A Practical Architecture for an Anycast CDN

HUSSEIN A. ALZOUBI, Case Western Reserve University
SEUNGJOON LEE, AT&T Research
MICHAEL RABINOVICH, Case Western Reserve University
OLIVER SPATSCHECK and JACOBUS VAN DER MERWE, AT&T Research

IP Anycast has many attractive features for any service that involve the replication of multiple instances across the Internet. IP Anycast allows multiple instances of the same service to be "naturally" discovered, and requests for this service to be delivered to the closest instance. However, while briefly considered as an enabler for content delivery networks (CDNs) when they first emerged, IP Anycast was deemed infeasible in that environment. The main reasons for this decision were the lack of load awareness of IP Anycast and unwanted side effects of Internet routing changes on the IP Anycast mechanism.

In this article we re-evaluate IP Anycast for CDNs by proposing a load-aware IP Anycast CDN architecture. Our architecture is prompted by recent developments in route control technology, as well as better understanding of the behavior of IP Anycast in operational settings. Our architecture makes use of route control mechanisms to take server and network load into account to realize load-aware Anycast. We show that the resulting redirection requirements can be formulated as a Generalized Assignment Problem and present practical algorithms that address these requirements while at the same time limiting connection disruptions that plague regular IP Anycast. We evaluate our algorithms through trace based simulation using traces obtained from a production CDN network.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks

General Terms: Performance, Algorithms, Design

Additional Key Words and Phrases: Anycast, content delivery networks, autonomous system, load balancing, routing

ACM Reference Format:
Alzoubi, H. A., Lee, S., Rabinovich, M., Spatscheck, O., and Van Der Merwe, J. 2011. A practical architecture for an Anycast CDN. ACM Trans. Web 5, 4, Article 17 (October 2011), 29 pages.
DOI = 10.1145/2019643.2019644 http://doi.acm.org/10.1145/2019643.2019644

1. INTRODUCTION

The use of the Internet to distribute media content continues to grow. The media content in question runs the gambit from operating system patches and gaming software,

The preliminary version of this article appeared in *Proceedings of the 17th International World Wide Web Conference (WWW'08)*.
This material is based on work supported in part by the National Science Foundation under Grant No. CNS-0615190. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
Author's addresses: H. A. Alzoubi and M. Rabinovich, Case Western Reserve University, Electrical Engineering & Computer Science, 10900 Euclid Avenue, Cleveland, OH 44106-7071; email: {Hussein.Alzoubi, Michael.Rabinovich@case.edu; S. Lee, O. Spatscheck, and J. Van der Merwe, AT&T Research Labs, 180 Park Ave., Florham Park, NJ 07932; email: {slee, spatsch, kobus}@research.att.com.

DOI 10.1145/2019643.2019644 http://doi.acm.org/10.1145/2019643.2019644

to more traditional Web objects and streaming events and more recently user generated video content [Cha et al. 2007]. Because of the often bursty nature of demand for such content [Jung et al. 2002], and because content owners require their content to be highly available and be delivered in timely manner without impacting presentation quality [Reibman et al. 2004], content distribution networks (CDNs) have emerged over the last decade as a means to efficiently and cost effectively distribute media content on behalf of content owners.

The basic architecture of most CDNs is simple enough, consisting of a set of CDN nodes distributed across the Internet [Biliris et al. 2001]. These CDN nodes serve as proxies where users (or "eyeballs" as they are commonly called) retrieve content from the CDN nodes, using a number of standard protocols. The challenge to the effective operation of any CDN is to send eyeballs to the "best" node from which to retrieve the content, a process normally referred to as "redirection" [Barbir et al. 2003]. Redirection is challenging, because not all content is available from all nodes, not all nodes are operational at all times, nodes can become overloaded and, perhaps most importantly, an eyeball should be directed to a node that is in close proximity to it to ensure satisfactory user experience.

Because virtually all Internet interactions start with a domain name system (DNS) query to resolve a hostname into an IP address, the DNS system provides a convenient mechanism to perform the redirection function [Barbir et al. 2003]. In this mechanism, the DNS system operated by the CDN receives queries for hostnames of the accelerated URLs and resolves them into the IP address of a CDN node that the DNS system selects for a given query. Most commercial CDNs make use of this mechanism to perform redirection. DNS-based redirection, however, exhibits several well-known limitations. First, in DNS-based redirection, the client sends its DNS query through its local DNS server, and the CDN's DNS system only knows the latter when making its redirection decision. In other words, the actual eyeball request is not redirected, rather the local DNS server of the eyeball is redirected [Barbir et al. 2003]. Unfortunately, not all eyeballs are in close proximity to their local-DNS servers [Mao et al. 2002; Shaikh et al. 2001], and what might be a good CDN node for a local DNS server is not necessarily good for all of its eyeballs. Second, the DNS system was not designed for very dynamic changes in the mapping between hostnames and IP addresses. As a consequence, the local DNS server can cache and reuse its DNS query responses for a certain period of time and for multiple clients. This complicates load distribution decisions for the CDN by limiting the granularity of its control over load balancing; furthermore, because the number of eyeballs behind a local DNS server is unknown to the CDN, the amount of load on a CDN node resulting from a single redirection decision can be difficult to predict. This problem can be mitigated significantly by having the DNS system make use of very short time-to-live (TTL) values, which control the extent of the DNS response reuse. However, a rather common practice of caching DNS queries by local-DNS servers, and especially certain browsers, beyond specified TTL means that this remains an issue [Pang et al. 2004]. Finally, the DNS-based redirection assumes that the CDN explicitly selects a nearby CDN node for the originator of a given DNS query. To know the distance between any two IP addresses on the Internet requires a complex measurement infrastructure.

In this work we revisit IP anycast as redirection technique, which, although examined early in the CDN evolution process [Barbir et al. 2003], was considered infeasible at the time. IP anycast refers to the ability of the IP routing and forwarding architecture to allow the same IP address to be assigned to multiple endpoints, and to rely on Internet routing to select between these different endpoints. Endpoints with the same IP address are then typically configured to provide the same service. For example, IP anycast is commonly used to provide redundancy in the DNS root-server

deployment [Hardie 2002]. Similarly, in the case of a CDN, all endpoints with the same IP anycast address can be configured to be capable of serving the same content.

Because it fits seamlessly into the existing Internet routing mechanisms, IP anycast packets are routed "optimally" from an IP forwarding perspective. That is, for a set of anycast destinations within a particular network, packets are routed along the shortest path and thus follow a proximally optimal path from a network perspective. Packets traveling towards a network advertising an IP anycast prefix, will similarly follow the most proximal path towards that destination within the constraints of the inter-provider peering agreements along the way.

From a CDN point of view, however, there are a number of problems with IP anycast. First, because it is tightly coupled with the IP routing apparatus, any routing change that causes anycast traffic to be re-routed to an alternative instance of the destination IP anycast address may cause a session reset to any session-based traffic such as TCP. Second, because the IP routing infrastructure only deals with connectivity, and not the quality of service achieved along those routes, IP anycast likewise is unaware of and can not react to network conditions. Third, IP anycast is similarly not aware of any server (CDN node) load, and therefore cannot react to node overload conditions. For these reasons, IP anycast was originally not considered a viable approach as a CDN redirection mechanism.

Our revisiting of IP anycast as a redirection mechanism for CDNs was prompted by two recent developments. First, route control mechanisms have recently been developed that allow route selection within a given autonomous system to be informed by external intelligence [Duffield et al. 2007; Van der Merwe et al. 2006; Verkaik et al. 2007]. Second, recent anycast-based measurement work [Ballani et al. 2006] shed light on the behavior of IP anycast, as well as the appropriate way to deploy IP anycast to facilitate proximal routing.

Based on these developments, we design a practical load-aware IP anycast CDN architecture for the case when the CDN is deployed within a single global network, such as AT&T's ICDS content delivery service [ATT ICDS 2010]. When anycast endpoints are within the same network provider, the route control mechanism can install a route from a given network entry point to the anycast end-point deemed the most appropriate for this entry point; in particular, both CDN node load and internal network conditions can be taken into account. This addresses the load-awareness concern and in part the route quality of service concern – although the latter only within the provider domain. Route control also deals with the concern about resetting sessions because of route changes. We note that in practice there are two aspects of this problem: (i) Route changes within a network that deploys IP anycast addresses and (ii) Route changes outside of the network which deploys anycast IP addresses. Route control mechanisms can easily deal with the first aspect preventing unnecessary switching between anycast addresses within the network. As for route changes outside of the IP anycast network, the study in Ballani et al. [2006] has shown that most IP prefixes exhibit very good affinity, that is, would be routed along the same path towards the anycast enabled network.

An anycast CDN is free of the limitations of DNS-based CDNs: it redirects actual client demand rather than local DNS servers and thus is not affected by the distance between eyeballs and their local CDN servers; it is not impacted by DNS caching; and it obviates the need for determining proximity between CDN nodes and external destinations. Yet it introduces its own limitations. First, it delivers client requests to the nearest entry point of the CDN network with regard to the forward path from the client to the CDN network. However, due to route asymmetry, this may not produce the optimal reverse path used by response packets. Second, while route control can effectively account for network conditions inside the CDN's autonomous system,

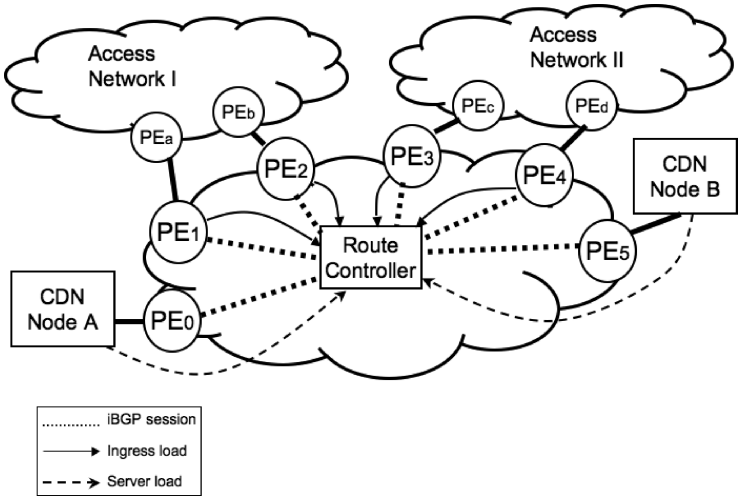


Fig. 1. Load-aware anycast CDN architecture.

external parts of the routes are purely the result of BGP routing. Thus, an obvious question, which we could not answer with the data available, is the end-to-end performance comparison between the anycast and DNS CDNs. Our contribution is rather to make the case that, contrary to the common view, anycast CDN is a viable approach to build a content delivery platform and that it improves the operation of the CDN in comparison with an existing DNS-based approach within the CDN's network. The key aspects of this contribution are as follows.

