Efficiently Delivering Online Services over Integrated Infrastructure

Technical Report

Hongqiang Harry Liu Raajay Viswanathan Matt Calder
Aditya Akella Ratul Mahajan Jitendra Padhye Ming Zhang

Microsoft University of Wisconsin

Abstract— We present Footprint, a system for delivering online services in the increasingly common "integrated" setting, where the same provider operates multiple elements of the infrastructure (e.g., proxies, data centers, and a wide area network). Such integration can boost system efficiency and performance by finely modulating how traffic enters and traverses the infrastructure. We show how such modulation can be practically realized using a model that faithfully captures the complex, time-varying dynamics of service workload. For instance, when a group of users are directed to a new proxy, their ongoing sessions continue to arrive at the old proxy, and this load at the old proxy declines gradually. Footprint is currently being deployed in Microsoft's infrastructure. Our data-driven simulations show that, compared to the current method, it can carry at least 50% more traffic and reduce user delays by at least 38%.

1 Introduction

The emergence of cloud computing is reshaping how online services and content are delivered. Historically, the three types of infrastructure required for service delivery—*i*) data centers (DCs) that host application logic and state; *ii*) edge proxies that terminate TCP or HTTP connections and cache content close to users; *iii*) wide area networks (WAN) that connect DCs and proxies—were owned and operated by different organizations (Figure 1a). But now, large cloud providers such as Amazon, Google, and Microsoft operate all three types of infrastructures for their own and their customers' services [6, 7, 9] ((Figure 1b). Infrastructure integration is also ongoing for massively-popular service providers such as Facebook as they leverage their scale to amortize infrastructure cost [8], and for large ISPs as they begin offering content distribution services [10].

Infrastructure integration allows one to take a holistic view of the system, to improve both performance and efficiency. For instance, the state of the WAN (e.g. residual capacity of proxy-to-DC path) can be factored into deciding which proxies serve which users.

But to our knowledge, current systems for delivering online services do not take such a holistic view. Many such systems were designed for the traditional setting [13, 4, 11, 27]. Even those that operate in integrated settings fail to leverage its unique opportunities. For example, currently, in Microsoft's network [14], WAN traffic engineering (TE) operates independently, with no advance knowledge of load placed on it by edge proxies and has no ability to steer load to a different proxy or DC to relieve hotspots. At the same time, the edge proxies have no knowledge of WAN TE. When they select DCs for a user session, they need to know the quality of the WAN paths to different DCs. They do so by probing the paths, which is akin to looking in the rear view mirror: it can detect congestion only *after* it occurs and cannot guarantee congestion-freedom when load is moved.

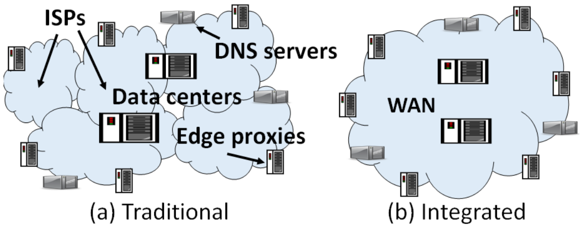


Figure 1: Online Service Delivery infrastructure.

This paper describes the design of the next generation of Microsoft's online service delivery infrastructure, called Footprint. Using an SDN-like central controller model, Footprint jointly coordinates all key routing and resource allocation decisions, to achieve desired objectives. It decides how to map users to proxies, proxies to DC(s), and traffic to WAN paths, and configures all infrastructure components involved in service delivery, including network switches, proxies, load balancers, and DNS servers to achieve this mapping.

While it is not surprising that coordination among system components (e.g., joint optimization of WAN TE and proxy load management) can help, we show that fully realizing the potential of infrastructure integration requires faithful modeling of system dynamics. A major issue is that after we change system configuration, its impact is not immediate but manifests only gradually.

The reason is that ongoing user sessions will continue to use the proxy that they picked at session start. Thus, when the controller changes the proxy (or the DC) mapping for a group of users, traffic from those users will not move immediately. Instead, the load on the second proxy (or the second DC) will increase as new sessions arrive and that on the first proxy (or DC) will decrease as old sessions depart. The system model and control algorithms must correctly account for this lag. Traditional network TE controllers such as SWAN [18] and B4 [19] do not have to deal with this lag, since they operate at packet granularity, and the impact of a configuration change is immediate.

In this paper, using data from Microsoft's service-delivery network, we illustrate the modeling challenge, and devise techniques to address it. To capture temporal variations, we model system load and performance as a function of time. Solving time-based models can be intractable (e.g., time is continuous), but we show how all load and performance constraints can be met by considering a small number of time points.

In addition to the modeling challenge, we address a number of practical issues to design a scalable and robust system. For example, we need to estimate the latency to various edge proxies from different user groups in a scalable manner. We will discuss these issues, and our solutions for them in more detail later in the paper.

We implement our model and other techniques in an SDN-style controller to build Footprint. Our prototype is deployed fully in a modest-sized testbed and it monitoring aspects are deployed in Microsoft's service-delivery infrastructure. We evaluate Footprint using these deployments and trace-driven simulations. We find that it enables the infrastructure to carry at least 50% more traffic, compared to Microsoft's current method that does not coordinate the selection of proxies, DCs, and WAN paths. At the same, it improves user performance by at least 38%. We also show that Footprint's system model is key to achieving these gains.

The basic issue tackled by our model—gradual impact of configuration changes—arises commonly in many large-scale systems. For instance, session-based load balancers and middleboxes have the same problem. We believe that our model is flexible and can be adapted to improve the efficiency of these other systems as well, which is an increasingly important concern as the scale of computing infrastructures grows.

2 Background and Motivation

Figure 1 shows a high-level view of online service delivery infrastructure. Data centers, which usually number $O(10)$, host application logic and hard state. Users connect to DCs via edge proxies. The proxies help boost performance by terminating TCP and HTTP connections (coming over possibly lossy last mile paths) close to the user and by caching some content (so it does not need to be fetched from a distant DC).

In the traditional architecture, the data centers, the edge proxies and the WAN that connects them are operated by different entities. For example, the data centers may be owned by a hosting service, the edge proxies may be owned by a company like Akamai and various ISPs may provide connectivity between the data centers, and to the edge proxies. As discussed earlier, large cloud providers like Microsoft and Google, are moving towards a more integrated architecture, wherein a single entity owns and operates the data centers, the WAN connecting the data centers, and the edge proxies.

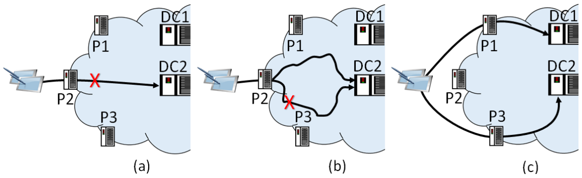


Figure 2: Spatial modulation via joint coordination

Regardless of the architecture, any online service delivery system makes three decisions for user requests: (*i*) selecting the proxy for the request (*ii*) selecting the DC(s) for user sessions at a proxy, and (*iii*) selecting WAN path(s) for traffic between proxies and DCs.

In the traditional setting, the three decisions are made largely independently of one another, and typically without the benefit of global knowledge. A third-party like Akamai makes a decision about which proxy the user selects, and which data center the request will be served from. Various ISP routing policies decide how the traffic flows between the data centers and the proxies.

Even in an integrated OSP, these decisions are often made independently. For example, in Microsoft's FastRoute system, anycast routing is used to direct clients to nearby proxies [14]. The proxies independently decide which data centers to route the request to, and the WAN TE is performed independently as well.

In this paper, we argue for making service-delivery decisions jointly. Joint decisions can significantly improve efficiency, not only because of global information, but also by offering new knobs that were previously unavailable. For example, consider Figure 2, where congestion appears between P2 and DC2. In the traditional setting, WAN TE cannot change how traffic enters and exits the network as that is determined by proxy and DC selection. To relieve congestion, it must rearrange how traffic flows within the network. Sometimes that may not sufficient, however (Figure 2b). Joint decisions can "spatially modulate" the traffic (i.e., change where it enters or exits the WAN) by simultaneously controlling the proxy and DC selection. As shown in Figure 2c, such spatial modulation can help relieve congestion.

Spatial modulation is especially helpful when a large fraction of WAN traffic is for user-facing services. This situation holds for large cloud providers; they have a separate WAN for non-user-facing traffic [18, 19]) To

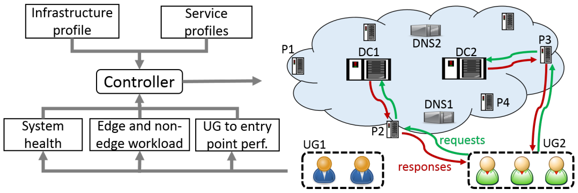


Figure 3: Overview of Footprint.

evaluate the benefit of spatial modulation in practice, we analyze traces from Microsoft's infrastructure, which runs WAN TE and service delivery controller independently [14]. We identified congestion events in the WAN as those where the utilization of at least one link is over 80% during a 5 minute window. We find that all of these events could be resolved using spatial modulation of service traffic. We also repeated the study by artificially scaling traffic by 50%: the number of congestion events went up by 1200% (because our WAN is heavily utilized), but all of them could still be resolved. This advantage of spatial modulation underlies the efficiency and performance improvements of Footprint (§8).

