failing PoP, even though the destination is different.

To route around PoP-level faults, we want the end-user to send the backup request to a topologically distant proxy, such as Proxy 2 in Figure 5. The proxy will forward the request through nearby CDN PoP *F* and avoid the failure. To test this, we configured two proxy instances in a popular cloud provider, on the East and West Coasts of the United States. These proxies forward requests to the set of front-ends collocated at Microsoft PoPs. We configured a load balancing service to send all requests to the East Coast instance by default, but with an exception to direct all requests from East Coast clients to the West Coast proxy. After collecting data globally for several days, we observed that only 3% of backup requests enter Microsoft's network at the same PoP as the primary, as opposed to the 50% above. This prototype is not scalable with so few proxy instances, but demonstrates an approach to mitigate PoP-level faults that we will develop in a future release.

8.3 Using Odin to evaluate CDN configuration

This section uses Odin to evaluate the performance impact of regional rings as compared to a global anycast ring (§7.3.1). The cross-region issue illustrated in Figure 6 still has the potential to introduce poor anycast performance, even though our graph partitioning attempts to minimize it. To measure the impact to end users, we configure an Odin experiment that compares the latency of the regional anycast ring with our standard "all front-ends" ring. Figure 10 shows that performance change at the median is relatively small – just about 2%. The 75th percentile consistently shows the highest percentage of degradation over time, fluctuating around 3%. While the median and 75th percentiles are stable over the five months, both 90th and 99th percentiles begin to trend higher in the starting in May, suggesting that static region performance may change over time at higher percentiles.

8.4 Evaluating Odin coverage

In this section we examine Odin's coverage of Microsoft's users as part of our requirement to cover paths between Microsoft users, Microsoft, and external networks. We will examine four applications which we have integrated with Odin. We have categorized them by their user base: General, Consumer, and Enterprise.

We first look at how much of Microsoft's user base is covered by individual and combined applications. Figure 11 shows Consumer1, Consumer2, and Enterprise1 have similar percent coverage of Microsoft end users by

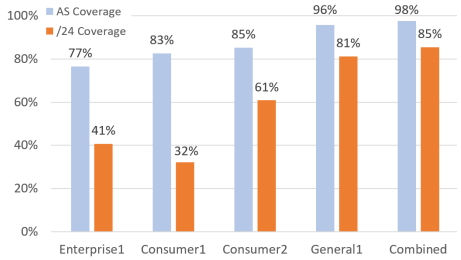

Figure 11: Percentages of ASes and /24s with measurements based on 4 properties with different user populations.

AS. The benefit of multiple applications is more apparent when looking at /24 coverage. We see that all four applications combined cover 85% of /24s whereas individually all except for General1 cover much less. We also examined the overlap between application coverage and found that the four properties only see around 42% pairwise overlap in /24 coverage, meaning that individual applications contribute a substantial amount of user diversity to Odin. General1 is the highest distinct contributor by providing about 18% of distinct /24s observed.

Breaking down the coverage by "high revenue" (e.g. Reverse Proxy for Enterprise scenarios), "medium revenue" (e.g. Consumer email, ad-supported content) and "low revenue" (commodity caching workloads), we observe a higher /24 prefix coverage with Odin for "high revenue" (95%) compared to "medium" (91%) and "low" (90%). This suggests that the missing coverage of Odin is in the tail of relatively low-value traffic.

9 Related Work

There has been a considerable prior work on improving and evaluating general CDN performance [4, 56, 59, 60, 61, 62, 63]. Prior work has also explored cooperation between ISPs and CDNs. Specifically, the efficacy of ISPs releasing distance maps to CDNs to enable more accurate client to server mappings [59], or ISPs hosting CDN servers on demand [60]. WISE [8] is a tool that predicts the deployment implications of new infrastructure in CDNs by using machine learning. WhyHigh [6] and LatLong [7] focus on automating troubleshooting for large content providers like Google, using active measurements and passive latency data, respectively.