— We present a practical anycast CDN architecture that utilizes server and network load feedback to drive route control mechanisms to realize CDN redirection (Section 2).
— We formulate the required load balancing algorithm as a *Generalized Assignment Problem* and present practical algorithms for this NP-hard problem that take into consideration the practical constraints of a CDN (Section 3).
— Using server logs from an operational production CDN (Section 4), we evaluate our algorithms by trace driven simulation and illustrate their benefit by comparing with native IP anycast and an idealized load-balancing algorithm, as well as with the current DNS-based approach (Section 5).

2. ARCHITECTURE

In this section we first describe the workings of a load-aware anycast CDN and briefly discuss the pros and cons of this approach vis-a-vis more conventional CDN architectures. We also give an informal description of the load-balancing algorithm required for our approach before describing it more formally in later sections.

2.1. Load-Aware Anycast CDN

Figure 1 shows a simplified view of a load-aware anycast CDN. We assume a single autonomous system (AS) in which IP anycast is used to reach a set of CDN nodes distributed within the AS. For simplicity we show two such CDN nodes, A and B in Figure 1. In the rest of this article, we use the terms "CDN node" and "content server" interchangeably. We further assume that the AS in question has a large footprint in the country or region in which it will be providing CDN service; for example, in the US,

Tier-1 ISPs have this kind of footprint.[1] Our article investigates synergistic benefits of having control over the PEs of a CDN. We note that these assumptions are both practical, and, more importantly, a recent study of IP anycast [Ballani et al. 2006] has shown this to be the ideal type of deployment to ensure good proximity properties.[2]

Figure 1 also shows the route controller component that is central to our approach [Van der Merwe et al. 2006; Verkaik et al. 2007]. The route controller activates routes with provider edge (PE) routers in the CDN provider network. As described in Van der Merwe et al. [2006], this mechanism involves pre-installed MPLS tunnels routes for a destination IP address (an anycast address in our case) from each PE to every other PE. Thus, to activate a route from a given PE PE_i to another PE PE_j, the controller only needs to signal PE_i to start using an appropriate MPLS label. In particular, route change does not involve any other routers and in this sense is an atomic operation.

The route controller can use this mechanism to influence the anycast routes selected by the ingress PEs. For example, in Figure 1, to direct packets entering through PE PE_1 to the CDN node B, the route controller would signal PE_1 to activate the MPLS tunnel from PE_1 to PE_5; to send these packets to node A instead, the route controller would similarly activate the tunnel from PE_1 to PE_0. For our purposes, the route controller takes as inputs, ingress load from the PEs at the edge of the network, server load from the CDN nodes for which it is performing redirection, and the cost matrix of reaching a given CDN server from a given PE to compute the routes in accordance with the algorithms described in Section 3.

The load-aware anycast CDN then functions as follows (with reference to Figure 1). All CDN nodes that are configured to serve the same content (A and B), advertise the same IP anycast address into the network via BGP (respectively through PE_0 and PE_5). PE_0 and PE_5 in turn advertise the anycast address to the route controller, which is responsible to advertise the (appropriate) route to all other PEs in the network (PE_1 to PE_4). These PEs in turn advertise the route via eBGP sessions with peering routers (PE_a to PE_d) in neighboring networks so that the anycast address becomes reachable throughout the Internet (in the figure represented by access networks I and II).

Request traffic for content on a CDN node will follow the reverse path. Thus, a request will come from an access network, and enter the CDN provider network via one of the ingress routers PE_1 to PE_4. In the simple setup depicted in Figure 1, such request traffic will then be forwarded to either PE_0 or PE_5 en-route to one of the CDN nodes.

Based on the two load feeds (ingress PE load and server load) provided to the route controller, it can decide which ingress PE (PE_1 to PE_4) to direct to which egress PE (PE_0 or PE_5). By assigning different PEs to appropriate CDN nodes, the route controller can minimize the network costs of processing the demand and distributed the load among the CDN nodes.

In summary, our approach utilizes the BGP-based proximity property of IP anycast to deliver clients packets to nearest ingress PEs. These external portions of the paths of anycast packets are determined purely by inter-AS BGP routes. Once packets enter the provider network, it is the route controller that decides where these packets will be delivered through mapping ingress PEs to content servers. The route controller makes these decisions taking into account both network proximity of the internal routes and server loads.

[1] `http://www.business.att.com`, `http://www.level3.com`

[2] Note that while our focus in this work is on anycast CDNs, we recognize that these conditions can not always be met in all regions where a CDN provider might provide services, which suggests that a combination of redirection approaches might be appropriate.

2.2. Objectives and Benefits

We can summarize the goals of the architecture described above as follows: (i) To utilize the natural IP anycast proximity properties to reduce the distance traffic is carried towards the CDN's ISP; (ii) To react to overload conditions on CDN servers by steering traffic to alternative CDN servers; (iii) To minimize the disruption of traffic that results when ongoing sessions are being re-mapped to alternative CDN servers. Note that this means that "load-balancing" per server is not a specific goal of the algorithm: while CDN servers are operating within acceptable engineering loads, the algorithm should not attempt to balance the load. On the other hand, when overload conditions are reached, the system should react to deal with that, while not compromising proximity.

A major advantage of our approach over DNS-based redirection systems is that the actual eyeball request is being redirected, as opposed to the local-DNS request in the case of DNS-based redirection. Further, with load-aware anycast, any redirection changes take effect very quickly, because PEs immediately start to route packets based on their updated routing table. In contrast, DNS caching by clients (despite short TTLs) typically results in some delay before redirection changes have an effect.

The granularity of load distribution offered by our route control approach is at the PE level. For large tier-1 ISPs the number of PEs is typically in the high hundreds to low thousands. A possible concern for our approach is whether PE granularity will be sufficiently fine grained to adjust load in cases of congestion. Our results in Section 5 indicate that even with PE-level granularity we can achieve significant performance benefits in practice.

Obviously, with enough capacity, no load balancing would ever be required. However, a practical platform needs to have load-balancing ability to cope with unexpected events such as flash crowd and node failures, and to flexibly react to even more gradual demand changes because building up physical capacity of the platform is a very coarse-grain procedure. Our experiments will show that our architecture can achieve effective load balancing even under constrained resource provisioning.

Before we describe and evaluate redirection algorithms that fulfill these goals, we briefly describe two other CDN-related functions enabled by our architecture that are not further elaborated upon in this article.

2.3. Dealing with Long-Lived Sessions

Despite increased distribution of rich media content via the Internet, the average Web object size remains relatively small [King 2006]. This means that download sessions for such Web objects will be relatively short lived with little chance of being impacted by any anycast re-mappings in our architecture. The same is, however, not true for long-lived sessions, for example, streaming or large file download [Van der Merwe et al. 2002]. (Both of these expectations are validated with our analysis of connections disruption count in Section 5.)

In our architecture, we deal with this by making use of an additional application level redirection mechanisms *after* a particular CDN node has been selected via our load-aware IP Anycast redirection. This interaction is depicted in Figure 2. As before an eyeball will perform a DNS request that will be resolved to an IP Anycast address (*i* and *ii*). The eyeball will attempt to request the content using this address (*iii*), however, the CDN node will respond with an application level redirect (iv) [Van der Merwe et al. 2003] containing a unicast IP address associated with this CDN node, which the eyeball will use to retrieve the content (v). This unicast address is associated only with this CDN node, and the eyeball will therefore continue to be serviced by the same node regardless of routing changes along the way. While the additional overhead associated with application level redirection is clearly unacceptable when downloading

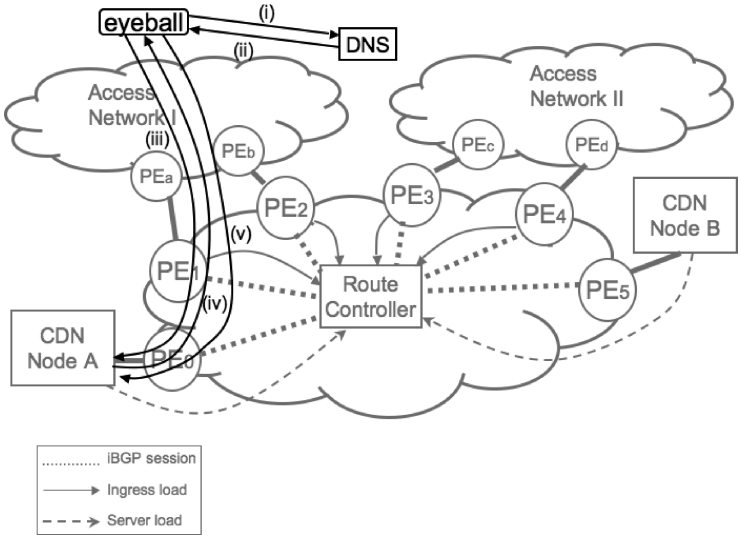


Fig. 2. Application level redirection for long-lived sessions.

small Web objects, it is less of a concern for long-lived sessions where the startup overhead is amortized.