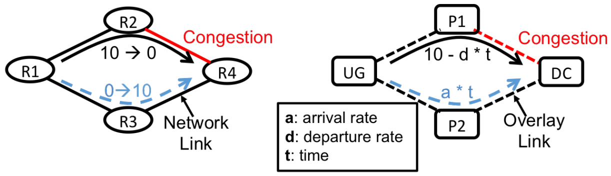
While joint decisions can help, we will see that accurate modeling of system dynamics is necessary to realize its benefits. Next, we provide an overview of the Footprint architecture, and outline key challenges.

3 Overview of Design and Challenges

Figure 3 shows an overview Footprint. The controller is bootstrapped with infrastructure and service profiles. Infrastructure profile describes the topology, capacity in terms of multiple resources (e.g., CPU, memory, bandwidth), and latency of each component. A service's profile describes which proxies and DCs host it—not all services may be hosted everywhere—and any constraints on mapping users to proxies and DCs (e.g., Chinese users must be served from China). When running, the controller gets up-to-date information on system health, workload, and user-to-proxy delays. Periodically, or after a failure, the controller computes and deploys new system configuration based on this information. This configuration determines, for the next period, how user requests map to proxies, which DCs are used by proxies, and which WAN paths are used.

Our design must address three categories of challenges: obtaining the necessary inputs, computing the desired configuration, and implementing the computed configuration. We provide a brief overview of these challenges in this section. Future sections provide more detail, with a focus on the system model.

Obtaining dynamic inputs: In addition to static inputs such as WAN topology, the controller needs up-to-date information about user-to-proxy delays, the load on the system (i.e. load on WAN links, data center and proxy utilization etc.), and information about system health (e.g. which links or proxies have failed etc.). We have scalable infrastructure in place to collect the needed information about WAN and proxy load and health [28].



(a) Instantaneous configuration change in traditional TE.

(b) Gradual configuration change in session-level overlay routing.

Figure 4: Session affinity results in gradual load changes in session routing on top of server overlays.

A key challenge lies in scalably collecting information about user-to-proxy delays. We address it in two ways. First, we group users into groups–a user group (UG) is a set of users that are expected to have similar relative latencies to edge proxies (e.g., because they are proximate in Internet topology). Second, we measure delays between UGs and proxies in a probabilistic manner. §6 describes these aspects in more detail.

Computing the configuration: We initially believed that we could compute system configurations using a linear program (LP) similar to TE controllers such as SWAN [18]. However, we realized that Footprint controller faces qualitatively different problems. The key issue is that service sessions last longer than individual packets and these sessions stick to their originally chosen proxies and DCs during their lifetime.

More specifically, online services rely on DNS to direct different users to different proxies–IP addresses of the desired proxies are returned when the user looks up the name of the service. The mapping of name to IP addresses is changed to move load from one proxy to another. The problem is that DNS changes cannot change traffic distribution instantaneously. In addition to DNS mappings being cached at the LDNS servers for the TTL duration, there are two other problems. First, DNS mappings may be cached at the client well beyond the TTL value (e.g., many browsers will not look up the same hostname again within a tab, as long as the tab is open). Second, persistent TCP connections used by HTTP 1.1 (and 2.0), can last well beyond DNS TTL as well.

This caching means that even after the Footprint controller updates DNS mapping to point a group of users to a new proxy, the traffic from ongoing user sessions from that UG continues to arrive at the old proxy. Additionally, the proxy must continue to send traffic from ongoing sessions to the same DC. Otherwise, those sessions may be abruptly terminated whenever system configuration is changed (e.g., every 5 minutes).

Session stickiness makes it harder to compute robust system configurations compared to traditional TE. For instance, in Figure 4(a), traffic from R1 to R4, is ini-

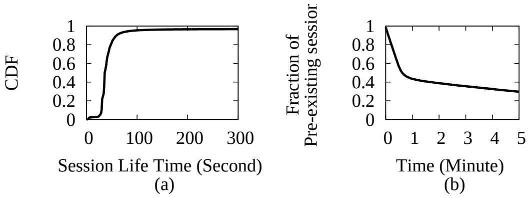


Figure 5: Session life time.

tially routed via R2. When the link R2-R4 is congested, TE controller configures R1 to forward the traffic via R3. This change is near instantaneous, and more importantly, largely transparent to the applications. However, the Footprint controller does not have this luxury. Figure 4(b) shows an example. Initially, a group of users (UG) use proxy P1 to access the service S hosted in the data center (DC). When the path P1-DC is congested, we need to reroute the traffic via P2. This can be done by changing the changing the DNS mapping; i.e. the name for service S resolves to the IP address of proxy P2. However, this change only affects new user sessions, and traffic from old sessions continues to arrive at P1,

The severity of the problem is illustrated in Figure 5. Figure 5(a) shows the CDF of lifetime of TCP connections at our proxies. We see that about 5% of the connections last longer than 100 seconds. In our current implementation, Footprint adjusts DNS mappings every 5 minutes. Since new HTTP sessions arrive roughly uniformly in a five minute interval, a large fraction TCP connections continue to send data to the "old" proxy after the mapping is updated. Figure 5(b) shows that the number of sessions that are still active as a function of time. Even if the DNS mapping is changed at the end of the 5 minute period, over 20% of the sessions will continue to send data to the previous proxy. The previous proxy must continue to handle this "old" traffic, and send it to the same DC as before.

We deal with this challenge by incorporating session lifetime and workload migration dynamics into our system model, as described in the next section.

The reader may wonder why we simply do not kill the old connections—that would obviate the need for modeling temporal behavior. However, as shown above, a large number of "old" connections are active on each epoch boundary. It is unacceptable to kill so many connections every five minutes. We may alleviate the problem by updating system configuration less frequently (e.g., an hour); but we need our system to be responsive and react quickly to demand bursts (e.g., flash crowds) and faster updates lead to greater efficiency, as shown earlier for WAN TE [18]. We may also alleviate the problem by updating the client code gracefully handle changes to proxy mappings; but this alternative is not available to us because Footprint must accommodate a large number of already-deployed (legacy) applications.

Implementing the computed configuration: The computed configuration is implemented by updating DNS mappings, proxy-to-DC mappings, and weights on WAN paths. One issue that we face here is that changing UG-to-proxy mappings (e.g., in response to WAN congestion) can force user traffic onto paths with unknown capacities. While we monitor UG-to-proxy path delays, we are not reliably aware of path capacities. We thus prefer that UGs continue to use current paths to the extent possible. To ensure this, Footprint uses load balancers at network edge that can forward user requests to remote proxies. These load balancers allow us to decouple how users reach our infrastructure and how their traffic flows internally. We omit details of this mechanism (and its modeling) due to lack of space.

4 System Model

The Footprint controller periodically decides how requests and responses from each UG are going to traverse the infrastructure. For example, in Figure 3, suppose it makes a fraction of sessions from UG2 go through edge proxy P2 and data center DC1, it also simultaneously computes how to route request traffic in network from P2 to DC1 and response traffic from DC1 to P2.

The controller computes how to assign *new* sessions from each UG to a collection of end-to-end paths (or e2e-paths) which includes two "servers"—an edge proxy y and datacenter c—and two WAN paths—request and response paths between y and c.

Each network path is a pre-configured tunnel, i.e., a series of links from the source to destination switch; there are usually multiple tunnels between a source-destination pair. Once a session is assigned to an e2e-path, it will stick to the assigned proxy and data center; the WAN paths taken by the traffic may change.

The assignments from sessions to e2e-path impacts system efficiency as well as the latency experienced by users. The controller must enable the infrastructure to accommodate as much workload as possible, while ensuring that the proxies, the data centers and the network paths are not overloaded and that the traffic prefers low-latency paths. The key to meeting these goals is to model the load on resources with high fidelity, which we do as described below.