Entact [1], EdgeFabric [3], Espresso [2] measure the quality of egress paths from a CDN to the Internet. Entact describes a technique to measure alternative paths to Internet prefixes by injecting specific routes (/32) at border routers to force egress traffic through a particular location. These paths are utilized by a collection of "pinger" machines deployed in DCs to target IPs likely to be responsive within a prefix. EdgeFabric and Espresso direct a small percent of user traffic through alternative egress paths to measure alternate path performance.

Fathom [41], Netalyzr [39], Ono [37], Via [64], Dasu [38] are thick client applications that run measurements from end user machines; BISmark [40] measures from the home routers. Akamai collects client-side measurements using their Media Analytics Plugin [65] and peer-assisted NetSession [32, 66] platform. From commercial measurement platforms, Cedexis is the closest in nature to Odin. Cedexis partners with popular websites with large user bases such as LinkedIn and Tumblr that embed Cedexis' measurement JavaScript beacon into their page. Cedexis customers register their own endpoints to be measured by a portion of end-users of Cedexis' partners. In this way, a customer collects measurements to their endpoints from a relatively large user base. Conviva is a commercial platform which uses extensive client-side measurements from video players to optimize video delivery for content publishers [67, 68].

Akamai published a study on DNS-based redirection [32] showing that enabling ECS [48] greatly improved the performance of user. Alzoubi et al. [51, 53] have examined properties of anycast CDNs. Follow up work focuses on handling anycast TCP session disruption due to BGP path changes [52]. Our work is complementary and orthogonal to our colleagues' work, FastRoute [50], that load balances within an anycast CDN.

Odin uses a user-to-LDNS association technique similar to [34, 45] whereas Akamai uses their NetSession download manager software to obtain client-to-LDNS mappings [32]. Measuring latency using JavaScript beacons is a well established technique [16, 69].

10 Conclusion

CDNs are critical to the performance of large-scale Internet services. Microsoft operates two CDNs, one with 100+ endpoints that uses anycast and one for Azure-based services that uses DNS-redirection. This paper describes Odin, our measurement system that supports Microsoft's CDN operations. These operations span a wide variety of use cases across first- and third-party customers, with clients spread out worldwide. Odin has helped improve the performance of major services like Bing search and guided capacity planning of Microsoft's CDN. We believe that the key design choices we made in building and operating Odin at scale address the deficiencies of many prior Internet measurement platforms.

Acknowledgements

We thank the anonymous NSDI reviewers for a constructive set of reviews. We thank our shepherd, Michael Kaminsky, for providing insightful recommendations and thorough comments on multiple drafts. The work was partially supported by NSF awards CNS-1564242, CNS-1413978, and CNS-1351100.

References

[1] Z. Zhang, M. Zhang, A. G. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian, "Optimizing Cost and Performance in Online Service Provider Networks," in *NSDI*, pp. 33–48, 2010.

[2] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain, *et al.*, "Taking the Edge off with Espresso: Scale, Reliability and Programmability for Global Internet Peering," in *SIGCOMM*, pp. 432–445, ACM, 2017.

[3] B. Schlinker, H. Kim, T. Cui, E. Katz-Bassett, H. V. Madhyastha, I. Cunha, J. Quinn, S. Hasan, P. Lapukhov, and H. Zeng, "Engineering Egress with Edge Fabric," in *SIGCOMM*, 2017.

[4] V. Valancius, B. Ravi, N. Feamster, and A. C. Snoeren, "Quantifying the benefits of joint content and network routing," in *SIGMETRICS*, pp. 243–254, ACM, 2013.

[5] H. H. Liu, R. Viswanathan, M. Calder, A. Akella, R. Mahajan, J. Padhye, and M. Zhang, "Efficiently Delivering Online Services over Integrated Infrastructure," in *NSDI*, pp. 77–90, 2016.

[6] R. Krishnan, H. V. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao, "Moving Beyond End-to-End Path Information to Optimize CDN Performance," in *IMC*, 2009.