In parallel work, we proposed an alternative approach to handle extremely large downloads using anycast, without relying on HTTP redirection [Al-Qudah et al. 2009]. Instead, the approach in Al-Qudah et al. [2009] recovers from an disruption by reissuing the HTTP request for the remainder of the object using a range HTTP request. The CDN could then trigger these disruptions intentionally to switch the user to a different server mid-stream if the conditions change. However, that approach requires a browser extension.

Recently, some CDNs started moving into utility (also known as cloud) computing arena, by deploying applications at the CDN nodes. In this environment, applications often form long-lived sessions that encompass multiple HTTP requests, with individual requests requiring the entire session state to execute correctly. Commercial application servers, including both Weblogic and Websphere, allow servers to form a wide-area cluster where each server in the cluster can obtain the session state after successfully receiving any HTTP request in a session. Based on this feature, our approach for using anycast for request redirection can apply to this emerging CDN environment.

2.4. Dealing with Network Congestion

As previously described, the load-aware CDN architecture only takes server load into account in terms of being "load-aware". (In other words, the approach uses network load information in order to effect the server load, but does not attempt to steer traffic away from network hotspots). The Route Control architecture, however, does allow for such traffic steering [Van der Merwe et al. 2006]. For example, outgoing congested peering links can be avoided by redirecting response traffic on the PE connecting to the CDN node (e.g., PE_0 in Figure 1), while incoming congested peering links can be avoided by exchanging BGP Multi-Exit Discriminator (MED) attributes with appropriate peers [Van der Merwe et al. 2006]. We leave the full development of these mechanisms for future work.

3. REMAPPING ALGORITHM

The algorithm for assigning PEs to CDN nodes has two main objectives. First, we want to minimize the service disruption due to load balancing. Second, we want to minimize the network cost of serving requests without violating server capacity constraints. In this section, after presenting an algorithm that minimizes the network cost, we describe how we use the algorithm to minimize service disruption.

3.1. Problem Formulation

Our system has m servers, where each server i can serve up to S_i concurrent requests. A request enters the system through one of n ingress PEs, and each ingress PE j contributes r_j concurrent requests. We consider a cost matrix c_{ij} for serving PE j at server i. Since c_{ij} is typically proportional to the distance between server i and PE j as well as the traffic volume r_j, the cost of serving PE j typically varies with different servers.

The first objective we consider is to minimize the overall cost without violating the capacity constraint at each server. The problem is called *Generalized Assignment Problem* (GAP) and can be formulated as the following integer linear optimization problem [Shmoys and Tardos 1993].

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{m} x_{ij} = 1, \ \forall j \\ & \sum_{j=1}^{n} r_j x_{ij} \le S_i, \ \forall i \\ & x_{ij} \in \{0, 1\}, \ \forall i, j \end{aligned}$$

where indicator variable x_{ij}=1 iff server i serves PE j, and x_{ij}=0 otherwise. Note that this formulation reflects our "provider-centric" perspective with the focus on minimizing the costs on the network operator. In particular, the model favors overall cost reduction even if this means redirecting some load to a far-away server. In principle, one could bound the proximity degradation for any request by adding a constraint that no PE be assigned to a content server more than k times farther away than the closest server. In practice, however, as we will see later (Figure 9), the penalty for a vast majority of requests is very small relative to the current system.

When x_{ij} is an integer, finding an optimal solution to GAP is NP-hard, and even when S_i is the same for all servers, no polynomial algorithm can achieve an approximation ratio better than 1.5 unless P=NP [Shmoys and Tardos 1993]. Recall that an α-approximation algorithm always finds a solution that is guaranteed to be at most α times the optimum.

Shmoys and Tardos [1993] present an approximation algorithm (called ST-algorithm in this article) for GAP, which involves a relaxation of the integrality constraint and a rounding based on a fractional solution to the LP relaxation. It first obtains the initial total cost value C using linear programming optimization, by removing the restriction that x_{ij} be integer (in which case, this problem formulation becomes an LP optimization problem). Then, using a rounding scheme based on the fractional solution, the algorithm finds an integer solution whose total cost is at most C and the load on each server is at most $S_i + \max r_j$. ST-algorithm forms the basis for the traffic control decisions in our approach, as discussed in rest of this section.

In our approach, the route controller periodically re-examines the PE-to-server assignments and computes a new assignment if necessary using a *remapping algorithm*. We call the period between consecutive runs of the mapping algorithm the *remapping interval*. We explore two remapping algorithms: one that attempts to minimize the cost of processing the demand from the clients (thus always giving cost reduction a priority over connection disruption), and the other that attempts to minimize the connection disruptions even if this leads to cost increase.

3.2. Minimizing Cost

Algorithm 1. Minimum Cost Algorithm

```
INPUT: Offered_Load[j] for each PE j, Current_Load[i] for each server i, and cost matrix
Cost[Servers][PEs] {as Miles*Mbps}
Run expanded ST-algorithm
{Post_Processing}
repeat
   Find the most overloaded server i;
   Let P_i be the set of PEs served by i. Map PEs from P_i to i, in the descending order of
   Offered_Load, until i reaches it capacity
   Remap(i, {the set of still-unmapped PEs from P_i})
until None of the servers is overloaded OR No further remapping would help
return
...
Remap(Server: i, PE_Set: F):
for all j in F, in the descending order of Offered_Load: do
   Find server q with the minimum Cost[][j] and enough residual capacity for Offered_Load[j]
   Find server t with the highest residual capacity
   if q exists and q != i then
      Remap j to q
   else
      Map j to t {t is less overloaded than i}
   end if
end for
```

The remapping algorithm for minimizing costs is shown in pseudocode as Algorithm 1. It begins by running what we refer to as *expanded* ST-algorithm to try to find a feasible solution as follows. We first run ST-algorithm with given server capacity constraints, and if it could not find a solution (which means the load is too high to be satisfied within the capacity constraints at any cost), we increase the capacity of each server by 10% and try to find a solution again.[3] In our experiments, we set the maximum number of tries at 15, after which we give up on computing a new remapping and retain the existing mapping scheme for the next remapping interval. However, in our experiments, the algorithm found a solution for all cases and never skipped a remapping cycle.

Note that ST-algorithm can lead to server overload (even relative to the increased server capacities), although the overload amount is bounded by $\max r_j$. In practice, the overload volume can be significant since a single PE can contribute a large request load (e.g., 20% of server capacity). Thus, we use the following post-processing on

[3]Note that the capacity constraints are just parameters in the algorithm and in practice assigned to be less than the physical capacity of the servers.

the solution of ST-algorithm to find a feasible solution without violating the (possibly increased) server capacities.

We first identify the most overloaded server i, and then among all the PEs served by i, find the set of PEs F (starting from the least-load PE) such that server i's load becomes below the capacity S_i after off-loading F. Then, starting with the highest-load PEs among F, we offload each PE j to a server with enough residual capacity q, as long as the load on server i is above S_i. (If there are multiple such servers for j, we choose the one with minimum cost to j, although other strategies such as best-fit are possible.) If there is no server with enough capacity, we find server t with the highest residual capacity and see if the load on t after acquiring j is lower than the current load on i. If so, we off-load PE j to server t even when the load on t goes beyond S_t, which will be fixed in a later iteration.

Once the overload of server i is resolved, we repeat the whole process with then-highest overloaded server. Note that the overload comparison between i and t ensures the monotonic decrease of the maximum overload in the system and therefore termination of the algorithm either because there are no more overloaded servers in the system or "repeat" post-processing loop could further offload any of the overloaded servers.

3.3. Minimizing Connection Disruption

Algorithm 2. Minimum Disruption Algorithm

INPUT: Offered_Load[j] for each PE j, Current_Load[i] for each server i, and cost matrix Cost[Servers][PEs] {as Miles*Mbps}
Let FP be the set of PEs mapped to non-overloaded servers. {These will be excluded from remapping}
For every nonoverloaded server i, set server capacity S_i to be $S_i - Current\_Load[i]$
For every overloaded server i and all PEs j currently mapped to i, set Cost[i][j] = 0
Run expanded ST-algorithm for all servers and all PEs that are NOT in FP {This will try to remap only PEs currently mapped to overloaded servers but can move these PEs to any server to reduce costs}
{Post_Processing}
repeat
  Find the most overloaded server i; {Note that before ST-algorithm this server could have been nonoverloaded so some of its PEs can be in FP}
  Map PEs from ($P_i \cap FP$) to i {Map fixed PEs}
  Map remaining PEs from P_i to i, in the descending order of Offered_Load, until i reaches it capacity
  Remap(i, {the set of still-unmapped PEs from P_i})
until None of the servers is overloaded **OR** no further remapping would help

While Algorithm 1 described in Section 3.2 attempts to minimize the cost, it does not take the current mapping into account and can potentially lead to a large number of connection disruptions.

To address this issue, we present another algorithm, which gives connection disruption a certain priority over cost. For clarity, we start by describing an algorithm that attempts a remapping only when there is a need to offload one or more overloaded servers.

The pseudo-code of the algorithm is shown as Algorithm 2. The algorithm divides all the servers into two groups based on load: overloaded servers and nonoverloaded servers. The algorithm keeps the current mapping of the nonoverloaded servers and only attempts to remap the PEs assigned to the overloaded servers. Furthermore, even

for the overloaded servers, we try to retain the current mappings as much as possible. Yet for the PEs that do have to be remapped due to overloads, we would like to use ST-algorithm to minimize the costs.

We manipulate input to ST-algorithm in two ways to achieve these goals. First, for each nonoverloaded server i, we consider only its residual capacity as the capacity S_i in ST-algorithm. This allows us to retain the server current PEs while optimizing costs for newly assigned PEs. Second, for each overloaded server j, we set the cost of servicing its currently assigned PEs to zero. Thus, current PEs will be reassigned only to the extent necessary to remove the overload.