4.1 Preliminaries

Table 1 shows key notations in our model, including its inputs and outputs. The outputs contain routing decisions for two types of traffic. The first type is *unsettled edge traffic* due to new user sessions for services hosted on the edge proxies. Here, the routing decision $\mathbf{w}_{\theta,g}$ denotes the fraction of new sessions from UG g assigned to e2e-path θ. The second type is *settled traffic*, due to existing edge

Inputs	
g	A user group (UG)
a_g^j	Session arrival rate of g in jth epoch
$q(t)$	CDF of session lifetime
θ	An e2e-path
Θ_g	E2e-paths that can selected by UG g
e	A "server" on an e2e path: i.e. an edge proxy or a datacenter
p, l	p: network path; l: a network link
bw_l	Bandwidth of link l
$P_{s,d}$	All network paths that starts from server s and ends with server d
$\xi'_{s,d}$	Non-edge traffic demand from s to d
$h_{\theta,g}$	Latency experienced by g when going through θ
α	A resource (e.g. CPU, memory etc.) at an edge proxy or a datacenter
$M_{\alpha,e}$	Capacity of resource α at (e)
c_{req}, c_{rsp}	Bandwidth consumption of a request, response
T	Length of an epoch
Intermediate variables	
$\mu_{\alpha,z}(t)$	Resource α's utilization on z
$n_{\theta,g}(t)$	Number of sessions on θ from g
$n_{e,g}(t)$	Number of sessions on e from g: $\sum_{\forall\theta:e\in\theta} n_{\theta,g}(t)$
$a_{\theta,g}(t)$	Session arrival rate on θ from g
$a_{e,g}(t)$	Session arrival rate on e from g: $\sum_{\forall\theta:e\in\theta} a_{\theta,g}(t)$
$\xi_{s,d}(t)$	Traffic of settled sessions s to d
$f(t)$	CCDF of session lifetime
Outputs	
$\mathbf{w}_{\theta,g}$	Weight of new sessions of UG g on θ
$\omega_{p,s,d}$	Weight of traffic from s to d on network path p

Table 1: Key notations in Footprint model.

sessions that stick to their servers and all non-edge traffic carried by the WAN. Here, the routing decision $\omega_{p,s,d}$ denotes the fraction of non-edge (i.e., non-service) traffic from source s to destination d assigned to network path p. Note that s and d represent WAN endpoints connected to datacenters, edge proxies, or neighboring ISPs. For instance, for non-edge traffic s and d may be a neighboring ISP and a datacenter; for service request traffic generated from UGs, s is the proxy, while d is the datacenter.

Constraints: Because the sessions from g can only be assigned to a subset of e2e-paths Θ_g whose proxies are close enough to g, and similarly traffic from s to d can only traverse a subset of network paths $P_{s,d}$ that connect s and d, we have the following constraints on routing:

$$\forall g: \quad \textstyle\sum_{\forall\theta} \mathbf{w}_{\theta,g} = 1, \text{ if } \theta \notin \Theta_g, \text{ then } \mathbf{w}_{\theta,g} = 0 \tag{1}$$

$$\forall s,d: \quad \textstyle\sum_{\forall p} \omega_{p,s,d} = 1, \text{ if } p \notin P_{s,d}, \text{ then } \omega_{p,s,d} = 0 \tag{2}$$

Before describing the system model based on which we compute these routing decisions, we list the assumptions made in our modeling.

Assumptions: We assume that a data center is involved in serving each user request, i.e., requests cannot be fulfilled entirely by the proxy (though the proxy may have some local logic and cached data). All requests for our services require personalized responses based on hard state that is only maintained in DCs.

We assume that the session arrival rate for a user group g in j-th epoch a_g^j, is known and fixed. In §6, we describe how arrival rate is estimated. We have empirically verified that the arrival rate is fixed during each epoch, as the epoch length that we use (5 minutes) is short. Our model can be extended to account for errors in the estimated arrival rate which will be discussed in §5.2. Similarly, we assume that the distribution of session lifetimes, denoted by $q(t)$, is known.

We model proxies and datacenters as monolithic entities, ignoring their internal structure (and hence we refer to them as "servers"). Without this simplifying assumption, the model will become intractable as there will be too many individual servers.

For ease of exposition, we assume that the infrastructure supports only one type of service. This service generates request-response traffic, and the average bandwidths consumed by requests and responses is known (c_{req}, c_{resp}). We define the capacity $M_{\alpha,e}$ of resource α (e.g., CPU, memory) of server e in terms of number of sessions. That is, we say that the CPU on a proxy can handle a certain number of sessions. We assume that this number is known, and fixed for a given α and a given e. Since links can be viewed as a "server" with a single resource–bandwidth—we will occasionally leverage this view to simplify notation. In §5, we extend our model to multiple services and a more detailed view of resource and bandwidth consumption.

Finally, we assume that the system's objective is to find end-to-end paths that minimize user delays. We do not consider other properties such as throughput or loss rate, though we do model the impact of high utilized resources on delay.

4.2 Temporal system dynamics

To model resource utilization, we first model the number of active sessions consuming that resource. Let z denote any element of an end-to-end path - a "server" or a link. The number of active sessions on z is:

$$n_z(t) = \sum_{\forall g} \sum_{\forall\theta:e\in\theta} n_{\theta,g}(t) \tag{3}$$

$n_{\theta,g}$(t) is the number of active sessions from UG g on e2e-path θ at time t, and thus n_z(t) is the total number of active sessions on element z. $n_{\theta,g}(t)$ evolves with time, as new sessions arrive and old ones depart.

Consider epoch k, which lasts from time $t \in [kT,(k+1)T]$, where T is the epoch length. At the beginning of the epoch ($t = kT$), there are $n_{\theta,g}^{old}$(kT) pre-existing sessions that will terminate per the pattern defined by the distribution of session life time. Simultaneously, new sessions will continuously arrive, some of which terminate inside the current epoch and others will last beyond the epoch. At any given time, the total number of sessions in the epoch is:

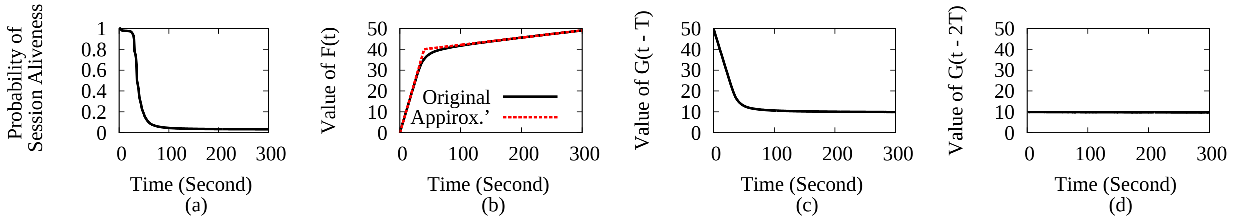


Figure 6: The pattern functions derived from the session life time distribution of Bing in an epoch. The time (x-axis) on each graph is relative to the start of the epoch.

$$n_{\theta,g}(t) = n^{new}_{\theta,g}(t) + n^{old}_{\theta,g}(t) \tag{4}$$

We need to faithfully model how $n^{new}_{\theta,g}(t)$ and $n^{old}_{\theta,g}(t)$ evolve with time to provide achieve desired performance and efficiency.

New sessions: The new session arrival rate on θ from UG g is:

$$\forall\theta, g : a_{\theta,g} = a_g \times \mathbf{w}_{\theta,g} \tag{5}$$

Recall that a_g is the total arrival rate of sessions from UG g, and we assume it to be fixed within an epoch.

At any given time t within the epoch k, $n^{new}_{\theta,g}(t)$ is the sum of the number of sessions which arrived in interval $[kT,\ t]$ and are still alive at t. From the session life time CDF distribution $q(t')$, we can easily derive $f(t') = 1 - q(t')$, which is probability that a session is still alive after duration t' since it started. Figures 5(a) and 6(a) show examples of $q(t')$ and $f(t')$, respectively.

Therefore, at any given time $\tau \in [kT, t]$, the number of new sessions that arrived in the interval $[\tau,\ \tau+\Delta\tau]$ is $a_{\theta,g} \times \Delta\tau$. Among these sessions, there will be $f(t-\tau)a_{\theta,g} \times \Delta\tau$ sessions left at time t. When $\Delta\tau \to 0$:

$$\begin{aligned} n^{new}_{\theta,g}(t) = \int_{kT}^{t} f(t-\tau) \times a_{\theta,g} d\tau = a_{\theta,g} \times \int_{0}^{t-kT} f(\tau) d\tau \\ = a_{\theta,g} \times F(t-kT) \end{aligned} \tag{6}$$

where $F(t) = \int_0^t f(\tau)d\tau$, which represents the number of sessions alive at t assuming unit arrival rate. Figure 6(b) shows $F(t)$, obtained from Figure 6(a).

