FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs

Ashley Flavel, Pradeepkumar Mani, David A. Maltz, and Nick Holt, *Microsoft;*
Jie Liu, *Microsoft Research;* Yingying Chen and Oleg Surmachev, *Microsoft*

https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/flavel

This paper is included in the Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI '15).

May 4–6, 2015 • Oakland, CA, USA

ISBN 978-1-931971-218

FastRoute: A Scalable Load-Aware Anycast Routing Architecture for Modern CDNs

Ashley Flavel
Microsoft
ashleyfl@microsoft.com

Pradeepkumar Mani
Microsoft
prmani@microsoft.com

David A. Maltz
Microsoft
dmaltz@microsoft.com

Nick Holt
Microsoft
nickholt@microsoft.com

Jie Liu
Microsoft Research
jie.liu@microsoft.com

Yingying Chen
Microsoft
yinchen@microsoft.com

Oleg Surmachev
Microsoft
olegsu@microsoft.com

Abstract

Performance of online applications directly impacts user satisfaction. A major component of the user-perceived performance of the application is the time spent in transit between the user's device and the application existing in data centers. Content Delivery Networks (CDNs) are typically used to improve user-perceived application performance through a combination of caching and intelligent routing via proxies. In this paper, we describe *FastRoute*, a highly scalable and operational anycast-based system that has significantly improved the performance of numerous popular online services. While anycast is a common technique in modern CDNs for providing high-performance proximity routing, it sacrifices control over the load arriving at any individual proxy. We demonstrate that by collocating DNS and proxy services in each FastRoute node location, we can create a high-performance, completely distributed system for routing users to a nearby proxy while still enabling the graceful avoidance of overload an any individual proxy.

1 Introduction

The latency between user requests and computer reactions directly relates to user engagement and loyalty [11, 31]. With the expansion of applications connecting to cloud servers, the network latency between users and cloud servers becomes a major component of the overall application performance.

As network delays are largely proportional to the routing distance between clients and servers, application operators often employ services of Content Distribution Networks (CDNs). A CDN deploys proxy servers in geographically dispersed locations and then tunnels user requests through them. By utilizing these proxies, the CDN provides (among other things) performance improvements via techniques such as caching and split-TCP connections [27, 19].

While the overarching goal of latency reduction is universal, the logic to determine the proxy an individual user is routed to can be CDN-specific, based on what they offer to their customers, what the requirements of the application are, and what capacity limits the CDN has. Concentrating solely on the "optimal" proxy selection for every user based on latency can introduce additional complexity in the routing logic. In contrast, our design goals were two-fold, a) deliver a low-latency routing scheme that performed better than our existing CDN, and b) build an easy-to-operate, scalable and simple system

The operational aspect of the design goal results in an architecture that is willing to sacrifice some user performance in scenarios that occur rarely in order to maintain a simple design pattern. A major early design choice was to utilize anycast routing (see Section 2.2) as it enabled each FastRoute node (see Section 3.1) to operate independently of other FastRoute nodes (i.e. no real-time communication between nodes). Anycast routing has also successfully been used to deliver content by other CDNs including Edgecast and Cloudflare [28].

Although anycast routing has its advantages, there is the potential for an individual FastRoute node to become overloaded with user traffic (as the CDN does not control which proxy receives traffic, leaving it at the mercy of the intermixed routing policies of Internet Service Providers). To account for this, the FastRoute architecture utilizes multiple "layers" of FastRoute nodes — each with its own anycast IP (see Section 3.2.2). When a FastRoute node in a layer is overloaded, traffic is redirected to node(s) in the next layer. This layer-based approach is an example of choosing simplicity over user performance (for the expected, but rare overloaded node scenario[1]). I.e. instead of attempting to direct users from an overloaded node to a nearby node regardless of layer, we route users to the closest node in the next layer.

An artifact of using anycast routing is that the DNS

[1]If an overloaded node is not rare it indicates a build-out capacity issue.

must choose which anycast IP address to return to a client without knowing which proxy (of the ones announcing that address) the client's traffic will reach. Consequently, some intelligence is needed to determine which DNS responses must be redirected to the next layer. In Section 3.2.1 we show that by collocating DNS servers with proxy servers in the same node locations, a large percentage of user and DNS traffic land at the same a FastRoute node. Although the likelihood of a DNS query and its associated subsequent HTTP requests landing on the same node is not 100% (73% in our network), for the purposes of offloading traffic this has proven sufficient in production for over 2 years. Consequently, a FastRoute node only needs to know about its own load — preserving the independence of nodes that anycast provides.

Although load management is a critical component of the overall system, it is expected to operate rarely. In most situations, users will be routed to the first layer and their performance is based on the intermixed decisions of all ISPs in the Internet. Although technically possible to dynamically influence routing decisions in real-time (e.g. [8]), the system needed to do this would require significant development effort and most critically — introduce additional complexity. Consequently, in Section 4 we introduce several monitoring tools we use for analysing user performance offline that influence our peering policies and who we choose to peer with.

FastRoute has been in operation for several years improving the performance of a number of popular Internet applications. The initial move to FastRoute from a third-party CDN demonstrated noticeable performance improvements. Further, the simple load management implementation has handled all such overload scenarios since its inception, and its fully distributed nature has enabled us to quickly add new proxy locations – further improving our user performance.

Our contributions in this paper are the following novel and interesting aspects of FastRoute:

Architecture:

- A scalable architecture that is robust, simple to deploy and operate, and just complex enough to handle challenges such as overloaded proxies due to anycast routing
- A simple, yet unique DNS-based load management technique that leverages the self-correlation between proxy traffic and DNS traffic in a collocated DNS and proxy system
- Use of multiple anycast rings of FastRoute nodes for load absorption to prevent ping-ponging of load between overloaded proxies

Longitudinal Experimental Results from Production System:

- Data showing that FastRoute works effectively, even in the face of traffic that would overload a simpler system.
- Data showing that the DNS-to-Proxy correlation is relatively stable and high across time.
- Data showing performance (latency) improvements at the 75th and 95th percentiles with our initial limited set of FastRoute nodes, for customers of 10 ISPs in the USA.
- Data showing Anycast stability is sufficient to run a production system

2 Background

Content Distribution Networks direct users to their nearby proxies to improve their performance. In this section we will first examine two fundamental technologies that are core to many of the techniques used to route traffic to the optimal proxy. We will then examine several known techniques that CDNs utilize to help select and route users to a nearby proxy.

2.1 DNS

The Domain Name System (DNS) [23] translates human friendly names (such as `www.contoso.com`) into IP addresses (such as 1.2.3.4). DNS utilizes a hierarchical system where end-users consult recursive DNS resolvers that identify and query an authoritative DNS resolver to discover the translated IP address. The recursive DNS resolver caches the translation for the duration of the time-to-live (TTL) associated with the response. Other clients that utilize the same recursive DNS resolver will receive the same response for the duration of the TTL.