As described, this algorithm reassigns PEs to different servers only in overloaded scenarios. It can lead to suboptimal operation even when the request volume has gone down significantly and a simple proximity-based routing would yield a feasible solution with lower cost. One way to address this is to exploit the typical diurnal pattern and perform full remapping once a day at a time of low activity (e.g., 4 am every day). Another possibility is to compare the current mapping and the potential lowest-cost mapping at that point, and initiate the reassignment if the cost difference is beyond a certain threshold (e.g., 70%). Our experiments do not account for these optimizations.

To summarize, in our system, we mainly use the algorithm in Section 3.3 to minimize the connection disruption, while we infrequently use the algorithm in Section 3.2 to find an (approximate) minimum-cost solution for particular operational scenarios.

4. EVALUATION METHODOLOGY

This section describes the methodology of our experimental study.

4.1. Dataset

We obtained two types of datasets from a production single-AS CDN: the netflow datasets from its ingress PEs and Web access logs from its cache servers. The access logs were collected for a weekday in July 2007. Each log entry has detailed information about an HTTP request and response such as client IP, cache server IP, request size, response size, and the time of arrival. Depending on the logging software, some servers provide service response time for each request in the log, while others do not. In our experiments, we first obtain sample distributions for different response size groups based on the actual data. For log entries without response time, we choose an appropriate sample distribution (based on the response size) and use a randomly generated value following the distribution.

We use concurrent requests being processed by a server as the load metric that we control in our experiments. In addition, we also evaluate data serving rate as server load indication. To determine the number of concurrent requests r_j coming through an ingress PE j, we look at the client and server IP pair for each log entry and use netflow data to determine where the request has entered the system. We then use the request time from the log and the service response time (actual or computed as previously described) to determine whether a request is currently being served.

One of our objectives is to maximize network proximity in processing client requests. In particular, because we focus on reducing the costs of the CDN's network provider, our immediate goal is to maximize network proximity and network delays inside the CDN's autonomous system. Since the internal response path is always degenerate independently of our remapping (it uses hot-potato routing to leave the AS as quickly as possible), the network proximity between the client's ingress PE and server is determined by the request path.[4] Thus, we use the request path as our cost metric

[4]The proximity of the request's external path (from the client to an entry point into the CDN's AS) is further provided by IP anycast. At the same time, our focus on internal proximity may result in a suboptimal

reflecting the proximity of request processing. Specifically, we obtained from the CDN the distance matrix d_{ij} between every server i and every ingress PE j in terms of air miles and used it as the cost of processing a request. While we did not have access to the full topological routing distances, the latter are known to be highly correlated with air-miles within an autonomous system since routing anomalies within an AS are avoided. Thus, using air miles would not have any significant effect on the results and at the same time make the results independent of a particular topology and routing algorithms. Topological routing distances, if available, could be equally used in our design.

Then, for ST-algorithm, we use the product $r_j d_{ij}$ as the cost c_{ij} of serving requests from PE j at server i.

Another input required by ST-algorithm is the capacity S_i of each server i. To assign server capacity, we first analyze the log to determine the maximum aggregate number of concurrent requests across all servers during the entire time period in the log. Then, we assign each server the capacity equal to the maximum aggregate concurrent requests divided by the number of servers. This leads to a high-load scenario for peak time, where we have sufficient aggregate server capacity to handle all the requests but only assuming ideal load distribution. Note that server capacity is simply a parameter of the load balancing algorithm, and in practice would be specified to be below the servers' actual processing limit. We refer to the latter as the server's physical capacity. In most of the experiments we assume the physical capacity to be 1.6 times the server capacity parameter used by the algorithms.

The CDN under study classifies content into *content groups* and assigns each content group to a certain set of CDN nodes. We use two such content groups for our analysis: one containing *Small Web Objects*, assigned to 11 CDN nodes, and the other *Large File Downloads*, processed by 8 CDN nodes.

4.2. Simulation Environment

We used CSIM[5] to perform our trace driven simulation. CSIM creates process-oriented, discrete-event simulation models. We implemented our CDN servers as a set of facilities that provide services to requests from ingress PEs, which are implemented as CSIM processes. For each request that arrives we determine the ingress PE j, the response time t, and the response size l. We assume that the server responds to a client at a constant rate calculated as the response size divided by the response time for that request. In other words, each request causes a server to serve data at the constant rate of l/t for t seconds. Multiple requests from the same PE j can be active simultaneously on server i. Furthermore, multiple PEs can be served by the same facility at the same time.

To allow flexibility in processing arbitrary load scenarios, we configured the CSIM facilities that model servers to have infinite capacity and very large bandwidth. We then impose capacity limits at the application level in each scenario. Excessive load is handled differently in different systems. Some systems impose access control so that servers simply return an error response to excess requests to prevent them from

external response path since we choose the closest CDN node to the ingress PE and the reserve path could be asymmetric. In principle, the route controller could take into account the proximity of the various CDN nodes to the clients from the perspective of the overall response path. The complication, however, is that our current architecture assumes repinning is done at the granularity of the entire ingress PEs. Thus, any server selection decision would apply to all clients that enter the network at a given PE. Whether these clients are clustered enough in the Internet to exhibit similar proximity when reached from different CDN nodes is an interesting question for future work.

[5]`http://www.mesquite.com`

affecting the remaining workload. In other systems, the excessive requests are admitted and may cause overall performance degradation. Our simulation can handle both these setups. In the setup with access control, at each request arrival, it will be passed to the simulation or dropped depending on the current load of the destination server of this connection. In the setup without access control, we admit all the requests and simply count the number of *over-capacity requests*. An over-capacity request is a request that at the time of its arrival finds the number of existing concurrent requests on the server to already equal or exceed the server's physical capacity limit.

In describing our experiments we will specify which of the setups various experiments follow. In general, the number of over-capacity requests in the setup without access control will exceed the number of dropped requests in the case with access control because, as previously explained, a dropped request imposes no load on the server while the over-capacity connection contributes to server load until it is processed. However, the ultimate goal in dimensioning the system is to make the number of excess requests negligible in either setup, in which case both setups will have the same behavior.

The scale of our experiments required us to perform simulation at the time granularity of one second. To ensure that each request has a non-zero duration, we round the beginning time of a request down and the ending time up to whole seconds.

4.3. Schemes and Metrics for Comparison

We experiment with the following schemes and compare the performance.

— *Trace Playback* (PB). In this scheme we replayed all requests in the trace without any modification of server mappings. In other words, (PB) reflects the current CDN routing configuration.
— *Simple Anycast* (SAC). This is "native" Anycast, which represents an idealized proximity routing scheme, where each request is served at the geographically closest server.
— *Simple Load Balancing* (SLB). This scheme employs anycast to minimize the difference in load among all servers without considering the cost.
— *Advanced Load Balancing, Always* (ALB-A). This scheme always attempts to find a minimum cost mapping as described in Section 3.2.
— *ALB, On-overload* (ALB-O). This scheme aims to minimize connection disruptions as described in Section 3.3. Specifically, it normally only reassigns PEs currently mapped to overloaded servers and performs full remapping only if the cost reduction from full remapping would exceed 70%.

In SAC, each PE is statically mapped to a server, and there is no change in the mappings across the entire experiment run. SLB and ALB-A recalculate the mappings every Δ seconds (the remapping interval). The initial Δ value that we used to evaluate the different algorithms is 120 seconds. Later, in Section 5.5, we examine various values of Δ.

We utilize the following metrics for performance comparison.

— *Server load.* We use the *number of concurrent requests* and *service data rate* at each server as measures of server load. A desirable scheme should keep the number below the capacity limit all the time.
— *Request air-miles.* We examine the *average miles* a request traverses within the CDN provider network before reaching a server as a proximity metric of content delivery within the CDN's ISP. A small value for this metric denotes small network link usage in practice.

— *Disrupted Connections and Over-Capacity Requests.* Another metric of redirection scheme quality is the *number of disrupted connections* due to remapping. Disruption occurs when a PE is remapped from server A to server B; the ongoing connections arriving from the PE may be disconnected because B may not have the connection information. Finally, we use the *number of over-capacity requests* as a metric to compare the ability of different schemes to prevent server overloading. A request is counted as over-capacity if it arrives at a server with existing concurrent requests already at or over the physical capacity limit.

With our redirection scheme, a request may use a server different from the one used in the trace, and its response time may change, for example, depending on the server load or capacity. In our experiments, we assume that the response time of each request is the same as the one in the trace no matter which server processes it as a result of our algorithms.

5. EXPERIMENTAL RESULTS

In this section, we present our simulation results. We first consider the server load, the number of miles for request traffic and the number of disrupted and over-capacity requests that resulted for each of the redirection schemes. In all these experiments, presented in Sections 5.1–5.3, we assume all server capacities to be the same and equal in aggregate the 100% of the maximum total number of concurrent requests in the trace (as described in Section 4.1). Specifically, this translates to 1900 concurrent requests per server for the large-file group and 312 concurrent requests for the small-object group. The remapping interval in these experiments is fixed at 120 seconds, and we assume there is no access control in the system (i.e., excess requests are not dropped and only counted as over-capacity requests). Section 5.5 investigates different remapping interval values.

5.1. Server Load Distribution

We first present the number of concurrent requests at each server for the large files group. For the clarity of presentation, we use the points sampled every 60 seconds.

In Figure 3, we plot the number of concurrent requests at each server over time. Figure 3(a) shows the current behavior of the CDN nodes (Trace Playback (PB)). It is clear that some servers (e.g., server 4) process a disproportionate share of load – 4 to 5 times the load of other servers. This indicates current over-provisioning of the system and an opportunity for significant optimization.