Pre-existing sessions. At time t in epoch k, the number of pre-existing sessions that arrived in epoch j ($j < k$) is:

$$n^{old,j}_{\theta,g}(t) = \int_{jT}^{(j+1)T} f(t-\tau) \times a^j_{\theta,g} d\tau \tag{7}$$

$$= a^j_{\theta,g} \times \int_{t-(j+1)T}^{t-jT} f(\tau)d\tau = a^j_{\theta,g} \times G(t-jT)$$

where $a^j_{\theta,g}$ is the observed arrival rate in epoch j and $G(t) = F(t) - F(t-T)$. Therefore, the total number of pre-existing sessions is:

$$n^{old}_{\theta,g}(t) = \sum_{j=0}^{k-1} a^j_{\theta,g} \times G(t-jT) \tag{8}$$

Figures 6(c) and (d) show examples of $G(t-jT)$ in two epochs prior to current one, i.e., $j = (k-1)T$ and $j = (k-2)T$, respectively when $T = 300$ seconds.

Server utilization: Given the number of active sessions, the utilization of resource α on server e is:

$$\mu_{\alpha,e}(t) = \frac{n_e(t)}{M_{\alpha,e}}, \tag{9}$$

Combining Eqns. 3, 4, 6, 8, and 9, the utilization of a resource α on server e is:

$$\mu_{\alpha,e}(t) = \frac{F(t-kT) \times a_{e,g} + \sum_{j=0}^{k-1} G(t-jT) \times a^j_{e,g}}{M_{\alpha,e}} \tag{10}$$

Link utilization: To model link utilization, we account for non-edge traffic and the fact that requests and responses consume different amounts of bandwidth, c_{req} and c_{rsp}, respectively.

An e2e-path θ contains a request path θ_{req} from UG to DC and a response path θ_{rsp} from DC to UG. Thus, the total edge traffic load generated by new sessions on a network link l is:

$$\mu'_{bw,l}(t) = \frac{\sum_{\forall\theta:l\in\theta_{req}} n^{new}_{\theta}(t) c_{req} + \sum_{\forall\theta:l\in\theta_{rsp}} n^{new}_{\theta}(t) c_{rsp}}{bw_l} \tag{11}$$

Pre-existing sessions stick to their originally assigned servers, but the WAN paths they use can be adjusted. All such sessions from site s to site d generates traffic demand $\xi_{s,d}$:

$$\xi_{s,d}(t) = \sum_{\forall g}[\sum_{\forall\theta:s,d\in\theta_{req}} n^{old}_{\theta,g}(t) c_{req} + \sum_{\forall\theta:s,d\in\theta_{rsp}} n^{old}_{\theta,g}(t) c_{rsp}] \tag{12}$$

Network links are shared by edge and non-edge traffic. Let $\xi'_{s,d}$ be the traffic demand from source router s to destination router d, the link load by non-edge traffic on link l is:

$$\mu''_{bw,l}(t) = \frac{\sum_{\forall s,d} \sum_{\forall p:l\in p} [\xi_{s,d}(t) + \xi'_{s,d}] \times \omega_{p,s,d}}{bw_l} \tag{13}$$

Thus, the total utilization of network link l should be:

$$\mu_{bw,l}(t) = \mu'_{bw,l}(t) + \mu''_{bw,l}(t) \tag{14}$$

4.3 Optimization objective

Equations 10 and 14 model the impact of routing decisions on resource utilization. For computing the final decisions, resource utilization is only half of the story.

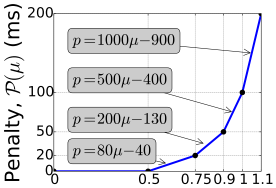


Figure 7: Penalty function.

Our goal is not to exclusively minimize utilization, as that can come at the cost of poor performance if user sessions start traversing long paths. Similarly, the goal is not to exclusively select shortest paths, as that may cause overly high utilization that induces delays for users.

To balance the two concerns, as is common in TE systems, we penalize high utilization in proportion to expected delay it imposes [1]. Figure 7 shows the piece-wise linear approximation of the penalty function $\mathtt{P}(\mu_{\alpha,e})$ we use. The results are not sensitive to the exact shape—which can differ across resource types—as long as the function has monotonically non-decreasing slope.

Thus, our objective function is:

$$\text{min.} \sum_{\forall \alpha,e} \int_0^T \mathtt{P}(\mu_{\alpha,e}(t))dt + \sum_{\forall g,\theta} \int_0^T h_{g,\theta} n_{g,\theta}(t)dt \tag{15}$$

The first term integrates utilization penalty over the epoch, and the second term captures path delay. The variable $h_{g,\theta}$ represents the total latency when sessions of UG g traverse e2e-path θ. It is the sum of UG-to-proxy and WAN path delays.

4.4 Solving the model

Minimizing the objective under the constraints above will assign values to our output variables. However, our model uses continuous time and we must ensure that the objective is limited at all possible times. To tractably guarantee that, we make two observations. First, $n^{new}_{\theta,g}(t)$ monotonically increases with time and is also concave. The concavity is valid if only $\frac{dF(t)}{dt} = f(t)$ is monotonically non-increasing with t, which is always true because $f(t)$ is a CCDF (complementary cumulative distribution function). Hence, we can use a piecewise linear concave function $F'(t)$ that closely upper-bounds $F(t)$. For instance, the red, dashed line in Figure 6b shows a two-segment $F'(t)$ we use for Bing.

The second observation is that $n^{old}_{\theta,g}(t)$ is monotonically decreasing and convex, e.g. Figure 5(b). The convexity depends on both the shape of $f(t)$ and the length of epoch T we choose. We found the convexity of $n^{old}_{\theta,g}(t)$ is valid for all the services in our infrastructure. So, in this paper, we assume for simplicity that $n^{old}_{\theta,g}(t)$ is always convex. [1]

Therefore, when we use $F'(t)$ instead of $F(t)$, from Eqn. 10 we derive:

$$\mu_{\alpha,e}(t) \leq \bar{\mu}_{\alpha,e}(t) = \frac{F'(t-kT)a_{e,g} + \sum_{j=0}^{k-1} G(t-jT)a^j_{e,g}}{M_{\alpha,e}} \tag{16}$$

where $\bar{\mu}_{\alpha,e}(t)$ is upper-bounding $\mu_{\alpha,e}(t)$ all the time, so that we can limit $\mu_{\alpha,e}(t)$ by limiting $\bar{\mu}_{\alpha,e}(t)$.

Since $\sum_{j=0}^{k-1} G(t-jT)$ is also convex with time t, and let $\tau_1, \ldots, \tau_m$, where $\tau_1 = 0, \tau_m = T$, be the conjunction points of linear segments in $F'(t)$, $\bar{\mu}_{\alpha,e}(t)$ becomes a piecewise convex function and each convex piece i ($1 \leq i \leq m-1$) has boundary τ_i and τ_{i+1}. Because the maximum value of a convex function must be on the boundary, the maximum value of convex piece i in $\bar{\mu}_{\alpha,e}(t)$ happens on either $t = \tau_i$ or $t = \tau_{i+1}$. Hence, overall, the maximum value of $\bar{\mu}_{\alpha,e}(t)$ always happens at a collection of particular moments which are $\tau_1, \ldots, \tau_m$. Formally, we have:

$$\bar{\mu}^{max}_{\alpha,e} = \max\{\bar{\mu}_{\alpha,e}(\tau_i) | i = 1, \ldots, m\} \tag{17}$$

Similarly, for link utilizations and number of sessions, we also have:

$$\bar{\mu}^{max}_{bw,l} = \max\{\bar{\mu}_{bw,l}(\tau_i) | i = 1, \ldots, m\} \tag{18}$$

$$\bar{n}^{max}_{\theta,g} = \max\{\bar{n}_{\theta,g}(\tau_i) | i = 1, \ldots, m\} \tag{19}$$

where $\bar{\mu}_{bw,l}(t)$ and $\bar{n}_{\theta,g}(t)$, similar to $\bar{\mu}_{\alpha,e}(t)$, is also derived from replacing $F(t)$ with $F'(t)$ in Eqn. 14 and 3.

Therefore, we can transfer our original objective function Eqn. 15 into following formulation:

$$\text{min.} \sum_{\forall \alpha,e} \mathtt{P}(\bar{\mu}^{max}_{\alpha,e}) \times T + \sum_{\forall g,\theta} h_{g,\theta} \bar{n}^{max}_{g,\theta} \times T \tag{20}$$

We can now obtain an efficiently-solvable LP combining the new objective in Eqn 20 with constraints Eqn. 1∼5 and 16∼19. Note that the linear wise penalty function $\mathtt{P}(\mu)$ can also be encoded with a group of linear constraints [15].

5 Model Extentions

In this section, we extend the model presented in §4 to handle several practical issues.

5.1 Supporting multiple services

Given n services $\{\pi_1, \ldots, \pi_n\}$ which share the resources in the infrastructure, we can first model the resource load ($\mu^{\pi}_{\alpha,e}$) from each of them with Eqn. 1∼19. Finally, we modify the objective function which penalizes the overall high resource utlization from all services:

$$\text{min.} \sum_{\forall \pi} \sum_{\forall \alpha,e} \mathtt{P}(\bar{\mu}^{max,\pi}_{\alpha,e}) \times T + \sum_{\forall \pi} \sum_{\forall g,\theta} h^{\pi}_{g,\theta} \bar{n}^{max,\pi}_{g,\theta} \times T \tag{21}$$

[1] Otherwise, we can also use a piecewise linear function to upper-bound $n^{old}_{\theta,g}(t)$.