2.2 Anycast Routing

Anycast is a routing technique historically popular with DNS systems due to its inherent ability to spread DDOS traffic among multiple sites as well as provide low latency lookups. It utilizes the fact that routers running the de-facto standard inter-domain routing protocol in the Internet (BGP [30]), select the shortest (based on policy and the BGP decision process) of multiple routes to reach a destination IP prefix. Consequently, if multiple destinations claim to be a single destination, routers independently examine the characteristics of the multiple available routes and select the shortest one (according to the BGP path selection process). The effect of this is that individual users are routed to the closest location claiming to be the IP prefix (see [28] for a good explanation of Anycast routing). Note that latency is not a consideration in the BGP path selection process. However, by "tuning" anycast routing announcements and negotiating policies of peering ISPs (as in Section 4), BGP routing decisions can align with latency based routing.

2.3 Proxy Selection Techniques

FastRoute uses Anycast TCP to route a user to a proxy. In this section, we describe the general approach a CDN would use with Anycast TCP as well as examine several other alternatives (a summary is included in [9]).

2.3.1 Anycast TCP

Anycast TCP is used by many modern CDNs including Edgecast and CloudFlare[28]. This approach has all proxies responding on the same IP address and Internet routing protocols determine the closest proxy[8]. If a location cannot serve traffic, it withdraws its route and the Internet's routing protocols route users to the next closest location.

A difficulty with this approach is that control over where user traffic lands is relinquished to Internet routing protocols. Consequently, avoiding the overload of an individual proxy by controlling routes becomes challenging to accomplish in an automated fashion, as this either requires a traffic engineering technique such as [17, 29, 8] or a DNS based approach as presented in this paper.

A second concern with this approach is that a route change in the middle of a TCP session can result in the user communicating with an alternative proxy mid-session. Attempts can be made to direct rogue TCP flows and route them to the correct proxy[8]; however, much like [22], we analysed the availability of end-users (see Section 5.3), finding the availability of an anycast destination for a small file download was equivalent to unicast indicating this issue did not warrent implementing such a solution at the time of creation.

2.3.2 Anycast DNS

Utilizing an anycast based DNS has become the standard mechanism used by major providers to offer a high level of performance and denial-of-service protection. However, one mechanism a CDN can use to select which proxy to route traffic to takes advantage of information obtained from **where** the DNS lookup occurs [15]. By co-locating the DNS servers with each proxy, a request landing on a proxy simply returns the unicast IP of the collocated proxy. This is a simple solution that utilizes the Internet's routing protocols to find the shortest route to proxy location. However, with this approach, the 'closest' proxy selected is based on 'closeness' to the recursive DNS of the user instead of the 'closeness' to the user themselves. In practice we found the closest proxy for the DNS and user (self-correlation) for our network was the same for 73% of requests based on the analysis described in Section 3.2.1. Although we do not use this unicast based approach (due to it sacrificing the performance benefits of Anycast TCP), our architecture can easily be modified to return unicast IPs instead of anycast IPs (making the self correlation 100%).

2.3.3 Internet Map

An Internet map based approach relies on projecting the entire IP address space onto the set of available proxies (e.g. [13]). By collecting performance data either passively [20, 25], actively through probing mechanisms [12, 24], or statically mapping using geo-to-IP mappings [4], a map can be created that maps IP address space to proxy locations. The map is updated over time as network conditions change or proxies come up/down.

We did not pursue this approach (in contrast to the Anycast TCP approach) as it required global knowledge to analyse user latency data as well as real-time load and health metrics of nodes to decide where to route traffic. Further, the lack of granularity of DNS based responses [26] (which is the predominant method to route users based on an Internet map), the lack of motivation for ISPs (who still perform a majority of the DNS resolutions for users — 90% in the USA from our calculations) to implement more granular DNS requests [14, 18, 26] and the additional complexity introduced when supporting IPv6[2] when supporting IPv6 made this approach less appealing.

2.3.4 Other techniques

Several other techniques including manifest modification for video providers [7] (not applicable to other content types) or HTTP redirection are possible. The content we are delivering was predominantly dynamic content making the cost of a HTTP redirection high compared to transfer time making this approach infeasible.

3 FastRoute Architecture

In this section we describe a) the components within an individual FastRoute node b) why we can make local

[2]DNS requests are **not** guaranteed to be made over the same protocol for the answer they are requesting. I.e. an IPv4 (A record) resolution can be made over IPv6 and vice-versa. Consequently, supporting IPv6 introduces a 4x complexity over Ipv4 only by adding an additional 3 maps.

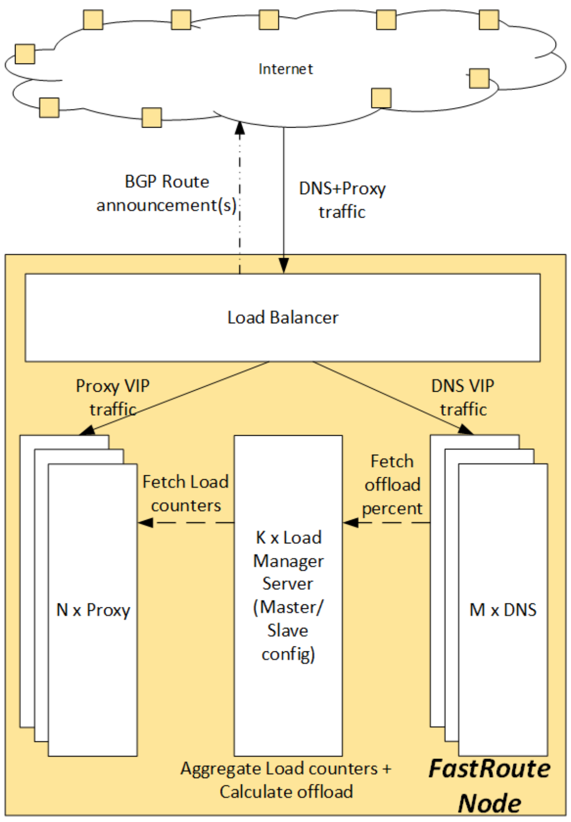

Figure 1: The FastRoute node architecture consists of 4 major components: Load Balancer (to balance traffic to a single Virtual IP (VIP) to multiple instances), DNS (to answer user DNS queries), Proxy (to serve application traffic) and Load Manager (to determine the offload percentage).

decisions on redirecting load away from a node and c) how the local algorithm makes its decisions.

3.1 FastRoute Node

In this section we describe the services within an individual FastRoute node and the communication between them. As explained in Section 3.2, no communication is needed outside an individual node to route users to a proxy.

In Figure 1 we show the four major stand-alone services that exist within a FastRoute node:- Load Balancer, Proxy, DNS and Load Manager. Each service may be on independent or co-existing on the same physical machines.

The Load Balancer is responsible for spreading user traffic between N instances of the Proxy and DNS traffic between M instances of the DNS service. When the number of healthy proxy or DNS services drops below a threshold, the anycast BGP prefixes of the DNS and proxy are withdrawn. Equivalently, when the number of healthy proxy and DNS services is higher than a threshold the BGP prefixes are announced. Announcing and withdrawing routes is the mechanism by which a FastRoute node either chooses to receive traffic or not.