Turning to anycast-based redirection schemes, since SAC does not take load into account but always maps PEs to a closest server, we observe from Figure 3(b) that the load at only a few servers grows significantly, while other servers get very few requests. For example, at 8 am, server 6 serves more than 55% of total requests (5845 out of 10599), while server 4 only receives fewer than 10. Unless server 6 is provisioned with enough capacity to serve significant share of total load, it will end up dropping many requests. Thus, while reducing the peak load of the most-loaded server compared to the playback, SAC still exhibits large load misbalances. As another extreme, Figure 3(c) shows that SLB evenly distributes the load across servers. However, SLB does not take cost into account and can potentially lead to high connection cost.

In Figures 3(d) and Figure 3(e), we present the performance of ALB-A and ALB-O - the two schemes that attempt to take both cost and load balancing into account in remapping decisions. According to the figures, these algorithms do not balance the load among servers as well as SLB. This is expected because their main objective is to find a mapping that minimizes the cost as long as the resulting mapping does not violate the server capacity constraint. Considering ALB-A (Figure 3(d)), in the morning

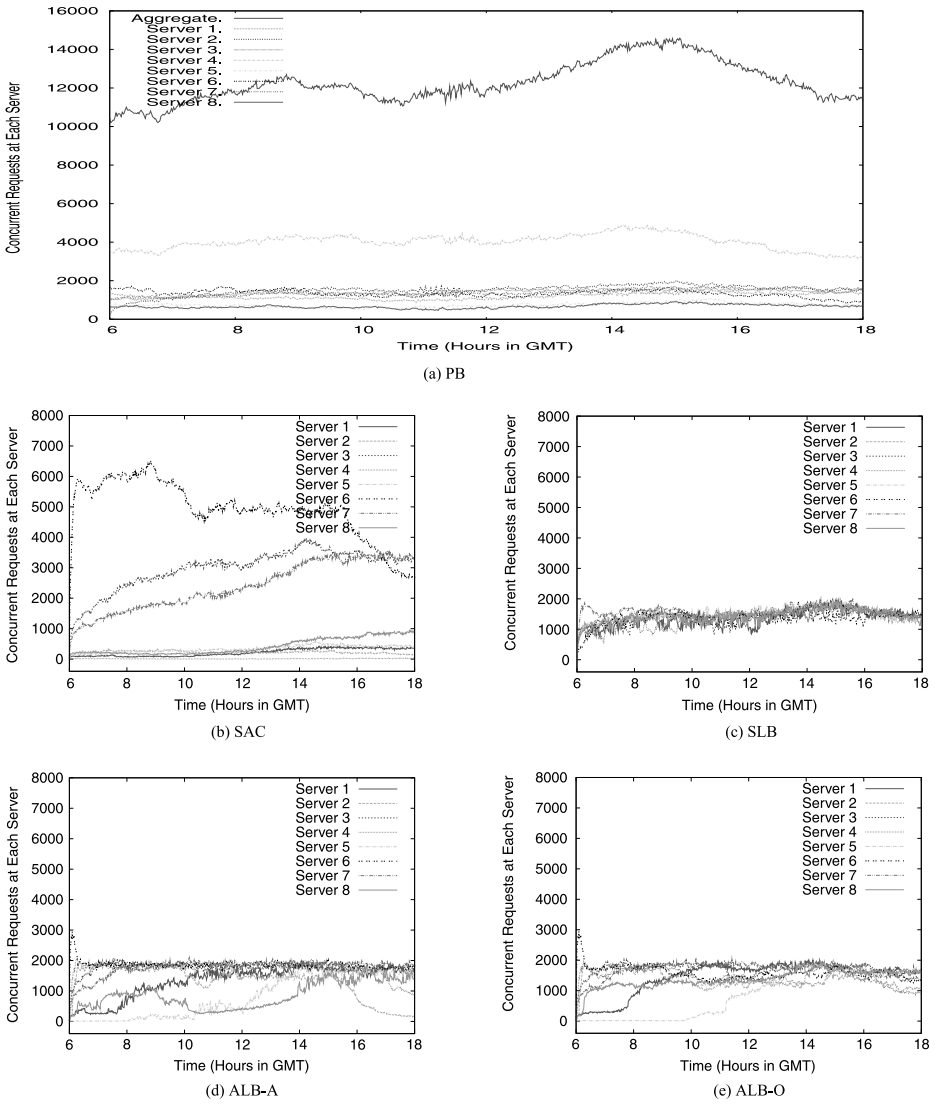


Fig. 3. Number of concurrent requests for each scheme (Large files group).

(around 7 am), a few servers receive only relatively few requests, while other better located servers run close to their capacity. As the traffic load increases (e.g., at 3 pm), the load on each server becomes similar in order to serve the requests without violating the capacity constraint. ALB-O initially shows a similar pattern to ALB-A (Figure 3(e)), while the change in request count is in general more graceful. However, the difference becomes clear after the traffic peak is over (at around 4 pm). This is because ALB-O attempts to reassign the mapping only when there is an overloaded server. As a result, even when the peak is over and we can find a lower-cost mapping, all PEs stay with their servers that were assigned based on the peak load (e.g., at around 3 pm). This property of ALB-O leads to less traffic disruption at the expense of increased overall cost (as we will see later in Section 5.2).

Overall, from the load perspectives, we see that both ALB-A and ALB-O manage to keep maximum server load within roughly 2000 concurrent requests, very close to the 1900 connections capacity used as a parameter for these algorithms. Within these load limits, the algorithms attempt to reduce the cost of traffic delivery.

In Figure 4, we present the same set of results using the logs for small object downloads. We observe a similar trend for each scheme, although the server load changes more frequently. This is because their response size is small, and the average service time for this content group is much shorter than that of the previous group. We also present the serviced data rate of each server in Figure 5 for the large file server group and Figure 6 for small object server group. We observe that there is strong correlation between the number of requests (Figures 3 and 4) and data rates (Figures 5 and 6). In particular, the data rate load metric confirms the observations we made using the concurrent requests metric.

5.2. Disrupted and Over-Capacity Requests

Remapping of PEs to new servers can disrupt active connections. In this section, we investigate the impact of each remapping scheme on connection disruption. We also study the number of over-capacity requests assuming the physical capacity limit of servers to be equal to 1.6 times the capacity parameter used in the remapping algorithms. Specifically, the server physical capacity is assumed to be 2500 concurrent requests in the large file group and 500 concurrent requests in the small file group.

The results are shown in Figure 7. Since SAC only considers PE-to-server proximity in its mappings and the proximity does not change, SAC mappings never change and thus connection disruption does not occur. However, by not considering load, this scheme exhibits many over-capacity requests – over 18% in the large-file group. In contrast, SLB always remaps to achieve as balanced load distribution as possible. As a result, it has no over-capacity requests but a noticeable number of connection disruptions. The overall number of negatively affected requests is much smaller than for SAC but as we will see in the next section, this comes at the cost of increasing the request air-miles.

Figure 7 shows significant improvement of both ALB-A and ALB-O over SAC and SLB in the number of affected connections. Furthermore, by remapping PEs judiciously, ALB-O reduces the disruptions by an order of magnitude over ALB-A without affecting the number of overcapacity requests. Overall, ALB-O reduces the number of negatively affected connections by two orders of magnitude over SAC, by an order of magnitude over SLB in the small files group, and by a factor of 5 over SLB in the large file group.

Finally, Figure 7 shows that large file downloads are more susceptible to disruption in all the schemes performing dynamic remapping. This is because the longer service response of a large download increases its chance of being remapped during its lifetime (e.g., in the extreme, if an algorithm remapped all active connections every time, every connection lasting over 120 seconds would be disrupted). This confirms our architectural assumption concerning the need for application level redirect for long-lived sessions.

In summary, the disruption we observed in our experiments is negligible: at most 0.04% for the ALB-O algorithm (which we ultimately advocate), and even less – 0.015% – for small objects download. Further, disruption happens in the ALB-O algorithm when the platform is already overloaded, when the quality of service is already compromised. In fact, by pinning a client to a fixed server at the beginning of the download, DNS-based CDNs may lead to poor performance in long-running downloads (during which conditions can change). On the other hand, with a simple extension to browsers, as we show in a separate work [Al-Qudah et al. 2009], an anycast-based CDN