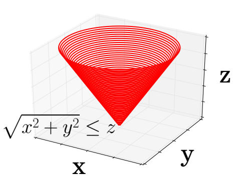


Figure 8: An example of a cone in 3D space.

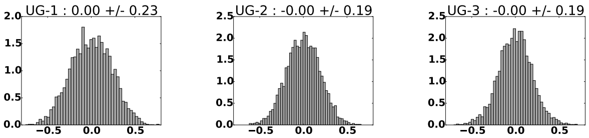


Figure 9: Distribution of prediction errors.

5.2 Handling prediction errors

Before describing how we handle prediction errors, we provide a brief background on second order cone programming. SOCP is a convex optimization problem with cone-shaped constraints besides linear ones. The general format of conic constraints is $\sqrt{\sum_{i=0}^{n-1} x_i^2} \le x_n$, where $x_0, \ldots, x_n$ are variables. Figure 8 shows a concrete example of a cone that results from $\sqrt{x^2+y^2} \le z$ in 3D space. Such constraints can be solved efficiently using the modern interior point method.

To translate our problem into a stochastic model and then to an SOCP, our approach is to model the workload as a random variable, which would make component utilizations random as well. We can then obtain desirable traffic distributions by bounding the random variables for utilization. The challenge is to tractably capture the relationship between random variables that represent workload and those that represent utilization.

To do so, we assume that prediction errors (i.e., differences from actual values) are normally distributed with zero mean. This assumption hold to a first order for our EWMA-based predictor. Figure 9a shows the distribution of the arrival rate estimation errors for three representative UGs. We also assume that the error distributions of different UGs are independent. Independence is not required for actual rates of UGs, which may be correlated (e.g., diurnal patterns). Independence is also not required for estimation errors for different resource (e.g., memory, bandwidth) needed by a UG (because different resource types never co-occur in a cone).

Our formulation ensures that, even with prediction errors, the utilization of resource α on a server or a link e does not exceed $\mu'_{\alpha,e}(t)$ with a high probability p_α (default 99.9%). It computes $\mu'_{\alpha,e}(t)$ based on the predicted workload. The deterministic LP above does not offer this guarantee if the workload is underestimated. Formally, we want:

$$\forall \alpha, e : P[\mu_{\alpha,e}(t) \le \mu'_{\alpha,e}(t)] \ge p_\alpha \tag{22}$$

Our key idea to find $\mu'_{\alpha,e}(t)$ is to establish the relationship between the variance of resource utilizaitons and prediction errors in model inputs.

Let $c_{\alpha,e}$ be the consumption of resource α on a server e, the utilization of α on e is:

$$\mu_{\alpha,e}(t) = \frac{n_e(t) \times c_{\alpha,e}}{M_{\alpha,e}} \tag{23}$$

Both $n_e(t)$ and $c_{\alpha,e}$ can have prediction errors in Eqn. 23. The prediction errors of $n_e(t)$ is mainly from the estimation of session arrival rates of UGs. Let $\tilde{a}_g^k$ be the actual arrival rate of g in epoch-k, the actually number of sessions on server e is:

$$\tilde{n}_e(t) = \quad F(t-kT) \times \textstyle\sum_{\forall g} \tilde{a}_g^k w_{e,g} + n_e^{old}(t) \tag{24}$$

where $n_e^{old}(t)$ is deterministic since we know the actual arrival rates in previous epochs.

The errors lying in $c_{\alpha,e}$ mainly from the heterogeneity of user requests. In each epoch, we predict $c_{\alpha,e}$ but the actual value $\tilde{c}_{\alpha,e}$ could be different. Suppose $\tilde{c}_{\alpha,e}$ is also normally distributed with mean $c_{\alpha,e}$, the actual utilization of resource α on server e is $\tilde{\mu}_{\alpha,e} = \tilde{n}_e \times \tilde{c}_{\alpha,e}/M_{\alpha,e}$. Let Δn_e and $\Delta c_{\alpha,e}$ be the estimation errors of n_e and $c_{\alpha,e}$ respectively, we have:

$$\begin{aligned} \tilde{\mu}_{\alpha,e}(t) &= \frac{(n_e(t) + \Delta n_e) \times (c_{\alpha,e} + \Delta c_{\alpha,e})}{M_{\alpha,e}} \\ &= \frac{n_e(t)c_\alpha + \Delta n_e c_\alpha + n_e(t)\Delta c_{\alpha,e} + \Delta n_e \Delta c_{\alpha,e}}{M_{\alpha,e}} \end{aligned} \tag{25}$$

While $\Delta n_e \Delta c_{\alpha,e}$ is not normally distributed, we make a simplification to upper-bound it with a normal distribution. For instance, let $C_{\alpha,e}$ be the maximum value of $\Delta c_{\alpha,e}$ in practice, we have:

$$\tilde{\mu}_{\alpha,e}(t) \le \tilde{\eta}_{\alpha,e}(t) = \frac{n_e(t)c_{\alpha,e} + \Delta n_e(c_{\alpha,e} + C_{\alpha,e}) + n_e(t)\Delta c_{\alpha,e}}{M_{\alpha,e}} \tag{26}$$

in which $\tilde{\eta}_{\alpha,e}(t)$ is normally distributed. Therefore, we realize Eqn. 22 with its sufficient condition as following:

$$\forall \alpha, e : P[\tilde{\eta}_{\alpha,e}(t) \le \mu'_{\alpha,e}(t)] \ge p_\alpha \tag{27}$$

which is equivalent to:

$$\forall \alpha, e : E[\tilde{\eta}_{\alpha,e}(t)] + \Phi^{-1}(p_\alpha)\sigma[\tilde{\eta}_{\alpha,e}(t)] \le \mu'_{\alpha,e}(t) \tag{28}$$

where Φ^{-1} is the inverse normal cumulative distribution function of $N(0,1)$, and $E[\tilde{\eta}_{\alpha,e}(t)]$ and $\sigma[\tilde{\eta}_{\alpha,e}(t)]$ are the mean and standard variance of $\tilde{\eta}_{\alpha,e}(t)$ respectively. We can compute $E[\mu_{\alpha,e}(t)]$ as a function of the sessions that e carries, using equations similar to those in the temporal model. We can compute $\sigma[\tilde{\eta}_{\alpha,e}(t)]$ as follows. Because $\tilde{n}_e(t)$ is normally distributed, its standard variance is:

$$\sigma[\tilde{n}_e(t)] = \sigma[\Delta n_e(t)] = F(t-kT) \times \sqrt{\sigma[\tilde{a}_g]^2 w_{e,g}^2} \tag{29}$$

Thus, the standard variance of $\bar{\mu}_{\alpha,e}(t)$:

$$\sigma[\tilde{\eta}_\alpha(t)] = \frac{\sqrt{n_e(t)^2\sigma[\tilde{c}_{\alpha,e}]^2 + (C_{\alpha,e}+c_{\alpha,e})^2\sigma[\tilde{n}_e(t)]^2}}{M_{\alpha,e}} \tag{30}$$

The quadratic formulations in Eqns. (28)-(30) are essentially cone constraints. For example, if we merge Eqn. (28), (29) and (30), we derive:

$$\sqrt{\sigma[\tilde{c}_{\alpha,e}]^2 n_e(t)^2 + [F(t-kT)(C_{\alpha,e}+c_{\alpha,e})]^2 \sum_{\forall g} \sigma[\tilde{a}_g]^2 w_{e,g}^2} \\ * \frac{\Phi^{-1}(p_\alpha)}{M_{\alpha,e}} + E[\tilde{\eta}_{\alpha,e}(t)] \leq \mu'_{\alpha,e}(t) \tag{31}$$

in which $w_{e,g}$, $n_e(t)$ and $\bar{\mu}'_{\alpha,e}(t)$ are variables while others are known. To make sure Eqn 33 is valid during the whole epoch $t \in [0,T]$, we take advantage of the fact that

$$\sqrt{\sigma[\tilde{c}_{\alpha,e}]^2 n_e(t)^2 + [F(t-kT)(C_{\alpha,e}+c_{\alpha,e})]^2 \sum_{\forall g} \sigma[\tilde{a}_g]^2 w_{e,g}^2} \tag{32} \\ \leq \sigma[\tilde{c}_{\alpha,e}] n_e(t) + F(t-kT)(C_{\alpha,e}+c_{\alpha,e}) \sqrt{\sum_{\forall g} \sigma[\tilde{a}_g]^2 w_{e,g}^2}$$

so that we instead require:

$$\frac{\Phi^{-1}(p_\alpha)}{M_{\alpha,e}} [\sigma[\tilde{c}_{\alpha,e}] n_e(t) + F(t-kT)(C_{\alpha,e}+c_{\alpha,e}) \tag{33} \\ * \sqrt{\sum_{\forall g} \sigma[\tilde{a}_g]^2 w_{e,g}^2}] + E[\tilde{\eta}_{\alpha,e}(t)] \leq \bar{\mu}'_{\alpha,e}(t)$$

in which the maximum value of the left-hand-side term also happen at $t = 0, T$ or the turning points of $F'(t)$.