The Proxy service is responsible for handling user traffic (e.g. terminating user TCP sessions, caching, fetching content from origin servers, blocking DDOS traffic etc). For each type of traffic it is handling, a counter defining the load is published locally.

The DNS service responds to each DNS query with one of two possible responses: either the anycast IP of its own FastRoute node or a CNAME (redirection) to the next layer (details included in Section 3.2). The probability of returning the CNAME is determined by reading at regular intervals a configuration published by the load management service.

The load management service is responsible for aggregating the counters collected across all proxy nodes and publishing the probability of returning the redirection CNAME for each DNS name. It operates in a master/slave configuration so all DNS services within the node receive the same offload probability. Details of the algorithm the load management service uses are included in Section 3.2.

3.2 Distributed Load Management

When no individual proxy is receiving more traffic than it is configured to handle, the operation of the system follows the pure anycast mechanism with all DNS requests being responded with the same IP address and the Internet's routing protocols determine the closest proxy for each user.

However, as described in Section 2.3.1, Internet routing protocols have no knowledge of the load on any individual proxy. Consequently, an individual proxy can be overloaded when using anycast if proxy locations aren't significantly over provisioned relative to the expected traffic load. The ability to over-provision such proxy locations is often limited as power and space comes at a premium. Further, the ability to add new capacity to overloaded locations has significant lead-time (can be weeks to months), hence we must have the ability to dynamically shift load in real-time - even if we expect this situation to be rare.

We considered two main techniques for Load Management - (1) altering BGP routes, and (2) modifying DNS responses. When an individual proxy is overloaded, utilizing BGP techniques such as AS Path pre-pending or withdrawing routes to one or more peers is one way to reduce the traffic on the proxy. However, such techniques are difficult to perfect as they can suffer from cascading failures (as an action from one proxy can cause a traffic swarm to a nearby proxy causing it to take action, and so on). A centralized system could be used to manage such actions; however, a significant amount of complexity would need to be introduced into predicting where traffic would route to, given a particular action. Further, taking BGP-level actions when utilizing an anycast based system results in route-churn and subsequently, an increased rate of TCP session breakages.

Conversely, modifying DNS responses enables the BGP topology to remain unchanged. This has two main attractive features. First, the BGP topology improvements and monitoring described in Section 4.1 remain independent to load management. Second, modifying DNS responses will only affect the routing of new users (as users already connected to a proxy will continue their session). Hence, DNS is a less abrupt change to users and a more gradual shift of overall traffic patterns than modifying BGP. However, there are two primary difficulties when using DNS for load management: first, the DNS server must infer that its response will result in additional traffic landing on an overloaded proxy (or not); second, given that a DNS server knows which DNS responses to modify to prevent load from landing on an overloaded proxy, what answer does it then respond with to redirect users causing the excessive load?

To solve the first difficulty, we used an artifact used by many traditional CDNs — the LDNS and the users of that LDNS are often in a similar network location. Consequently, we collocated our authoritative DNS servers and proxy servers in a FastRoute Node. Our hypothesis was that there would be a high correlation between the location of the proxy receiving the user traffic and the authoritative DNS server receiving the DNS request. Given a high correlation, by altering only the decision of the collocated DNS server, we can divert traffic and avoid overloading the proxy. This has a very appealing characteristic that the only communication needed is between the collocated proxy and DNS in a given FastRoute node. We discuss more on this communication in Section 3.3.2.

The second difficulty we encounter is where to direct traffic that would normally be directed to an overloaded proxy. One option is to return the unicast IP address of the next closest non-overloaded proxy, however, this in essence means creating a parallel system like the Internet map as described in Section 2.3.3 for the (expected) rare situation of an overloaded proxy. In contrast, we translate the problem from one of “where to send the traffic”, to one of “where not to send the traffic”. As we only have collocated proxy-DNS pair communication, the only load a DNS server is aware of is its own. Consequently, each DNS simply knows to direct traffic to “not-me”. By configuring multiple anycast IP addresses on different sets of proxies, the DNS server can direct

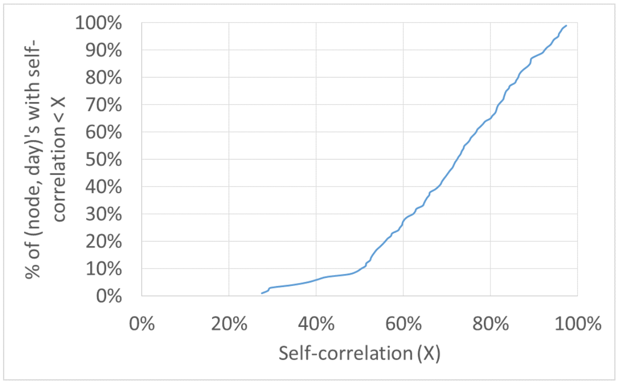

Figure 2: CDF of self-correlation values observed each day for all FastRoute nodes over a week. Each datapoint is a self-correlation for an individual node for a single day. 90% of datapoints have a self-correlation greater than 50% with no datapoint less than 27% justifying FastRoute’s decisoin to use onl local decisions for load management.

traffic to one of the anycast IPs that it is not responsible for. However, when using such an approach, it is possible that multiple proxies experience high load and direct traffic to each other — causing more load on proxies that are already overloaded. The underlying problem is there is a possibility for ping-ponging of traffic among overloaded proxies, if we are not careful. We address this concern in Section 3.2.2, by setting up loop-free traffic diversion paths.

3.2.1 Local Controllability of Load

If DNS queries and subsequent user traffic to proxies lands on the same FastRoute Node, we call such user traffic as *controllable*. We measure correlation between two FastRoute nodes *i* and *j* as the likelihood of the DNS query landing on FastRoute node *i* (DNS response is the anycast IP of the proxy) and the subsequent user traffic landing on the proxy in node *j*. The *self-correlation* of any node *i* is a measure of the controllable load on that node. Every node could have a mix of controllable and uncontrollable load. From the data gathered using the approach shown in [16], we can construct a correlation matrix for all the nodes in the system. The diagonal of the correlation matrix gives the self-correlation values for the various nodes.

For our solution to be able to handle a given load, we rely on the self-correlation being high enough to offload sufficient traffic to avoid congestion. In Figure 2 we show the CDF of self-correlation values observed each day for every FastRoute node using over 10 million DNS-to-HTTP request mappings collected over a week in February 2015 from a representative sample of users of a major application utilizing FastRoute. Each node contributed around 7 data points (one self-correlation value per day) towards computing the CDF. Any FastRoute node receiving less than 100 samples on an individual day was excluded from the results.

We see that more than 90% of $(Node, day)$ pairs have a self-correlation greater than 50%. No node on any day had a self-correlation below 27%. Further, when examining the individual node self-correlation values, they remain relatively constant over the entire week.