[7] Y. Zhu, B. Helsley, J. Rexford, A. Siganporia, and S. Srinivasan, "LatLong: Diagnosing wide-area latency changes for CDNs," in *Transactions on Network and Service Management*, 2012.

[8] M. Tariq, A. Zeitoun, V. Valancius, N. Feamster, and M. Ammar, "Answering What-If Deployment and Configuration Questions with WISE," in *SIGCOMM*, pp. 99–110, ACM, 2008.

[9] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing Web Latency: The Virtue of Gentle Aggression," *SIGCOMM*, vol. 43, no. 4, pp. 159–170, 2013.

[10] J. Hamilton, "AWS re:Invent 2016: Amazon Global Network Overview." https://www.youtube.com/watch?v=uj7Ting6Ckk.

[11] "Netflix Open Connect." https://media.netflix.com/en/company-blog/how-netflix-works-with-isps-around-the-globe-to-deliver-a-great-viewing-experience.

[12] Y. Chen, S. Jain, V. K. Adhikari, and Z.-L. Zhang, "Characterizing Roles of Front-end Servers in End-to-End Performance of Dynamic Content Distribution," in *IMC*, pp. 559–568, ACM, 2011.

[13] A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, R. Kern, J. Li, and K. W. Ross, "Measuring and Evaluating TCP Splitting for Cloud Services," in *PAM*, pp. 41–50, Springer, 2010.

[14] "Azure regions." https://azure.microsoft.com/en-us/regions/.

[15] A. Flavel, P. Mani, D. A. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "Fastroute: A Scalable Load-aware Anycast Routing Architecture for Modern CDNs," in *NSDI*, vol. 27, p. 19, 2015.

[16] M. Calder, A. Flavel, E. Katz-Bassett, R. Mahajan, and J. Padhye, "Analyzing the Performance of an Anycast CDN," in *IMC*, pp. 531–537, ACM, 2015.

[17] "Google Cloud CDN." https://cloud.google.com/cdn/.

[18] "Google Cloud Load Balancer." https://cloud.google.com/load-balancing/.

[19] "Amazon CloudFront." https://aws.amazon.com/cloudfront/.

[20] "Amazon AWS Route53." https://aws.amazon.com/route53/.

[21] "Amazon Web Services." https://aws.amazon.com/.

[22] "Google Cloud Platform." https://cloud.google.com/.

[23] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, "B4: Experience with a Globally-deployed Software Defined WAN," in *SIGCOMM*, pp. 3–14, ACM, 2013.

[24] P. Huang, C. Guo, L. Zhou, J. R. Lorch, Y. Dang, M. Chintalapati, and R. Yao, "Gray Failure: The Achilles' Heel of Cloud-Scale Systems," in *HotOS*, 2017.

[25] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or Die: High-Availability Design Principles Drawn from Googles Network Infrastructure," in *SIGCOMM*, pp. 58–72, ACM, 2016.

[26] "Akamai Compliance Management." https://www.akamai.com/uk/en/multimedia/documents/product-brief/akamai-for-compliance-management-feature-sheet.pdf.

[27] "Ripe atlas network coverage." https://atlas.ripe.net/results/maps/network-coverage/.

[28] "Dynatrace." https://www.dynatrace.com/capabilities/synthetic-monitoring/.

[29] "Thousandeyes." https://www.thousandeyes.com/.

[30] "Catchpoint." https://www.catchpoint.com.

[31] "Cedexis." https://www.cedexis.com/.

[32] F. Chen, R. K. Sitaraman, and M. Torres, "End-user Mapping: Next Generation Request Routing for Content Delivery," in *SIGCOMM*, vol. 45, pp. 167–181, ACM, 2015.

[33] C. Huang, N. Holt, A. Wang, A. G. Greenberg, J. Li, and K. W. Ross, "A DNS Reflection Method for Global Traffic Management.," in *USENIX ATC*, 2010.

[34] C. Huang, D. A. Maltz, J. Li, and A. Greenberg, "Public DNS System and Global Traffic Management," in *INFOCOM*, pp. 2615–2623, IEEE, 2011.