We solve these constraints along with the earlier ones temporal model to obtain desired outputs. In the objective function (Eqn. 20), we use $\mu'_{\alpha,e}(t)$ instead of $\bar{\mu}^{max}_{\alpha,e}(t)$. The same principles as before are used to remove the dependence on time t.

6 Footprint Design

We now describe the design of Footprint in more detail.

Defining UGs We start with each /24 IP address prefix as a UG because we find experimentally that such users have similar performance. In the presence of eDNS, where LDNS resolvers report users' IP addresses when querying our (authoritative) DNS servers, this definition of UGs suffices. However, eDNS is not widely deployed and our DNS servers tend to see only resolvers' (not users') addresses. This lack of visibility means that we cannot independently map to entry point(s) two /24 prefixes that share LDNS resolvers. Thus, we merge non-eDNS UGs that share LDNS resolvers.[2] Our entry point performance monitoring (described below) provides the user IP-to-LDNS resolver mapping.

Entry point performance monitoring We leverage client-side application code to monitor performance of UGs to different entry points. Our measurement method borrows ideas from prior work [22, 5]. After a query finishes, the user requests a URL from current and alternative entry points. It then reports all response times to a measurement server, which allows us to compare entry points head-to-head, without worrying about differences across users (e.g., home network performance).

However, because there can be $O(100)$ entry points, requesting that many URLs will take a long time and place undue burden on users. We thus perform these measurements with a small probability and limit each to three requests. Each URL is of the form `http://<guid>.try<k>.service.footprint.com/monitor`, where *guid* is a globally unique identifier and $k \in (1..3)$.

What sits behind *monitor* is service-specific transaction. For a browsing-type service (e.g., search, e-commerce, or social networking) it may correspond to downloading its typical Web page; and for a video streaming service, large objects may be downloaded. This way, the response time reflects what users of the service experience.

The measurement mechanics are as follows. Because of the GUID, the URL hostname does not exist in DNS caches and each request triggers a lookup at our DNS server. We resolve the name based on the user's UG and k. For k=1, we resolve to the current-best entry point; for k=2, to a randomly selected entry point from the ten next best; and for k=3, to a random selection from the remaining entry points. Each response-time triplet yields the relative performance of the best and two other entry points. Aggregating across triplets and users provides a view of each entry point's performance for each UG.

This view is more up-to-date for better entry points for a UG as they are sampled from a smaller set (of 10). When a UG's entry point is changed, it is likely mapped to another nearby entry point; up-to-date view of such entry points is important, which would be hard to obtain with unbiased sampling of all entry points.

Finally, we learn the mapping from users' IP addresses to LDNS resolvers by using GUIDs to join the logs at HTTP transaction servers (which see users' addresses) and DNS servers (which see resolver addresses).

Clustering UGs After LDNS-based grouping, we are left with O(100K) UGs, which presents a scaling problem for our LP solver. To reduce the number of UGs, we aggregate UGs at the start of each epoch. For each UG, we first rank all entry points in decreasing order of performance and then combine into virtual UGs that

[2] We find that such grouping hurts a small minority of users. 90% of the time, when two /24 prefixes have the same LDNS, their relative performance to top-3 entry points is similar.

have the same entry points in the top-three positions in the same order. We formulate the model in terms of VUGs. The performance of a VUG to an entry point is the average of the aggregate, weighted by UGs' number of sessions. For our infrastructure, this clustering creates O(1K) VUGs, and we do not observe a decline efficiency due to the loss in our ability to map individual UGs.

System workload The controller estimates the workload for the next epoch using workload information from previous epochs. DNS servers report the arrival rates of new sessions for each UG and each service; proxies report on resource usage and departure rate of sessions; and network switches that face the external world report on non-edge traffic matrix (in bytes/second). Edge workload is captured in terms of all resource(s) that are relevant for allocation (e.g., memory, CPU, traffic). We use exponentially weighted moving average (EWMA) to estimate workload for the next epoch. We also use linear regression to infer per-session resource consumption (e.g., CPU cycles) for each service, using overall resource usage and number of active sessions per service.

System health When failures occur, health monitoring services at proxy sites and DCs inform the controller how much total site capacity is lost (not which servers). This information granularity suffices because the controller does not allocate sessions to individual servers at a site and relies on local load balancers for that. In contrast, network failures are exposed at link-level, so that the controller can determine network paths.

Robustness to controller failures To make the system robust to controller or hosting-DC failures, we run multiple controllers in different DCs. All dynamic information required for the optimization (e.g., system workload) is reported to all controllers in parallel. However, only one controller—the leader, elected using ZooKeeper [31]—computes the new system configuration. When the leader fails, a new leader is elected, who can immediately start computing new system configurations as it already has all the requisite inputs.

7 Footprint Prototype

We have implemented the Footprint design outlined above. The client-side functionality for entry point performance monitoring is a JavaScript library that can be used with any Web service. This library is invoked after page load completes, so that it does not interfere with user experience. When fetching a URL in JavaScript, we cannot separate DNS lookup and object download times. To circumvent this limitation, before fetching the URL, we fetch a small object from the same hostname. Then, because of DNS caching, the response time of the URL does not include DNS lookup time. In cases where the browser supports the W3C Resource Timing API, we use the precise object fetch time. We implemented the DNS server-side functionality by modifying BIND [3] and proxy functionality using Application Request Routing [2], which works with unmodified Web servers. We use Mosek [24] to solve the LP.

Timely processing of monitoring data is critical. A particularly onerous task is the real-time join between HTTP and DNS data, to know which endpoints our JavaScript has measured and to attach detailed network and geographic information to each measurement. To help scale, we build our processing pipeline on top of Microsoft Azure Event Hub and Stream Analytics.

To scale the computation of new configurations, we limit the number of e2e-paths that a VUG (virtual UG) can use. Specifically, we limit each VUG to its best three entry points (the same ones that the VUG was clustered on), each load balancer to three proxies, and each source-destination switch pair to six paths (tunnels) in the WAN. In our benchmarks, these limits speed computation by multiple orders of magnitude, without noticeably impacting system efficiency or performance.

We deployed a prototype of Footprint in a modest-sized testbed. This environment emulates a WAN with eight switches and 14 links, three proxy sites, and two DCs. Proxy sites and DCs have one server each. We have 32 PCs that act as UGs and repeatedly query a service hosted in the DC. UG to entry point delays are controlled using a network emulator.

The monitoring aspects Footprint (but not the control functionality) are also deployed in Microsoft's service delivery infrastructure. This allow us to collect data from $O(100)$ routers, $O(50)$ edge sites, and $O(10)$ DCs worldwide. JavaScript library is randomly included in 20% of Bing user requests. We use the data collected from this deployment to drive simulations to evaluate the performance of Footprint.

8 Experimental Evaluation

We evaluate Footprint along several dimensions of interest. First, we use the testbed to show the viability and value of jointly controlling all types of infrastructure components. However, this setting does not shed light on its efficiency and performance in a real deployment. To assess those aspects, we conduct large-scale simulations based on data gathered using the monitoring deployment of Footprint in our production environment.

8.1 Testbed results

We use a simple experiment on our testbed to demonstrate the value of spatial traffic modulation. In this experiment, we recreate the example in Figure 2. Recall that in this example the WAN gets congested such that no

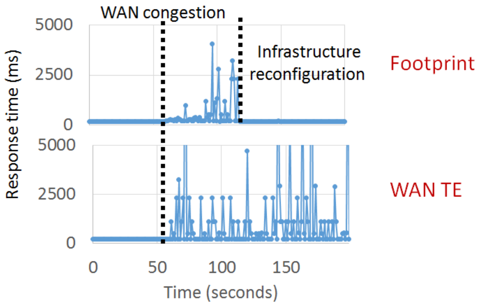


Figure 10: Testbed experiment: WAN congestion.

path between the entry point P2 and DC2 is congestion-free. We create such congestion by injecting non-edge traffic that uses those paths.

Figure 10 shows the results. It plots the response time for UGs that are originally mapped to P2 and DC2, with Footprint and with WAN TE alone. WAN TE estimates the WAN traffic matrix based on recent history and routes traffic to minimize link utilization while using short paths [18]. We see that soon after congestion occurs, Footprint spatially modulates the traffic such that UGs' performance is restored. But WAN TE is unable to resolve congestion and performance issues as it cannot change UGs' proxy and DC selections.