For the nodes with self-correlation below 50% we found that the *cross-correlation* with either one or several other neighboring nodes was relatively high. For example, one node in Europe with a self-correlation of approximately 28% had four other nodes in nearby cities with cross-correlation values of $\tilde{2}0\%$, 18%, 17% and 10%. A distinct North American node also with a self-correlation of approximately 28% had a single other node with a cross-correlation value of 50%. We see this pattern (of a small subset of nodes that have high cross-correlation values) consistently throughout other nodes with relatively low self-correlation.

FastRoute does not currently attempt to do anything special for nodes with low self-correlation. This is based on our design principle of simplicity — do not build unnecessary complexity unless absolutely needed (and so far it has not). The two FastRoute nodes discussed above serve less than 2% of total global traffic and are sufficiently over-provisioned to handle the load they receive. However, if any node (low self-correlation or not) is unable to offload sufficient traffic, we have the ability to alert an operator to manually divert traffic from other nodes (based on the historic non-diagonal terms of the correlation matrix).

In the future, if operators are being involved sufficiently often enough to justify the additional complexity, we can implement one or more of the following features:-

- Lowly correlated nodes can "commit suicide" (i.e., withdraw DNS and Proxy anycast BGP routes) when an offered load is unable to be sufficiently diverted, resulting in traffic (expected) to land on nearby nodes with higher self-correlation values and can divert traffic if necessary. This keeps our current desirable system property of no real-time communication between nodes.

- Lowly correlated nodes can inform the small set of nearby nodes that have a high cross-correlation to start offloading traffic (e.g. in the examples presented earlier, this would increase the ability to offload traffic to 93% for the European node and 78% for the North American node). This breaks our current system property of no real-time communication between nodes, but does limit it to a small subset of nodes.

- Nodes with low self-correlation can be configured in "anycast-DNS" mode (i.e. DNS served over anycast, but proxy over unicast addresses; see Section 2.3.2). Such nodes could always be configured in this mode, or nodes could automatically transition to this mode when they cannot divert sufficient traffic.

- The proxy can take steps to divert traffic including reducing its workload (e.g. dropping lower priority traffic) or diverting traffic via HTTP 302 redirects.

As more FastRoute nodes are added, we will continue to monitor the correlation matrix to ensure it is sufficient to handle our traffic patterns.

3.2.2 Loop-free Diversion of Load

So far we have discussed the control over load landing on a proxy with purely local actions (The DNS altering its decision to divert traffic away from the collocated proxy). We now discuss how we determine what the altered response should be.

Our approach is one that utilizes anycast *layers* where each layer has a different anycast IP address for the DNS and proxy services. Each DNS knows only the domain name of its parent layer. Under load, it will start CNAME'ing requests to its parent layer domain name[3]. By utilizing a CNAME, we force the recursive resolver to fetch the DNS name resolution from a FastRoute node within the parent layer. This mechanism ensures that a parent layer node has control over traffic landing in the parent layer with the parent layer following the same process if it becomes overloaded.

We see an example setup of anycast layers in Figure 3. Here we see FastRoute nodes 1 and 2 in the outermost layer becoming overloaded. This results in both nodes diverting traffic to the middle layer resulting in additional traffic landing on nodes 3 and 4. Node 4 determines that it is now being overloaded as a result and diverts load to the innermost layer with node 5 receiving the additional traffic. From a user perspective, although their DNS requests may be bounced off several nodes, their proxy traffic will not experience the redirects.

Higher level layers are not required to be as close to users as lower level layers, consequently, they can be in physical locations where space is relatively cheap and easy to add capacity (e.g. within large data centers with elastic capacity [21, 1]). Hence, bursts of traffic can be

[3]A CNAME is an alias within the DNS protocol that causes the recursive resolver to undertake a new lookup

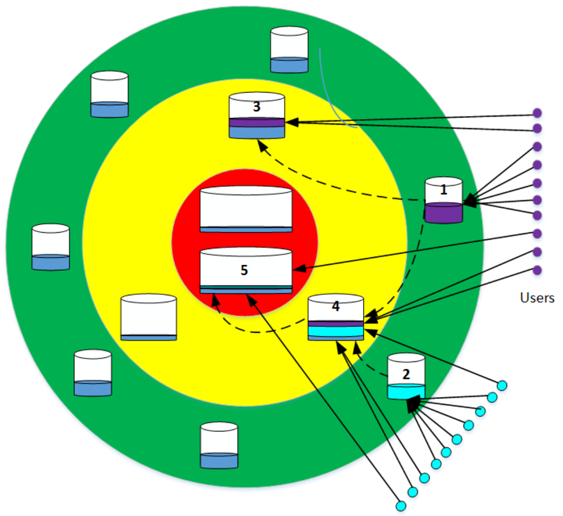

Figure 3: An example configuration with three Anycast *layers*. Solid arrows denote user connections, while dotted arrows denote the effect of diverting traffic by nodes that would otherwise be in overload.

handled by over-provisioning. By diverting lower priority traffic from higher layers first (as in Section 3.3.1) we can avoid the perceived user performance impact.

Although we have shown a single directed path between the lowest layer and highest layer, more advanced configurations are possible. Several extensions include an individual proxy may have two parent layers and offload proportionally between the layers (we did operate in this mode initially when the highest layer did not have sufficient spare capacity), different applications may have a different relationship between layers or individual proxies may exist in multiple layers (i.e. a layer may consist of locations that are subset of a lower layer). The only requirements are that the relationship between layers be loop-free and the highest layer be able to handle the load with no ability to divert traffic.

3.3 Local Offload Algorithm

In this section, we will discuss our approach to use DNS to manage load on a collocated proxy. We will begin by defining the notion of *load* first: user traffic hitting a proxy will consume various system resources such as CPU, memory, network bandwidth, etc., in the proxy. We refer to the strain placed on these resources as *load*. The nature of "load" could vary based on the traffic hitting each end point in the proxy (e.g. short HTTP request-response type queries are generally bottlenecked by CPU; file streaming applications are generally bottlenecked by network bandwidth, etc.). We can control "load" on a particular resource by controlling the user traffic hitting the end point(s) associated with the "load". For every such identified loaded resource associated with the end point (one resource per end point), we define an *overload threshold*, that defines the operating boundary of the proxy, and we consider the proxy overloaded if the load on any resource exceeds the threshold. The goal of FastRoute's load management scheme is to operate the system such that the load on any resource in a given proxy stays under the overload threshold. Also, as each FastRoute load manager instance expects a fraction of traffic that is not controllable locally, multiple instances of the load management service can operate on different endpoints hosted on the same physical machine — even if they utilize each other's bottlenecked resource (e.g. a filestreaming application may be bottlenecked by network bandwidth, but still consumes CPU). This behavior simply alters the fraction of uncontrollable load each load manager instance sees.