[35] M. H. Gunes and K. Sarac, "Analyzing Router Responsiveness to Active Measurement Probes," in *PAM*, pp. 23–32, Springer, 2009.

[36] R. A. Steenbergen, "A Practical Guide to (correctly) Troubleshooting with Traceroute," *NANOG 37*, pp. 1–49, 2009.

[37] D. Choffnes and F. E. Bustamante, "Taming the Torrent: A Practical Approach to Reducing Cross-ISP Traffic in Peer-to-Peer Systems," in *SIGCOMM*, 2008.

[38] M. A. Sánchez, J. S. Otto, Z. S. Bischof, D. R. Choffnes, F. E. Bustamante, B. Krishnamurthy, and W. Willinger, "Dasu: Pushing Experiments to the Internet's Edge," in *NSDI*, pp. 487–499, 2013.

[39] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson, "Netalyzr: Illuminating the Edge Network," in *IMC*, pp. 246–259, ACM, 2010.

[40] S. Sundaresan, S. Burnett, N. Feamster, and W. De Donato, "BISmark: A Testbed for Deploying Measurements and Applications in Broadband Access Networks.," in *USENIX ATC*, pp. 383–394, 2014.

[41] M. Dhawan, J. Samuel, R. Teixeira, C. Kreibich, M. Allman, N. Weaver, and V. Paxson, "Fathom: A Browser-based Network Measurement Platform," in *IMC*, pp. 73–86, ACM, 2012.

[42] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan, "An Internet-wide Analysis of Traffic Policing," in *SIGCOMM*, pp. 468–482, ACM, 2016.

[43] Y. Chen, R. Mahajan, B. Sridharan, and Z.-L. Zhang, "A Provider-side View of Web Search Response Time," in *SIGCOMM*, pp. 243–254, ACM, 2013.

[44] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, and F. Zane, "Clustering and Server Selection Using Passive Monitoring," in *INFOCOM*, vol. 3, pp. 1717–1725, IEEE, 2002.

[45] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang, "A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers.," in *USENIX ATC*, pp. 229–242, 2002.

[46] A. Jain, J. Mann, Z. Wang, and A. Quach, "W3C Resource Timing Working Draft." https://www.w3.org/TR/resource-timing-1/, July 2017.

[47] "USC CDN Coverage." http://usc-nsl.github.io/cdn-coverage.

[48] C. Contavalli, W. van der Gaast, S. Leach, and E. Lewis, "RFC7871: Client Subnet in DNS Queries." https://tools.ietf.org/html/rfc7871.

[49] M. Calder, X. Fan, Z. Hu, E. Katz-Bassett, J. Heidemann, and R. Govindan, "Mapping the Expansion of Google's Serving Infrastructure," in *IMC*, pp. 313–326, ACM, 2013.

[50] A. Flavel, P. Mani, D. Maltz, N. Holt, J. Liu, Y. Chen, and O. Surmachev, "FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs," in *NSDI '15*, 2015.

[51] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast CDNs Revisited," in *WWW*, 2008.

[52] Z. Al-Qudah, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast-aware Transport for Content Delivery Networks," in *WWW*, 2009.

[53] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van Der Merwe, "A Practical Architecture for an Anycast CDN," *ACM Transactions on the Web (TWEB)*, 2011.

[54] L. Wei and J. Heidemann, "Does Anycast Hang up on You?," in *TMA*, IEEE, 2017.

[55] G. Gürsun, "Routing-aware Partitioning of the Internet Address Space for Server Ranking in CDNs," *Computer Communications*, vol. 106, pp. 86–99, 2017.

[56] J. S. Otto, M. A. Sánchez, J. P. Rula, and F. E. Bustamante, "Content Delivery and the Natural Evolution of DNS," in *IMC*, 2012.

[57] J. S. Otto, M. A. Sánchez, J. P. Rula, T. Stein, and F. E. Bustamante, "namehelp: Intelligent Client-side DNS Resolution," in *SIGCOMM*, pp. 287–288, ACM, 2012.