8.2 Efficiency and performance

To understanding the efficiency and performance of Footprint at scale, we conduct detailed simulations using data from Microsoft's service delivery infrastructure.

8.2.1 Data and methodology

We drive our simulations using the following data: *i*) timestamps of new sessions obtained from system logs; *ii*) distribution of session lifetimes; *iii*) UG to entry point performance data from our monitoring deployment; *iv*) propagation latencies and capacities of all WAN links in WAN; *v*) server capacities at the edge proxies and data centers; *vi*) non-edge traffic carried by the WAN; and *vii*) per-session resource consumption (e.g., CPU) estimated using linear regression over the number active sessions.

We develop a custom, fluid-level simulator that uses this data to simulate the behavior of Footprint and other systems that we study. The results in this paper are based on one week's worth of data from August 2015. Results from other weeks are qualitatively similar.

To understand the benefit of Footprint's joint optimization, we compare it to Microsoft's current approach, where *i*) anycast routing is used to map UGs to their best proxy; proxies are organized in multiple anycast "rings," and when a proxy gets overloaded, its DNS server starts mapping UGs to the inner anycast ring; *ii*) each edge proxy independently chooses the closest DC for its user sessions based on up-to-date delay measurements; and *iii*) WAN TE periodically configures network paths based on observed traffic, to minimize maximum link utilization [18]. In our simulations, the control loops, for DC selection at each proxy and for WAN TE, run independently every 5 minutes. To mimic anycast routing, we use our monitoring data to map UGs to the best proxy, which enables a fair comparison by factoring out any anycast suboptimalities [5]. Abusing terminology, we call this system FastRoute, even though the FastRoute paper [14] discusses only UG-to-proxy mapping aspect of service delivery.

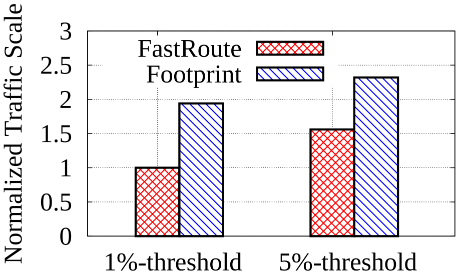


Figure 11: Efficiency of FastRoute and Footprint for SLO1 (excess traffic on overloaded components).

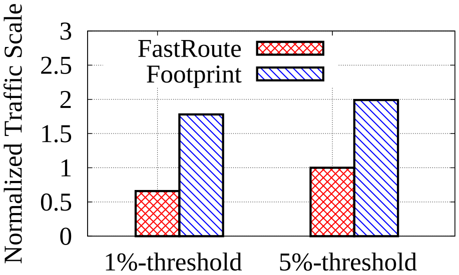
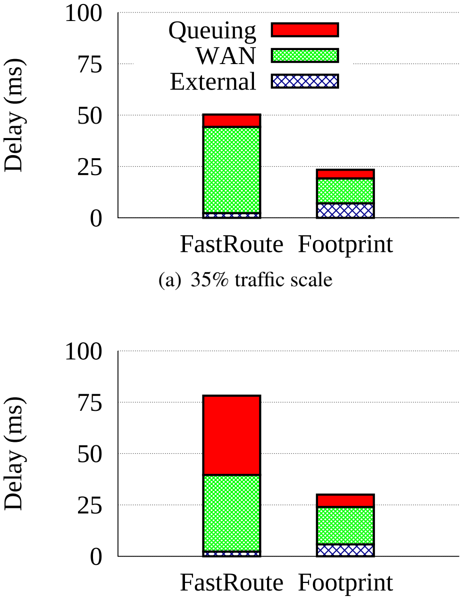


Figure 12: Efficiency of FastRoute and Footprint for SLO2 (total traffic on overloaded components).

We quantify efficiency of a TE scheme using *congestion-free scale*—maximum demand that it can carry without causing unacceptable congestion that violates service-level objectives (SLOs). We consider two definitions of unacceptable congestion: *i*) SLO1: across all epochs, the amount of traffic in excess of component capacities should be less than a threshold; *ii*) SLO2: across all epochs, the total traffic traversing overloaded (i.e., load greater than capacity) components should be less than a threshold. The difference in the two SLOs is that when traffic traverses an overloaded component, SLO1 considers only the fraction in excess of the capacity, but SLO2 considers all traffic passing through it. We study multiple congestion thresholds and compute congestion-free scale by iterating over demands that are proportionally scaled versions of the original demand.



(a) 35% traffic scale

(b) 70% traffic scale

Figure 13: Delays in the two systems.

8.2.2 Efficiency

Figure 11 shows the congestion-free scale for FastRoute and Footprint with SLO1 for two different congestion thresholds. For confidentiality, we report all traffic scales relative to the congestion-free scale of FastRoute with SLO1 at 1% threshold. We see that Footprint can carry 93% more traffic when the congestion threshold is 1% and 50% more traffic when the threshold is 5%.

These efficiency gains can be understood with respect to the spatial modulation enabled by joint coordination in Footprint. At different times of the day, different regions are active and get congested. By making joint decisions, Footprint can more easily divert traffic from currently active regions to uncongested proxies and WAN paths.

Figure 12 shows that Footprint's efficiency gains hold for SLO2 as well, which considers total traffic traversing overloaded components. For 1% and 5% congestion thresholds, Footprint can carry, respectively, 170% and 99% more traffic than FastRoute.

To understand the impact of topology and workload we restricted our analysis to different parts of the world (e.g., USA), which have substantially different infrastructure topology, capacities, and workload. In each case, we found that Footprint brings substantial efficiency benefit over FastRoute.

8.2.3 Performance

We quantify performance of user sessions using end-to-end path delays. We study its contributing factors: external (UG-to-proxy) delay, propagation delay inside the WAN, and queuing-induced delays. Queuing delay is quantified using utilization, per the curve in Figure 7. Figure 13 shows the performance of the two system for traffic scales that correspond to 35% and 70% of the congestion-free scale of FastRoute for SLO1. Each bar stacks from bottom three factors in the order listed above.

We see that even when the traffic demand is low (35%), Footprint has 46% lower delay. At this scale, the infrastructure is largely under-utilized. The delay reduction of Footprint stems from its end-to-end perspective. In contrast, FastRoute picks the best proxy for a UG and the best DC for the proxy. The combination of the two might not represent the best e2e path. Such a path may be composed of a suboptimal UG-to-proxy path but a much shorter WAN path. This effect can be seen in the graph, where the external delays are slightly higher but the sum of external and WAN delay is lower.

When the traffic demand is high (70%), both systems have higher delay. For FastRoute, most of the additional delay stems from queuing as traffic experiences highly utilized links and servers. Footprint is able to reduce queuing delay by being better able to find uncongested (albeit longer) longer paths. Overall, the end-to-end delay of Footprint is 38% lower than FastRoute.

8.3 Impact of system model

To isolate the impact of the high-fidelity model system model that Footprint uses, we compare its efficiency against two alternatives.

- JointAverageCase is a system that, like Footprint, jointly optimizes the infrastructure. It does not, however, model the temporal dynamics of traffic. Based on the session lifetimes, it uses Little's law [21] to estimate the number of active sessions as a function of session arrival rate. So, if the session arrival rate at a proxy is A per second and the average session lifetime is 10 seconds, on average the proxy will have $10 \times A$ active sessions. These estimates are then plugged into an LP that determines how new sessions are mapped to proxies and DC and how traffic is forwarded in the WAN.
- JointWorstCase also jointly optimizes the infrastructure. But for session dynamics, it makes a conservative, worst-case assumption about load on infrastructure components. Specifically, it assumes that new sessions arrive before any old sessions depart in an epoch. Since we do not do admission control, it is not the case that traffic that is estimated, per this model, to overload the infrastructure is rejected. Instead, the optimization spreads traffic to minimize utilization that is predicted by this model. This model will do well if it overestimates the traffic on each component by a similar amount.

Figure 14 compares these two models with Footprint for SLO1 with 5% congestion threshold (the configura-

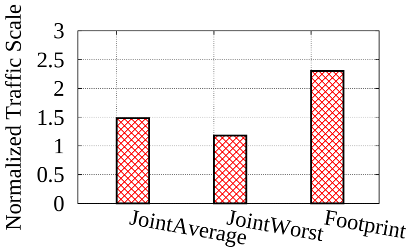


Figure 14: Efficiency of different system models: unscaled session lifetimes

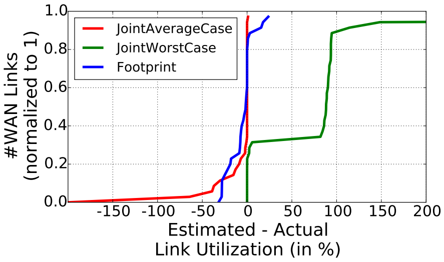


Figure 15: Fidelity of modeling. CDF of modeled and observed link utilization.