3.3.1 When to Divert Load?

In our design it is up to an individual node to discover when it is overloaded and divert some traffic. The load management algorithm that controls offload should be able to quickly recognize an overload situation and divert (just enough) load to another layer, so as to bring load under the overload threshold; equally important for the algorithm is to recognize that the overload situation has passed, and reduce or stop the offloading, as appropriate. Also, it is important to note that any delay in offloading during overload will cause measurable user impact (may cause service unavailability), while any delay in bringing back the traffic once overload has passed, has a relatively smaller penalty and user impact (e.g. higher latency due to being served from a farther layer). The two types of load to expect are:-

- Slowly increasing/decreasing load. This load is caused by the natural user patterns throughout the day. Generally, over a day a diurnal pattern is seen based on users' activities in the timezone of the proxy's catchment zone. Figure 4 shows diurnal traffic pattern observed over a period of 3 days in a proxy in production.

- Step changes in load. This is caused by a nearby proxy going down and all traffic from that proxy hitting the next closest proxy. We show an example of one such occurrence from our production system in Figure 5.

Consequently, our algorithm that determines which DNS answer to return must handle the above two scenarios. Challenges in this algorithm surround the limitations of the control mechanism. These include:-

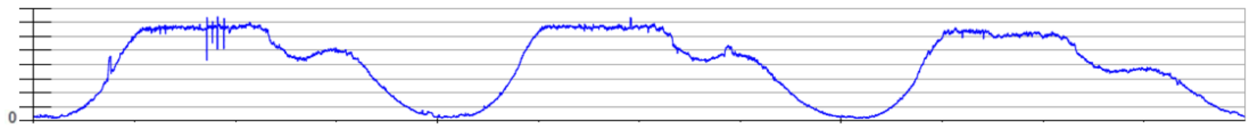

Figure 4: Traffic pattern over a period of 3 days for a single node. The Y-axis represents traffic volume. We have removed the values for confidentiality purposes

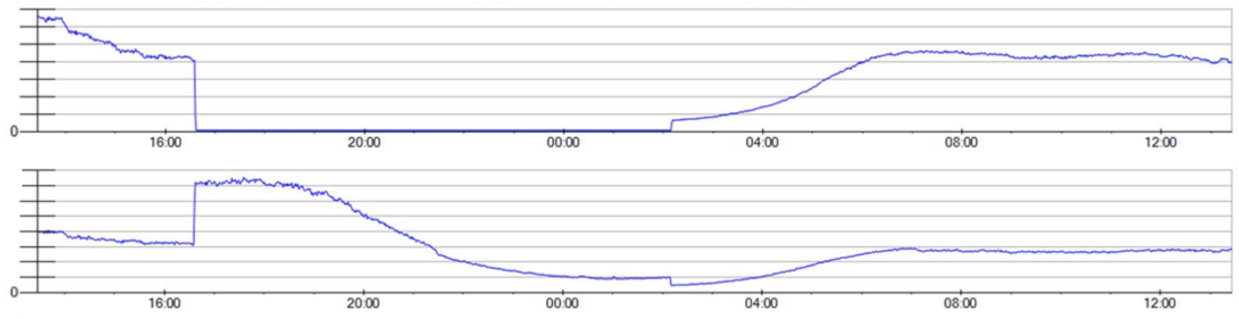

Figure 5: At around 17:00, a neighboring proxy (top) fails and as a result the closest proxy (bottom) is hit with all the load. The Y-axis represents traffic volume. We have removed the values for confidentiality purposes

- The TTL on a DNS response causes a delayed response to changes. Though it was shown in [16] that load management using DNS is feasible, the delay due to TTL is unavoidable.

- Local DNS servers have differing numbers of users behind them.

- A user's DNS lookup may not land on the same proxy as their TCP traffic (see Section 3.2.1 for analysis). Consequently, some load on a proxy (from its perspective) is uncontrollable.

Given the limitations of the control mechanism we have, we would like our control algorithm to be able to

- Quickly reduce load when a step change forces traffic above a desired utilization.

- Prioritize low value traffic to be offloaded

- Alert if the "uncontrollable" load becomes too large to maintain load under the desired utilization.

Many algorithms can support these characteristics. We present a simplified version of our algorithm in production. Let S be the current load on a given resource at node i, T be the overload threshold that is set as the operating boundary for this resource, under which we expect the proxy to operate at all times, and x be the fraction of traffic being offloaded to the next higher layer (offload probability). In order for the load management control loop to function effectively, the load sampling interval is set to higher than twice the TTL of the DNS responses (note that the TTL on the responses reflected the desire of responsiveness; i.e. longer TTL implied that a sustained overload condition and slower reaction to overload was acceptable).

- if $S > T$, the node i is overloaded. Offload probability x is increased super-linearly (maximum value = 1)

- if $S < T$, the node i is NOT overloaded. Offload probability x is decreased linearly (minimum value = 0)

As an extension, we implemented priority-based offloading of different end points that have the same load characteristics (no results shown in this paper). Among end points that contend for the same resource, we defined load management policies such that offload happens in some desired priority order. For example, say the proxy is the end point for both `http://www.foo.com` and `http://www.bar.com`, and traffic to either of these end points will use up CPU. Suppose, `http://www.foo.com` is more "important" than `http://www.bar.com`, and say the overload threshold is set to 70%. When overload occurs (i.e. CPU use exceeds 70%), the system will begin offloading customers of `http://www.bar.com` in an effort to control the load on the system. If the overload persists, then customers of `http://www.bar.com` are fully offloaded before offloading any customers of `http://www.foo.com`. If overload persists even after offloading 100% of customers of both endpoints, then manual intervention is sought by engaging personnel from the FastRoute operations team to artificially increase the measured load on highly cross-correlated neighboring nodes causing an increased diversion of traffic away from the overloaded node.

3.3.2 Scalability and Independent Operation

Given that (a) our operating assumption is that each node has sufficient self-correlation, and (b) the DNS and proxy are collocated in the FastRoute node, it thus follows that the load management system situated at any given node only needs to monitor the various aspects of load on the local node. Once it has collected the load data, the load management system computes the offload fraction for a given end point, and it only needs to communicate the results to the local DNS. Thus, all communication needed to make load management work effectively is contained fully within the same FastRoute node, without the need for any external input or sharing of global state, which makes the operation of FastRoute nodes completely independent of one another, and allows for simplified and easy-to-scale deployment.

4 Improving Anycast Routes over Time

We chose an anycast TCP based approach for FastRoute due to its simplicity and low dependence on DNS for optimal proxy selection. Consequently, we rely heavily on BGP (the de-facto standard inter-domain routing protocol used in the Internet) to best direct users to the closest proxy. The underlying assumption when using anycast is that the shortest route chosen by BGP is also the lowest latency route. However, due to the way the BGP route selection process operates, this may not always be the case. Although possible to implement a real-time system that adapts to the current network topology to modify route announcments of “flip-back” to unicast, this would introduce additional complexity — something we wished to avoid.

Consequently, we opt for a primarily offline approach to monitoring the behavior of anycast. We utilize the user performance telementry to analyse daily user performance (see Section 4.1) to prioritize network peering improvements and identify performance changes for a set of users (see Section 4.2). Availability is most critical, hence we monitor availability in real-time via active Internet based probes such as [3, 2, 5] and internal probes (from within the node itself).