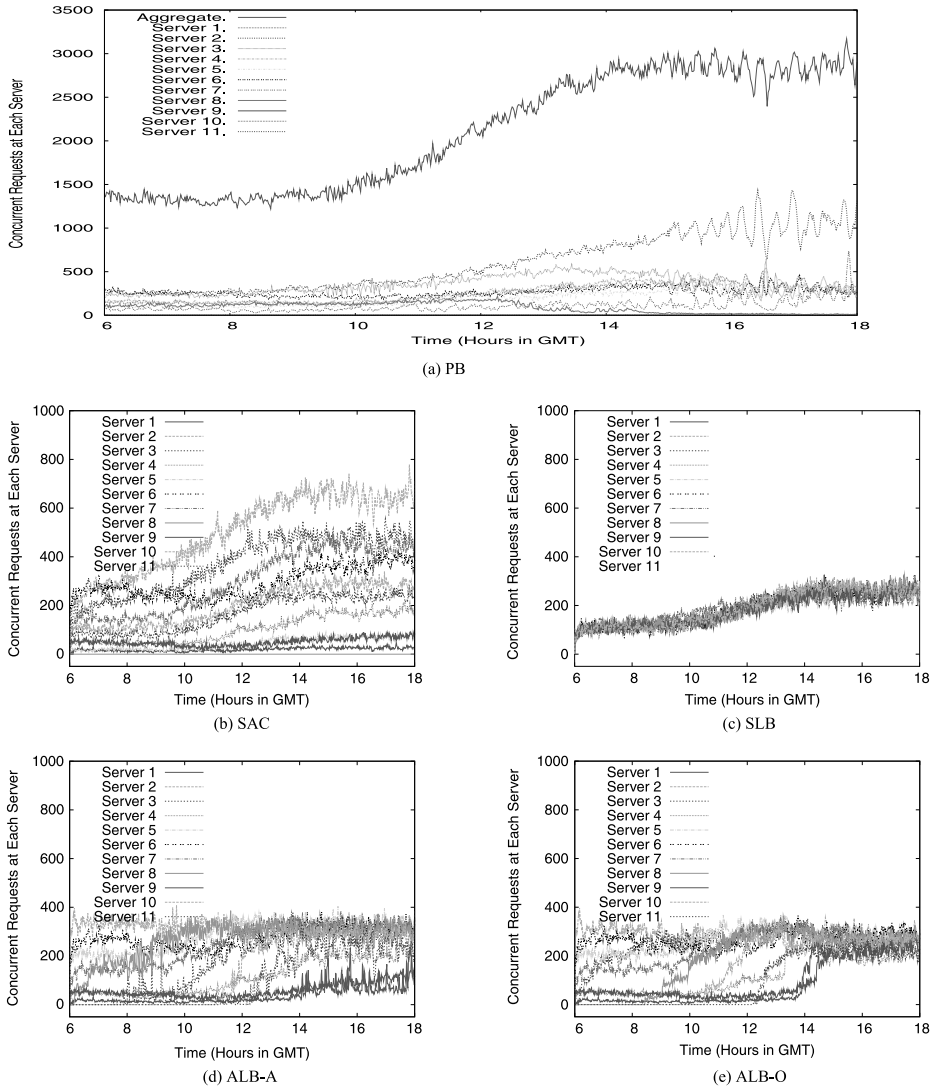


Fig. 4. Number of concurrent requests for each scheme (Small objects group).

could trigger these disruptions intentionally to switch the user to a different server on the fly.

5.3. Request Air Miles

This section considers the cost of each redirection scheme, measured as the average number of air miles a request must travel within the CDN's ISP. Figures 8(a) and 8(b) show the ratio of schemes average cost over the PB average cost calculated every 120 seconds. In SAC, a PE is always mapped to the closest server, and the average mileage for a request is always the smallest (at the cost of high drop ratio as previously shown). This can be viewed as the optimal cost one could achieve and thus it always has the lowest ratio in Figure 8. SLB balances the load among servers without taking cost into account and leads to the highest cost. We observe in Figure 8(a) that ALB-A is nearly

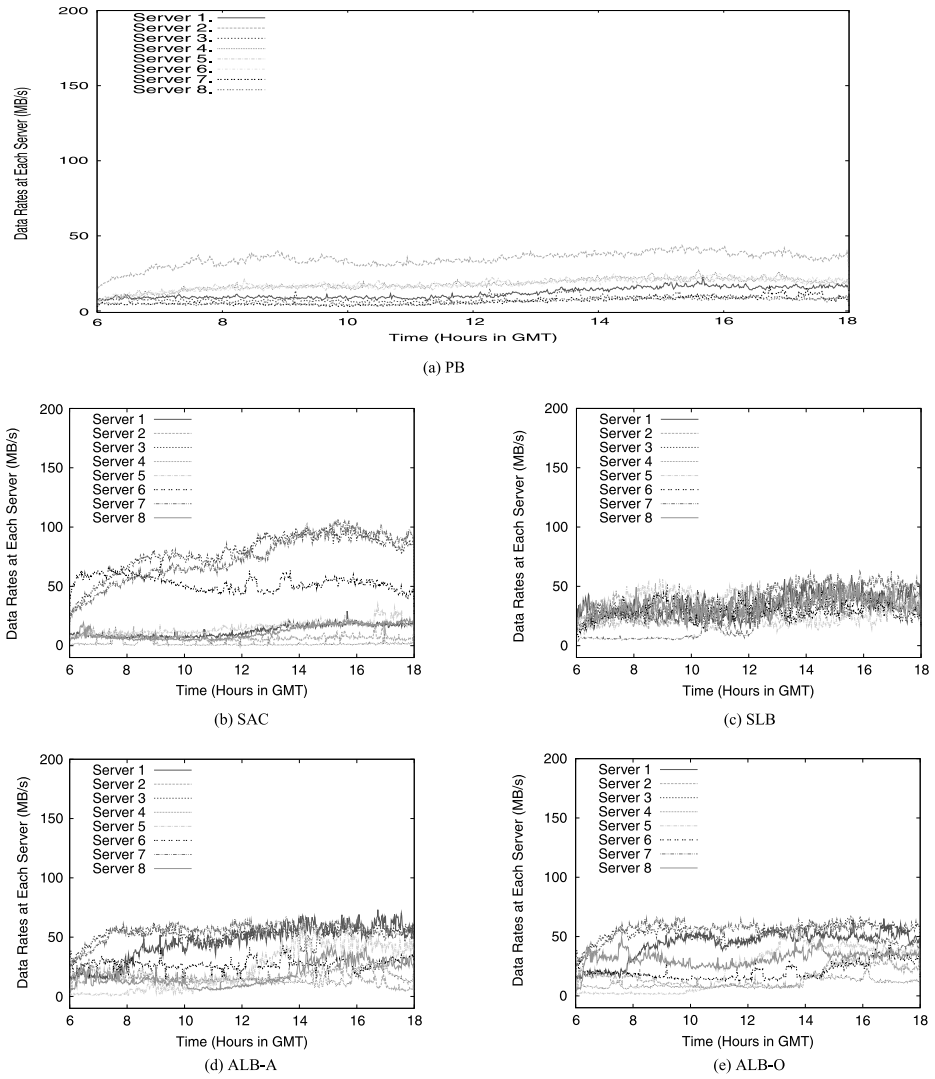


Fig. 5. Service data rate for each scheme (Large files group).

optimal in cost when the load is low (e.g., at 8 am) because in this case each PE can be assigned to the closest server. As the traffic load increases, however, not all PEs can be served at their closest servers without violating the capacity constraint. Then, the cost goes higher as some PEs are remapped to different (farther) servers. ALB-O also finds an optimal-cost mapping in the beginning when the load is low. As the load increases, ALB-O behaves differently from ALB-A because ALB-O attempts to maintain the current PE-server assignment as much as possible, while ALB-A attempts to minimize the cost even when the resulting mapping may disrupt many connections (Figure 7). This restricts the solution space for ALB-O compared to ALB-A, which subsequently increases the cost of ALB-O solution.

With our focus on optimizing the costs for the ISP, our optimization formulation does not restrict the distance for any individual request. Thus, a pertinent question is,

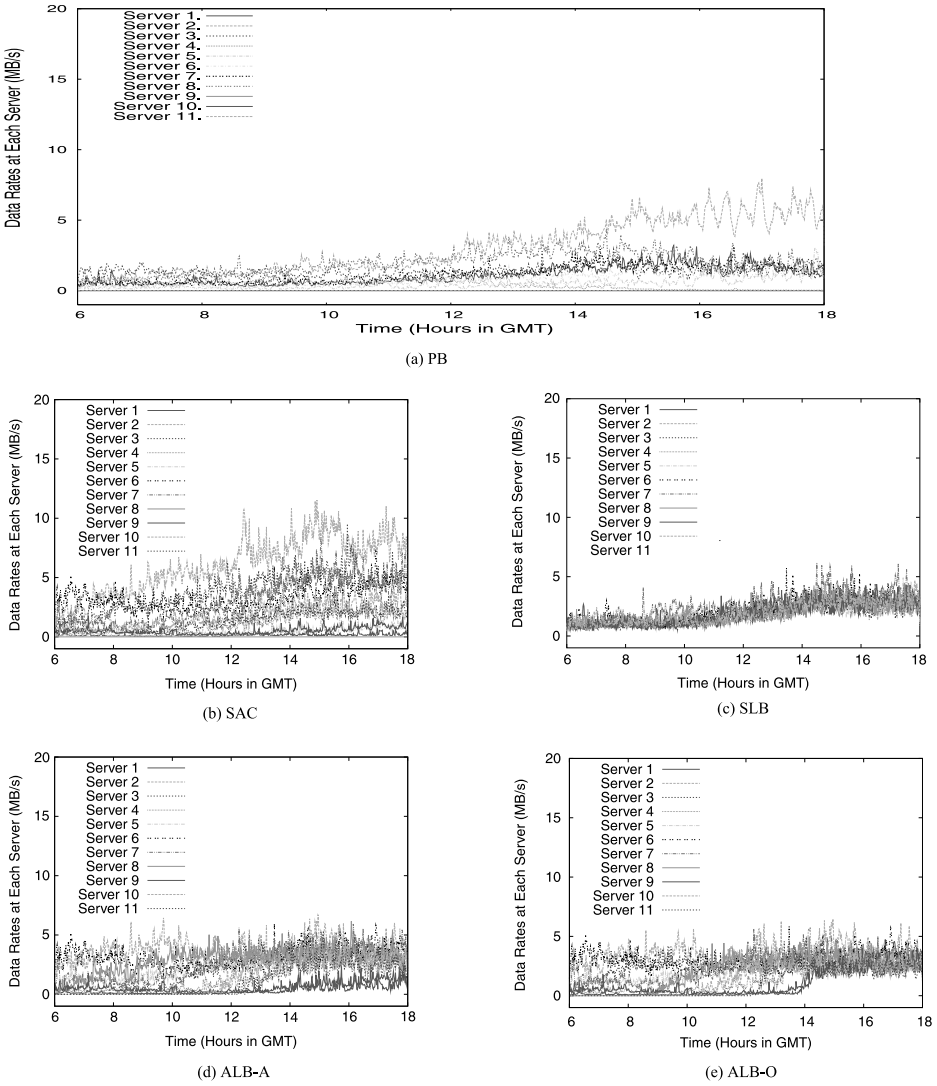


Fig. 6. Service data rate for each scheme (Small objects group).

to which extent individual requests might be penalized by our schemes. Consequently, Figure 9 plots the ratio of the cost of the 99-percentile requests in each scheme. Specifically, in every 120-second interval, we find the request whose cost is higher than 99% of all requests in a given anycast scheme and the request with cost higher than 99% of all requests in the playback, and we plot the ratio of the cost of these requests. Note that because the possible costs for individual requests can only take discrete values, the curves are less "noisy" than for the average costs. We can see that both adaptive anycast algorithms do not penalize individual requests excessively. The ALB-A algorithm actually reduces the cost for a 99-percentile request compared to the playback, and the ALB-O's penalty is at most 12.5% for the Large File Downloads group and 37.5% for the Small Web Object group.