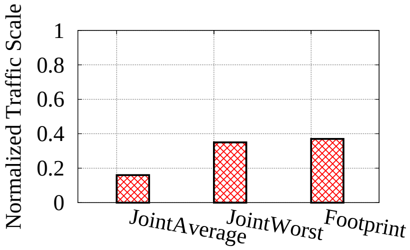


Figure 16: Efficiency of different system models: average session lifetime multiplied by 10

tion for which Footprint had least gain over FastRoute). We see that Footprint is substantially more efficient. It carries 56% and 96% more traffic than JointAverageCase and JointWorstCase.

We find that the gains of Footprint actually stem from its ability to better model load that will be placed on different infrastructure components. To demonstrate it, Figure 15 plots the distribution of estimated minus actual utilization for WAN links for each model. We see that JointAverageCase tends to underestimate utilization and JointWorstCase tends to overestimate it. With respect to appropriately spreading load through the infrastructure, neither over- nor under-estimation is helpful.

We also find that, if sessions were much longer, JointWorstCase becomes better because its conservative assumption about existing sessions continuing to load the infrastructure becomes truer. On the other hand, JointAverageCase gets worse because it ignores the impact of existing sessions altogether, which hurts more when sessions are longer. This is illustrated in Figure 16, which shows the impact on efficiency with average session lifetime multiplied by 10. Note that even for these abnormally long sessions, Footprint offers some gains.

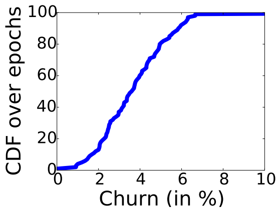


Figure 17: Churn in UG to proxy performance.

8.4 Computation time

We measured the time Footprint controller takes to compute system configurations, which includes converting inputs to an LP, solving it, and converting the output to system variables. On an Intel Xeon CPU (E5-1620, 3.70GHz) with 16 GB RAM and using Mosek v7, this time is 7 seconds. This level of performance is acceptable given that epochs are much longer (5 minutes).

8.5 Preliminary experience

Before concluding our evaluation, we make two observations based on the deployment of Footprint's monitoring components in Microsoft's infrastructure. First, we quantify the fraction of UGs for which the best proxy changes across epochs. If this fraction is substantial, optimal user-to-proxy mapping would move large amounts of WAN traffic, which is better done in coordination with WAN-TE, rather than independently.

Figure 17b shows the fraction of UGs, weighed by their demand, for which the best proxy changes across epochs. On average, this fraction is 5%. It means that a user-to-proxy mapping control loop, operating independently, could move this high a fraction of traffic on the WAN. Joint coordination helps make such movements safely. (In Footprint, since we consider WAN-internal capacity and performance as well, the traffic moved is lower, under 1% in our simulations.)

Second, an unexpected side-effect of Footprint's continuous monitoring is that we can discover and circumvent issues in Internet routing that hurt user performance. We have found several such events. For instance, in one case users in the middle of the USA started experiencing over 130 ms round trip delay to reach a proxy on the west coast, while the historical delay was under 50 ms. In another case, the difference in the delay to reach two nearby proxies in Australia, was over 80 ms. While debugging and fixing such issues requires manual effort, Footprint can automatically restore user performance in the meanwhile. Anycast-based systems, such as FastRoute, cannot do that.

9 Related Work

Content distribution systems Content and service delivery has been an important problem in the Internet for almost two decades. Akamai [23] developed the first large-scale solution, and we borrow several of its concepts such as using edge proxies to accelerate performance and mapping users to proxies based on path performance and proxy load. Since then, researchers have developed sophisticated techniques to tackle this general problem known as replica selection [13, 4, 11, 27]. Solutions tailored to specific workloads (e.g., video) have also been developed [25, 17].

Most of these works target the traditional context in which the WAN is operated separately from the proxy infrastructure. We target the increasingly common integrated infrastructure context, which provides a new opportunity to jointly coordinate routing and resource allocation decisions.

Coordinating decisions Other researchers have noted the downsides of independent decisions for network routing and content distribution. Several works [16, 20, 26, 12] consider coordinating ISP routing and DC selection—no proxies involved—through controlled information sharing (as ownership of infrastructures is not necessarily unified); PECAN develops techniques to coordinate proxy selection and external (not WAN) paths between users and proxies [29]; ENTACT balances performance and the cost of transit traffic for an OSP [30]. Our context is different; it includes the full complement of jointly selecting proxies, DCs, and network paths.

More importantly, prior works quantify baseline gains of coordination, which is equivalent to our Joint-WorstCase scheme. We go further and develop new techniques to improve performance and efficiency by effectively modeling workload dynamics. To our knowledge, such dynamics have not been modeled before, and we show that they result in significant efficiency gain.

10 Conclusions

Our work exploits the opportunity to boost performance and efficiency when online services are delivered over integrated infrastructure. We show that exploitation requires a faithful modeling of the complex dynamics of such systems due to gradual load migrations. We address this challenge using a time-based model that is also computationally efficient to solve. We implemented our approach in the Footprint system and partially deployed it in Microsoft's infrastructure. Data-driven simulations show that, compared to the current method, it improves system efficiency by at least 50% and user performance by at least 38%.

Our approach can be applied to other systems where load changes gradually with time in response to configuration changes (e.g., session-based load balancers). We plan to explore its general applicability in the future.

References

[1] G. Appenzeller, I. Keslassy, and N. McKeown. Sizing router buffers. In *SIGCOMM*, 2004.

[2] Application request routing. `http://en.wikipedia.org/wiki/Application_Request_Routing`.

[3] BIND. https://www.isc.org/downloads/bind/.

[4] S. Buchholz and T. Buchholz. Replica placement in adaptive content distribution networks. In *ACM symposium on Applied computing*, 2004.

[5] M. Calder, E. Katz-Bassett, R. Mahajan, and J. Padhye. Analyzing the Performance of an Anycast CDN. In *IMC*, 2015.

[6] Amazon CloudFront. http://aws.amazon.com/cloudfront/.

[7] Windows Azure CDN. http://azure.microsoft.com/en-us/services/cdn/.

[8] Facebook CDN. https://gigaom.com/2012/06/21/like-netflix-facebook-is-planning-its-own-cdn/.

[9] Google CDN. https://cloud.google.com/storage/.

[10] Level3 CDN. http://www.level3.com/en/products/content-delivery-network/.

[11] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. Dynamic replica placement for scalable content delivery. In *Peer-to-peer systems*, 2002.

[12] D. DiPalantino and R. Johari. Traffic engineering vs. content distribution: A game theoretic perspective. In *INFOCOM*, 2009.

[13] J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.

[14] 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*, 2015.

[15] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional IP routing protocols. *Comm. Mag.*, 40(10), 2002.

[16] 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. *SIGCOMM CCR*, 43(3), 2013.

[17] 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*, 2015.

[18] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven WAN. In *SIGCOMM*, 2013.

[19] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat. B4: Experience with a globally-deployed software defined wan. In *SIGCOMM*, 2013.

[20] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang. Cooperative content distribution and traffic engineering in an ISP network. In *SIGMETRICS*, 2009.

[21] Little's law. `https://en.wikipedia.org/wiki/Little′s_law`.

[22] R. Maheshwari. How LinkedIn used PoPs and RUM to make dynamic content download 25% faster. `https://engineering.linkedin.com/performance/how-linkedin-used-pops-and-rum-make-dynamic-content-download-25-faster`, June 2014.

[23] Cloud computing services and content distribution network (CDN) provider — akamai. `https://www.akamai.com`.

[24] Mosek ApS. `https://mosek.com/`.

[25] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, real-time centralized control for cdn-based live video delivery. In *SIGCOMM*, 2015.

[26] S. Narayana, J. W. Jiang, J. Rexford, and M. Chiang. To coordinate or not to coordinate? wide-area traffic management for data centers. In *CoNEXT*, 2012.

[27] L. Qiu, V. N. Padmanabhan, and G. M. Voelker. On the placement of web server replicas. In *INFOCOM*, 2001.

[28] P. Sun, R. Mahajan, J. Rexford, L. Yuan, M. Zhang, and A. Arefin. A network-state management service. In *SIGCOMM*, 2014.

[29] V. Valancius, B. Ravi, N. Feamster, and A. C. Snoeren. Quantifying the benefits of joint content and network routing. In *SIGMETRICS*, 2013.

[30] Z. Zhang, M. Zhang, A. Greenberg, Y. C. Hu, R. Mahajan, and B. Christian. Optimizing cost and performance in online service provider networks. In *NSDI*, 2010.

[31] Apache zookeeper. `https://zookeeper.apache.org/`.