4.1 Identifying Performance Problems

One of the most valuable visualization techniques we developed as part of FastRoute was to overlay performance data collected from users of our production application(s) on top of a Bing map. Multiple views of user performance data were then plotted on top of this map providing unprecedented insight into how our users were experiencing our service. The most basic view we created using this technique is shown in Figure 6. Here we see users in Washington state. Navigation timing data [6] for these users are aggregated based on the user’s geographic location, the ISP they are connected to and the proxy that they connect to. We display this data by sizing the bubble based on the relative number of users in the aggregate group and coloring the bubble based on the relative performance the users receive (red (worst), orange, yellow, light green, dark green (best)). From the example in Figure 6 we can quickly determine that we have a significant user base in the Seattle region (as expected due to the large population in this area)[4]. We can also see that one particular ISP is experiencing lower levels of performance than others in the same region. Upon further investigation (with data contained in the flyout box that appears when hovering over this bubble), we found this ISP was a cellular provider - expected to have slower performance than a cable or DSL network.

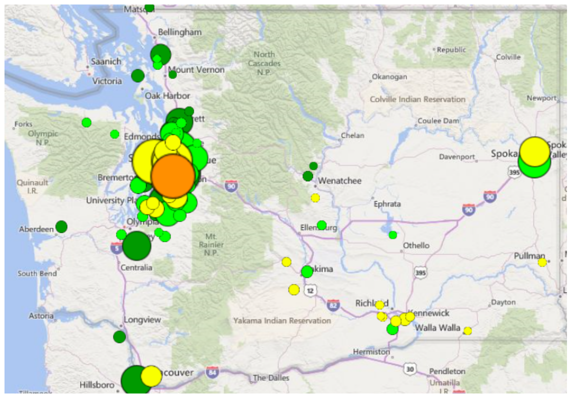

Figure 6: User performance grouped by ISP, geographic location and proxy location displayed on a Bing map. The size of the bubble represents the relative number of users. The color of the bubble represents the relative performance.

This display quickly identified large user populations that were receiving a lower level of performance than others. By filtering by individual proxies, it became immediately obvious when users were routed to a sub-optimal location (e.g. if European users were being routed to the North America). We found the performance of major ISPs to be relatively constant day-over-day. Consequently, by identifying the ISPs whose users

[4]Note that we have introduced random jitter around the actual geo-location of the user population to avoid bubbles being drawn directly on top of each other

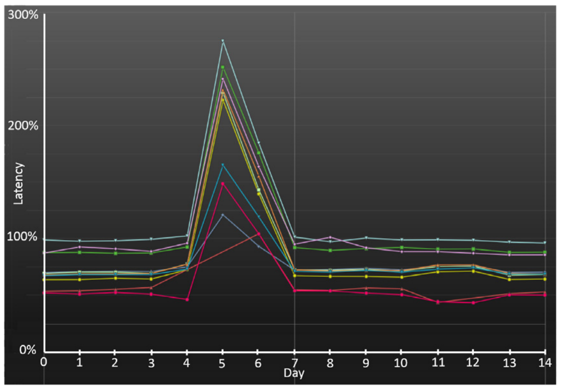

Figure 7: Latency vs Time for several ISPs. A peering link change on Day 5 resulted in a substantial increase in latency. Part-way through Day 6 the link was restored resulting in the expected performance returning.

were being sub-optimally routed (and prioritizing them based on user populations), our ISP peering team could prioritize their efforts to best satisfy our users (improving the performance of users accessing all applications of the Microsoft network - not just those running through FastRoute).

4.2 Identifying Changes in Performance

The above ‘map’ based view of performance is highly beneficial for analyzing a snapshot of performance. However, it is not as beneficial when trying to identify performance changes. Our goal is also to continually improve the performance for all our users. By considering the current performance of users as a benchmark, we can identify performance degradations, correlate the changes with known network changes and revert them if necessary. For example, in Figure 7 we see several ISPs’ latency dramatically increase in the middle of the time series. This was as a result of an alteration in our peering relationship with another ISP resulting in congestion. By identifying an anomaly in the expected performance of users from this ISP, we were able to quickly rectify the issue, ensuring the effect on our users was minimized.

Conversely, the addition of new peering relationships and their impact on user performance was directly attributable providing business justification for the (possible) additional cost.

In a similar way, we can identify black-holing (or hijacking) of traffic (e.g. [10]). By monitoring the current user traffic volumes, we can identify anomalies in the expected volumes of traffic from particular ISPs.

4.3 Active Monitoring

Passive analysis of users reaching our service provide the best aggregate view of the performance our users are receiving. However, active probing mechanisms from third-party sources (e.g., [3, 2, 5]) provides additional sources of data. We found that utilizing systems that existed outside of our own infrastructure avoided circular dependencies and enables us to have information that is normally unavailable using passive monitoring (e.g. traceroutes).

5 FastRoute in Production

FastRoute was designed to replace a third-party CDN that was currently in operation for our Internet applications. However, in order to do so, we had to prove that FastRoute was not only functional, but there was a performance improvement and no drop in availability when compared to the third-party CDN. We describe in Section 5.1 how we compared the two systems, presenting data from our initial comparison.

A critical component of FastRoute is its ability to handle an overloaded proxy. This is expected to be a rare scenario given appropriate capacity planning, but prevents availability drops under load. In Section 5.2 we examine how load manager has operated in production.

A concern when using anycast is the availabilty of anycast in comparison to unicast given route flaps. In Section 5.3 we see no difference in the availability of a third-party unicast based CDN and our anycast solution.

5.1 Onboarding to FastRoute

We took a two step process for ensuring we reliably onboarded our first application onto FastRoute: first, compare availability and performance of non-production traffic served through FastRoute vs our existing third-party CDN, before gradually increasing the fraction of production traffic that was directed to FastRoute instead of the third-party CDN — ensuring real-user performance and availability was not degraded throughout the transition.

5.1.1 Non-Production Traffic Comparison

One method of comparison for two CDNs is through the use of active monitoring probes from agents spread throughout the Internet [2, 3, 5]. However, active probes come from a very limited set of locations and do not reflect the network location of our users. Consequently, we utilized our existing user base as our probing set. We achieved this by placing a small image on both the third party CDN as well as FastRoute. We then directed a small random sampling ($\sim 5\%$) of users to download the image from both the CDN and from FastRoute (after their page had loaded) and report the time taken (utilizing the javascript as described in [16]).

This demonstrated that FastRoute frequently delivered the image faster than our third-party CDN. This was sufficient justification to initiate the delivery of the actual application through FastRoute.

5.1.2 Production Traffic Comparison

The above non-production traffic experiment indicated that performance improvements were possible using FastRoute, however, there are many differences between a small image download and our production application. Consequently, we were cautious when moving to FastRoute. Our first production traffic moved onto FastRoute was a small percentage of a single US-based ISP. We configured our DNS servers to direct a random small fraction of users from the ISP's known LDNS IPs to FastRoute (leaving the remainder on the third-party CDN).