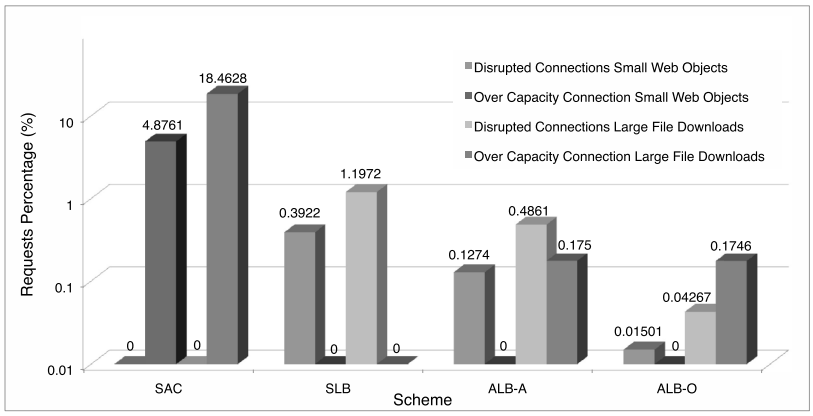
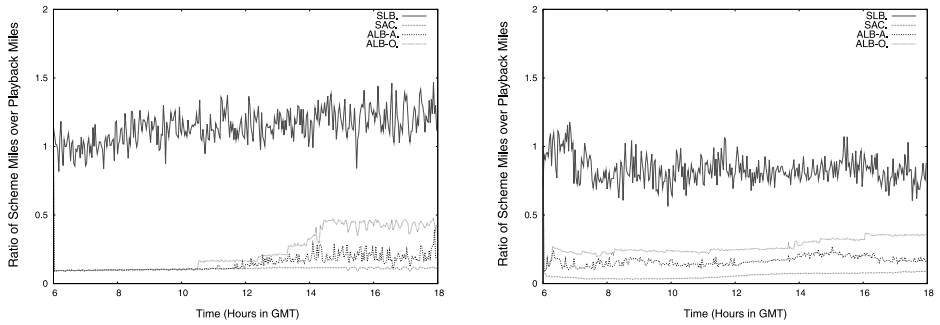
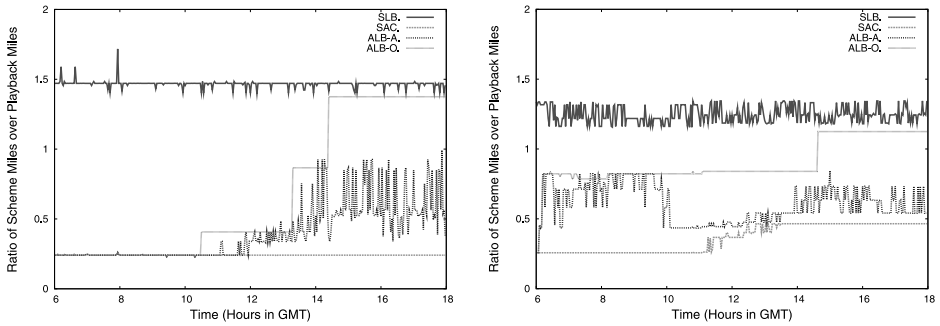


Fig. 7. Disrupted and over-capacity requests for each scheme (Y-axis in log scale).



(a) Small objects group (b) Large files group

Fig. 8. Average miles for requests calculated every 120 seconds.

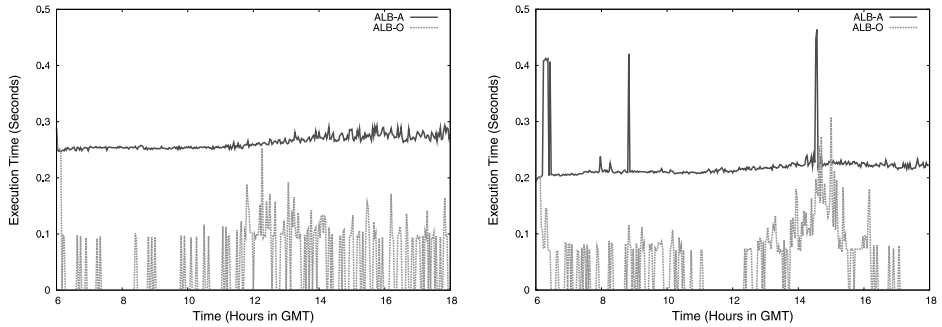


(a) Small objects group (b) Large files group

Fig. 9. 99th percentile of request miles calculated every 120 seconds.

5.4. Computational Cost of Remapping

We now consider the question of the execution time of the remapping algorithms themselves, concentrating on ALB-A and ALB-O as they are the most computationally intensive and also shown above to exhibit the best overall performance among the algorithms we considered. We first time the algorithms on the trace environment and



(a) Small objects group (b) Large files group

Fig. 10. Execution time of the ALB-A and ALB-O algorithms in the trace environment.

then consider how they scale with a potential growth of the platform size. All the experiments were conducted on a single-core Intel PC with Pentium 4 3.2GHz CPU and 1G RAM, running Linux 2.6.31.12 kernel.

Figure 10 plots the measured execution time of both remapping algorithms in our trace-driven simulation. Each data point reflects the actual measured time of each execution after each 120-second remapping interval. The figure shows that for both small and large groups, both algorithms never exceed 0.5s to execute, and in most cases take much less time: the 95th percentile is 0.26s for ALB-A and 0.14s for ALB-O in the small objects group, and 0.23s and 0.17s in the large files group. This is negligible time compared with expected frequencies of remapping decisions. We also observe that ALB-O is more efficient than ALB-A. This is because ALB-O only performs remapping only for overloaded servers, in effect reducing the size of the solution search space and in fact often not solving the LP problem at all (which is reflected in seemingly zero execution time in the figure).

We now turn to the question of how our algorithms will scale with the platform size. To this end, we time the algorithms in a synthetic environment with a synthetic randomly generated workload. We consider a platform with 1000 PEs and up to 100 data centers (compared to the trace environment of around 550 PEs and 8-11 data centers). Each data center (represented as a single aggregate server) has maximum capacity of 1000 concurrent connections. To generate the synthetic workload, we start with a given fraction of aggregate platform capacity as total offered load, and distribute this offered load randomly among the PEs in the following three steps.

— We iterate through the PEs, and for each PE, we assign it a random load between 0 and maximum server capacity (1000). This step results in random load assignment, but the aggregate offered load can significantly deviate from the target level. We make it close (within 10%) to the target level in the next two steps.
— While the total load assigned to all the PEs is below 0.9 of the target:
  — Pick a random PE P and a random load value L between 0 and 250 (one-forth of the server capacity);
  — If $current\_load(P) + L$ is less than server capacity, add L to P's offered load.
— While the total load assigned to all the PEs is above 1.1 of the target:
  — Pick a random PE P and a random load value L between 0 and 250 (one-forth of the server capacity);
  — If $current\_load(P) - L > 0$, subtract L from P's offered load.

We perform this random load assignment every two minutes and then time our algorithms as they remap the PEs to servers. Further, to see how the running time

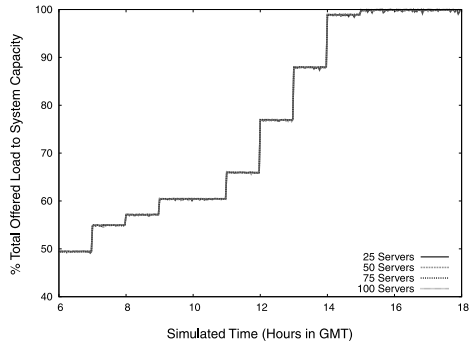
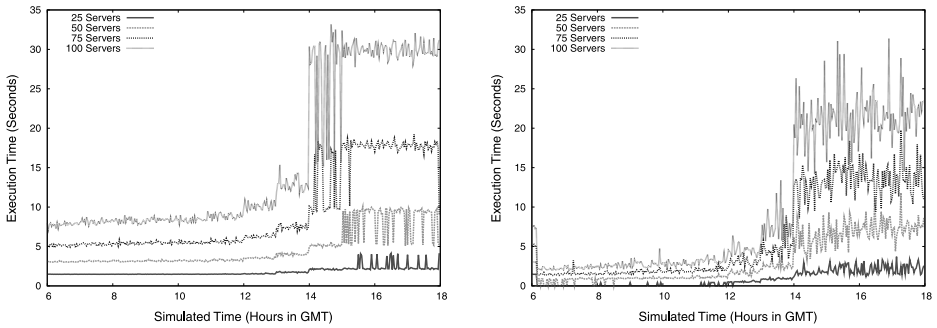


Fig. 11. Total offered load pattern (synthetic environment).