[58] "A Faster Internet." http://www.afasterinternet.com/participants.htm.

[59] I. Poese, B. Frank, B. Ager, G. Smaragdakis, S. Uhlig, and A. Feldmann, "Improving Content Delivery with PaDIS," *Internet Computing*, vol. 16, no. 3, pp. 46–52, 2012.

[60] B. Frank, I. Poese, Y. Lin, G. Smaragdakis, A. Feldmann, B. Maggs, J. Rake, S. Uhlig, and R. Weber, "Pushing CDN-ISP Collaboration to the Limit," *CCR*, vol. 43, no. 3, pp. 34–44, 2013.

[61] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Web Content Cartography," in *IMC*, 2011.

[62] E. Nygren, R. K. Sitaraman, and J. Sun, "The Akamai Network: A Platform for High-performance Internet Applications," in *SIGOPS*, pp. 2–19, ACM, 2010.

[63] M. J. Freedman, E. Freudenthal, and D. Mazieres, "Democratizing Content Publication with Coral," in *NSDI*, vol. 4, pp. 18–18, 2004.

[64] J. Jiang, R. Das, G. Ananthanarayanan, P. Chou, V. Padmanabhan, V. Sekar, E. Dominique, M. Goliszewski, D. Kukoleca, R. Vafin, and H. Zhang, "Via: Improving internet telephony call quality using predictive relay selection," in *SIGCOMM*, 2016.

[65] S. S. Krishnan and R. K. Sitaraman, "Video Stream Quality Impacts Viewer Behavior: Inferring Causality using Quasi-experimental Designs," *Transactions on Networking*, vol. 21, no. 6, pp. 2001–2014, 2013.

[66] M. Zhao, P. Aditya, A. Chen, Y. Lin, A. Haeberlen, P. Druschel, B. Maggs, B. Wishon, and M. Ponec, "Peer-assisted Content Distribution in Akamai NetSession," in *IMC*, pp. 31–42, ACM, 2013.

[67] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang, "C3: Internet-Scale Control Plane for Video Quality Optimization," in *NSDI*, vol. 15, pp. 131–144, 2015.

[68] J. Jiang, V. Sekar, H. Milner, D. Shepherd, I. Stoica, and H. Zhang, "CFA: A Practical Prediction System for Video QoE Optimization," in *NSDI*, pp. 137–150, 2016.

[69] Adnan Ahmed, Zubair Shafiq, Harkeerat Bedi, Amir Khakpour, "Peering vs. Transit: Performance Comparison of Peering and Transit Interconnections," in *ICNP*, 2017.

Appendices

A Measurement Counts

Let the number of measurements be n and the true failure rate be p. Analytically, the observed failure rate $\hat{p}$ is distributed as $Bin(n,p)/n$, so the average error is

$$E\left[|\hat{p}-p|\right] = E\left[\left|\frac{Bin(n,p)}{n} - p\right|\right].$$

Figure 12, however, is generated computationally via Monte Carlo simulations. For example, to find the value described in the caption, we simulated a large number (10^7) of draws from the binomial distribution

$$\hat{p} \sim Bin(n = 200, p = 0.01)/200,$$

then found the average value of $|\hat{p}-p| \approx 54\%$.

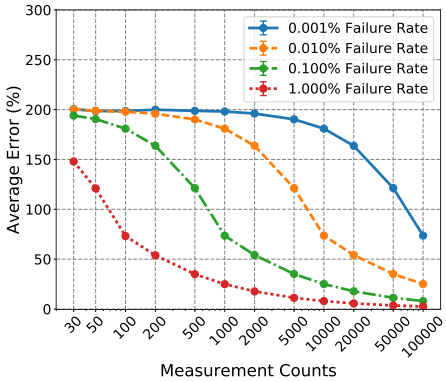

Figure 12: The average error of observed failure rate, as a function of number of measurements and true failure rate. For example, if the true failure rate of a service is 1.0% (red dotted line), then a sample of 200 measurements would yield an average error of about 50%, i.e., $1.0 \pm 0.5\%$.