By analysing the performance data for the random sampling of users and comparing with the third-party CDN, we were able to ascertain the performance difference between the two CDNs[5]. This also enabled us to gather confidence that we were functionally equivalent to the third-party CDN. We repeated this "flighting" of different sets of users at different times and for different durations. We see in Figure 8 that for 10 major ISPs contributing more than 60% of traffic within the United States, all experienced a performance improvement with FastRoute. This initial comparison was undertaken with our initial deployment of only 10 FastRoute nodes throughout the United States[6]. This data was sufficient to justify increasing the percentage of users directed to FastRoute until 100% of users now pass through FastRoute. Since the time of analysis, we have increased the number of FastRoute nodes, added new applications and and improved our network connectivity to ISPs to further improve user performance.

5.2 Load Management in Production

We designed FastRoute's load manager with a single configurable parameter for each application — the threshold that a metric must be kept under (see Section 3). This metric is collected periodically and Load Manager reacts based on the current and previous values of the metric. We see in Figure 9 the traffic patterns of one application running on FastRoute. This application has a particularly spiky metric that had a threshold set to 70%.

[5]Note that the performance improvement shown is for the entire FastRoute system (user to proxy to data center) not just for proxy selection.

[6]Nodes throughout the world were present, but for this analysis we focus on the United States.

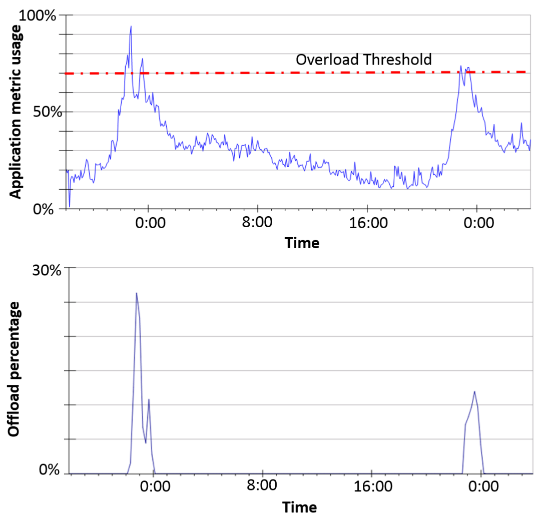

Figure 9: One application running on FastRoute had a very spiky traffic pattern within its diurnal. Load manager reacted automatically to divert the appropriate amount of traffic when load crossed the threshold, bringing it back when it had subsided sufficiently.

If the metric went above a hard limit of 100%, it would result in the loss of user traffic. We can see that the spiky nature of the burst in traffic resulted in the load manager offloading traffic quickly to bring the load back under the threshold. Some oscillation occurs around the threshold due to the delayed effects of DNS TTLs, but we control the traffic around the threshold

FastRoute's load management has been in operation for over 2 years. During this time we have seen a number of scenarios resulting in overloaded proxies (usually of the order of few incidents per week) including nearby proxies going down, naturally spiky user traffic patterns and code bugs in the proxy or DNS. FastRoute's load management scheme has provided the required safety net to handle all scenarios during this time without requiring manual intervention to modify routing policies or alter DNS configurations.

5.3 Anycast Availability

A concern when utilizing an anycast based solution is that the availability of the endpoint will be lower due to route fluctuations. In Figure 10 we show results from a synthetic test where approximately $20,000$ Bing toolbar clients downloaded a small image from an anycast IP announced from all 12 nodes (full set of nodes at time of experiment) throughout the Internet and the same image from a 3rd party (unicast) CDN over a period of a week. Although one datapoint showed the anycast availability

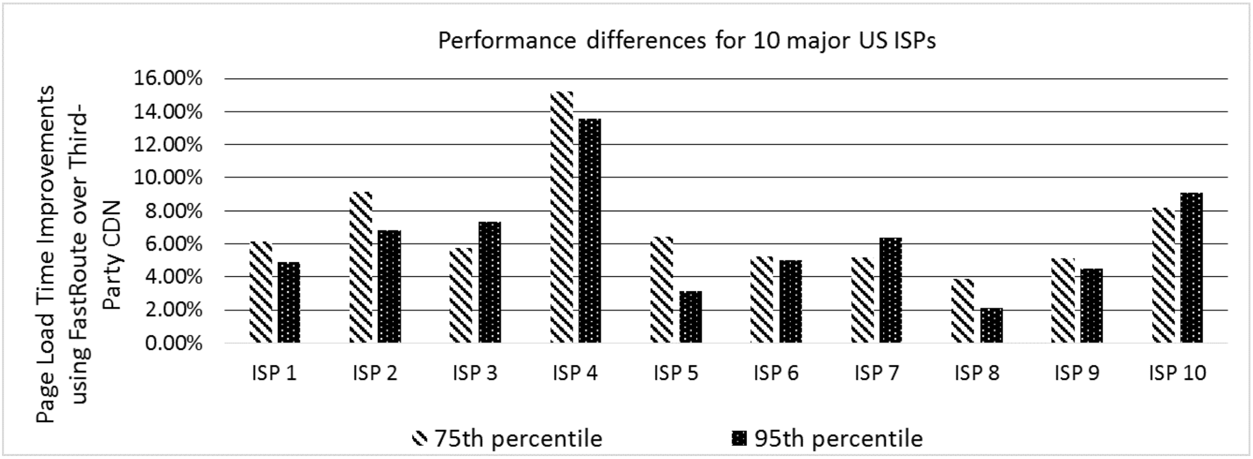

Figure 8: Performance improvements in 10 major US ISPs (contributing above 60% of all user traffic in the US) when using FastRoute compared to a third-party CDN. This data was collected when only 10 FastRoute nodes were in operation and no nodes were overloaded.

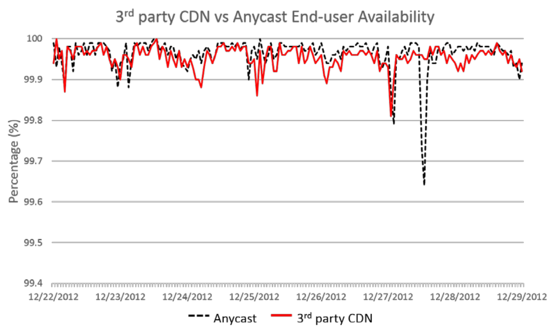

Figure 10: Anycast availability over a week compared to third-party CDN. Note the y-axis starts at 99.4%. The availabilty over the entire week was 99.96% vs 99.95% respectively.

dropped to 99.65% availabilty, we saw overall availabilities of 99.96% and 99.95% for anycast and third-party CDN availabilities respectively. These results, the success of other anycast TCP based CDNs (e.g. Cloudflare, Edgecast), the work presented in [22] and the lack of issues found in over 2 years serving production traffic (even as the set of nodes grows) indicate that anycast in the Internet is stable enough to run a production network on.

6 Future Work

Within this paper we have described the architecture of FastRoute, concentrating on proxy selection mechanism. Future work includes:

- Examining the selection of data center for user traffic landing on a proxy as well as techniques used to prioritize and multiplex user traffic to achieve optimal performance.
- Analyzing the impact to the self-correlation of DNS and proxy traffic when supporting IPv6.
- Analyzing the impact that local decisions made when diverting load, have on the global traffic patterns. In particular, we would like to understand the degree of sub-optimality introduced due to making local decisions, compared to making globally optimal decisions centrally.
- Studying the distributed load management algorithm from a control-theoretic perspective, and understand limits on correlation and user-traffic for stable system operation.

7 Conclusion

We have presented FastRoute, an anycast routing architecture for CDNs that is operational and provides high performance for users. FastRoute's architecture is robust, simple to deploy and operate, scalable, and just complex enough to handle overload conditions due to anycast routing. We highlighted performance gains obtained from our production system when routing users through FastRoute instead of a major third-party CDN.

We described a novel load management technique in FastRoute, which used the anycast DNS and multiple anycast proxy rings for load absorption. Excess traffic from one layer was directed to another higher layer using the collocated DNS. We provided data from our production system which showed that the correlation between DNS and corresponding user queries landing on the same node in our network to be sufficiently high, and relatively stable over time, both of which are crucial for effective load management. FastRoute load management has protected our production system numerous times from having an overload-induced outage, in addition to saving precious operator hours that would've otherwise been needed in a manual system. We provided one such example from our production system where proxy overload was dealt with quickly and effectively by FastRoute's load management.

Overall, FastRoute was designed with high performance, reliability and ease of operations in mind. By not over-complicating the design to handle rare scenarios — and trading off performance for simplicity to handle such rare scenarios — we were able to quickly adapt to new requirements with minimal development effort. We believe this is the biggest learning from the design, development, deployment and operation of FastRoute.

8 Acknowledgments

Many people were involved in the design of FastRoute. The authors would like to acknowledge the following people for their contribution to FastRoute: Dongluo Chen, Chao Zhang, Arun Navasivasakthivelsamy, Sergey Puchin, Bin Pang, Saurabh Mahajan, Darren Shakib, Kyle Peltonan, Mohit Suley, Ernie Chen, Aaron Heady, Brian Jack, Jeff Cohen, Randy Kern, Matthew Calder, Andrii Vasylevskyi, Narasimhan Venkataramaiah, Ashwin Sarin, Peter Oosterhof, Andy Lientz, Mark Kasten, Kevin Epperson, Christian Nielson, Dan Eckert.

We would also like to thank our shepherd Ethan Katz-Bassett.

References

[1] Amazon aws autoscaling. `http://aws.amazon.com/autoscaling/`.

[2] Gomez networks. `www.gomeznetworks.com`.

[3] Keynote. `www.keynote.com`.

[4] Neustar ip intelligence. `http://www.neustar.biz/enterprise/ip-intelligence`.

[5] Thousandeyes. `www.thousandeyes.com`.

[6] W3C navigation timing. `http://www.w3.org/TR/navigation-timing/`.

[7] V. K. Adhikari, Y. Guo, F. Hao, V. Hilt, and Z. Zhang. A tale of three cdns: An active measurement study of hulu and its cdns. *Global Internet Symposium*, 2012.

[8] H. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. V. der Merwe. A practical architecture for an anycast cdn. *ACM Transactions on the Web*, 2011.

[9] A. Barbir, B. Cain, R. Nair, and O. Spatscheck. RFC3568: Known content network request-routing mechanisms, July 2003.

[10] M. Brown. Renesys blog: Pakistan hijacks youtube, 2008. `http://velocityconf.com/velocity2009/public/schedule/detail/8523`.

[11] J. Brutlag. Speed matters. `http://googleresearch.blogspot.com/2009/06/speed-matters.html`.

[12] C. Bueno. Doppler: Internet radar, July 2011. `https://www.facebook.com/notes/carlos-bueno/doppler-internet-radar/10150212498738920`.

[13] M. Calder, X. Fan, Z. Hu, R. Govindan, J. Heidemann, and E. Katz-Bassett. Mapping the expansion of google's serving infrastructure. *IMC*, 2013.

[14] C. Contavalli, W. van der Gaast, S. Leach, and D. Rodden. Client subnet in dns requests. `http://tools.ietf.org/html/draft-vandergaast-edns-client-subnet-02`.

[15] A. Flavel, A. Karasaridis, and J. Miros. System and method for content delivery using dynamic region assignment. US Patent #20120290693, 2011.

[16] A. Flavel, P. Mani, and D. Maltz. Re-evaluating the responsiveness of dns-based network control. *IEEE LANMAN*, 2014.

[17] R. Gao, C. Dovrolis, and E. Zegura. Interdomain ingress traffic engineering through optimized as-path prepending. *Networking*, 2005.

[18] C. Huang, I. Batanov, and J. Li. A Practical Solution to the client-LDNS mismatch problem. *ACM SIGCOMM CCR*, 2012.

[19] R. U. Ibm and D. Rosu. An evaluation of tcp splice benefits in web proxy server. *WWW*, 2002.

[20] R. Krishnan, H. Madhyastha, S. Srinivasan, S. Jain, A. Krishnamurthy, T. Anderson, and J. Gao. Moving beyond end-to-end path information to optimize cdn performance. *IMC*, 2009.

[21] F. Lardinois. Microsoft adds auto scaling to windows azure, June 2013. http://techcrunch.com/2013/06/27/microsoft-adds-auto-scaling-to-windows-azure/.

[22] M. Levine, B. Lyon, and T. Underwood. TCP Anycast - Don't believe the FUD. *Proceedings of NANOG*, 2006.

[23] P. Mockapetris. Domain names - implementation and specification. *RFC1035*, 1987.

[24] R. Mukhtar and Z. Rosberg. A client side measurement scheme for request routing in virtual open content delivery networks. *Performance, Computing and Communications Conference*, 2003.

[25] E. Nygren, R. Sitaraman, and J. Sun. The akamai network: a platform for high performance internet applications. *ACM SIGOPS OSR*, 2010.

[26] J. Otto, M. Sanchez, J. Rula, and F. Bustamante. Content delivery and the natural evolution of dns: remote dns trends, performance issues and alternative solutions. *IMC*, 2012.

[27] A. Pathak, Y. A. Wang, C. Huang, A. Greenberg, Y. C. Hu, R. Kern, J. Li, and K. Ross. Measuring and evaluating tcp splitting for cloud services. *PAM*, 2009.

[28] M. Prince. A brief primer on anycast, 2011. `https://blog.cloudflare.com/a-brief-anycast-primer/`.

[29] B. Quoitin, S. Uhlig, C. Pelsser, C. Pelsser, L. Swinnen, and O. Bonaventure. Interdomain traffic engineering with bgp. *IEEE Communications Magazine*, 41, 2003.

[30] Y. Rekhter, T. Li, and S. Hares. RFC4271: A border gateway protocol 4, January 2006.

[31] E. Schurman and J. Brutlag. Performance related changes and their user impact. `http://velocityconf.com/velocity2009/public/schedule/detail/8523`.