(a) ALB-A (b) ALB-O

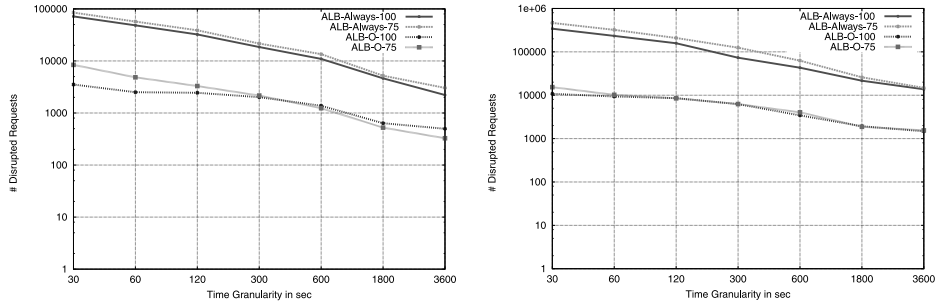
Fig. 12. Scalability of the ALB-A and ALB-O algorithms in a synthetic environment.

depends on the overall load (one can expect that the higher the total load relative to total capacity, the harder the algorithm has to work to find a solution), we increase the load target every hour. The resulting pattern of the total offered load relative to the total capacity is shown in Figure 11. Again, within each period of stable total load, its distribution among the PEs changes randomly every two minutes.

Figure 12 shows the execution time of the ALB-A and ALB-O algorithms for different platform sizes. Understandably, larger platform sizes translate to greater execution time. However, even for 100 data centers, as long as the total load is within 75%, ALB-O generally completes remapping under 5s, and ALB-A within 10s. The more efficient execution of ALB-O is again due to the fact that it performs only incremental remapping each time,[6] and as we see, in as the platform grows in size, the difference can be significant. This again (in addition to the reduction in the number of disrupted connections) argues in favor ALB-O.

Overall, we conclude that even using our very modest machine for remapping, the execution time of our remapping algorithms, especially ALB-O, is acceptable for a platform of a significant scale as long as the total load does not approach too closely the total platform capacity. The fact that our algorithm slows down significantly under extreme load conditions suggests a strategy where the remapping algorithm first checks the total load and if it is found close to (e.g., over 75% of) the platform capacity, switches to a "survival mode" whereby it no longer solves the cost optimization problem but merely redistributes excess load from overloaded to underloaded servers.

[6]Observe that its initial remapping takes the same time as in the ALB-A case.



(a) Small objects group (b) Large files group

Fig. 13. The effect of remapping interval on disrupted connections.

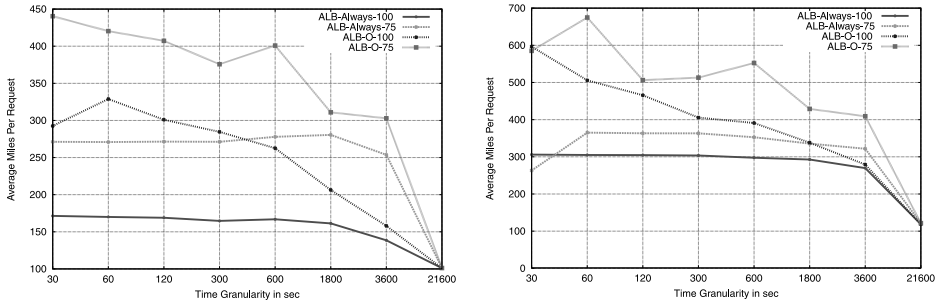
5.5. The Effect of Remapping Interval

In this section, we consider the issue of selecting the remapping interval Δ. Specifically, we consider how different values of the remapping interval affect our main performance metrics: the number of disrupted connections, the cost of operation (measured as the air miles that the request must travel within the AS), and the number of connections dropped due to server overload. Since we already showed that the ALB-A and ALB-O algorithms exhibited the best performance among the algorithms we considered, we concentrate on these two algorithms here.

We ran our simulations with the default server capacity (1900 and 312 concurrent requests for the large and small file groups respectively) which we refer to as 100% capacity scenario, and with a lower capacity equal to 75% of the default. Lowering server capacity for the same load (which is given by the trace) allows us to investigate the behavior of the system under high load conditions. To consider the effect of different assumptions, we assume that overcapacity requests are dropped by the admission control mechanism at their arrival; hence they do not consume any server resources.

In the experiments of this subsection, we run our simulations for the entire trace but collect the results only for the last 6-hour trace period. This allows every scenario to experience at least one remapping (not counting the initial remapping at the end of the first second) before the results are collected. For instance, for $\Delta = 6$ hours, the first remapping occurs on the first second of the trace and is affected by the initially idle servers, the second remapping occurs at 6 hours, and the results are collected in the remaining 6 hours of the trace. For smaller deltas, the results are collected for the same trace period to make the results the comparable across different deltas. Note that this is different from previous experiments where the remapping interval was fixed at 120s, which is negligible relative to the trace duration, allowing us to report the results for the entire trace.

5.5.1. Disruption Count. Figures 13(a) and 13(b) show the affect of the remapping interval on the disrupted connections. As expected, the number of disrupted connections decreases with the increase of the remapping interval in both schemes. However, the figures confirms the superiority of ALB-O in this metric: ALB-O exhibits a smaller number of disrupted connections for all Δ values. This is a consequence of the design of ALB-O, which only performs remapping to relieve overloaded servers when there is a significant potential for cost reduction. ALB-A on the other hand, performs remapping any time it can reduce the costs. Interestingly, the high-load scenario (corresponding to 75% curves) does not significantly affect the disruptions. We speculate that this is due to the fact that the trace period reported by these figures



(a) Small objects group (b) Large files group

Fig. 14. The effect of remapping interval on cost (common 6-hour trace period).

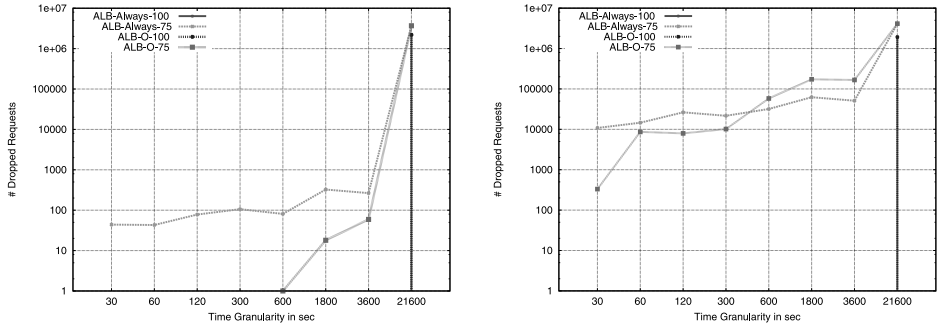
corresponds to the highest load in the trace, and even the 100% capacity scenario triggers similar number of remappings.

5.5.2. Request Air Miles. We now turn to the effect of remapping interval on the cost (in terms of average request air miles) of content delivery. Figures 14(a) and 14(b) show the results. An immediate and seemingly counter-intuitive observation is that costs generally decrease for larger remapping intervals. A closer inspection, however, reveals that this is due to the fact that less frequent remappings miss overload conditions and do not rebalance the load by using suboptimal (e.g., more distance but less loaded) servers. Indeed, the last 6-hour trace period reported in these graphs correspond to the period of the highest load; as the load increases, higher values of Δ retain a proximity-driven mapping from a less-loaded condition longer. This has especially pronounced effect for extremely large deltas, such as $\Delta = 6$ hours, when no remapping occurs after the load increase. Note also that these graphs reflect the costs for successful requests only. We will see how longer remapping intervals affect over-capacity requests that follow.

The comparison of different scenarios in Figures 14(a) and 14(b) reveals no further surprises. ALB-A has lower cost than ALB-O for a given server capacity, and a lower-capacity scenario has higher cost than the same scheme with higher-capacity. This is natural, since ALB-A optimizes cost at every remapping interval while ALB-O only when there is a compelling reason to do so. Also, the lower the capacity of servers, the more often the system must change mappings to relieve overloaded servers at the expense of increased costs.

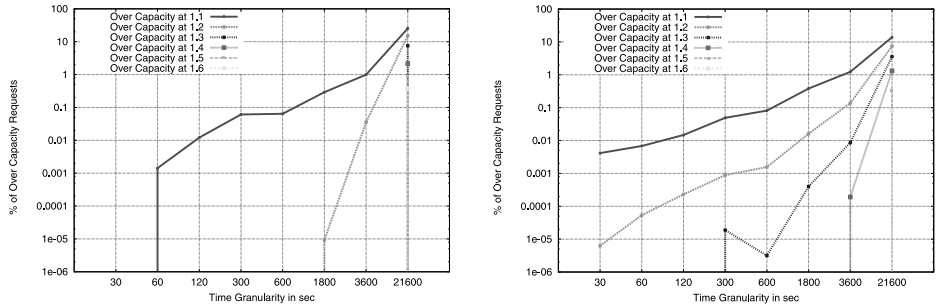
5.5.3. Over-Capacity Requests. Finally, we consider the effect of remapping interval on over-capacity requests. Intuitively, larger remapping intervals must lead to more over-capacity requests as the scheme would miss overload conditions between remappings. The results are shown in Figures 15(a) and 15(b). They confirm the above intuition; however, for the 100% capacity scenario, they show that no schemes exhibit any dropped requests until the remapping interval reaches 6 hour. Coupled with the previous results, this might suggest that very large values of delta, in the order of hours, should be used as it decreases connection disruption without increasing the costs and dropped requests.

However, in setting the physical capacity limit to be 1.6 times the server capacity used by the algorithms, we provisioned a significant slack between the load level where algorithms attempts to rebalance load and the level when a request is dropped. Consequently, Figures 16(a) and 16(b) show the behavior of the ALB-O algorithm when the



(a) Small objects group (b) Large files group

Fig. 15. The effect of remapping interval on dropped requests (common 6-hour trace period).



(a) Small objects group (b) Large files group

Fig. 16. The effect of over-provisioning on over-capacity requests (common 6-hour trace period).

servers are provisioned with a smaller capacity slack. In these experiments, we use 100% server capacity and do not drop over-capacity requests.

For the small files group, we still do not see over-capacity requests until the slack is reduced to 1.2. At 1.2 over-provisioning, the over-capacity requests appear only when remapping interval reaches 30 minutes. With over-provisioning factor of 1.1, over-capacity requests appear at 1-minute remapping interval and grow rapidly for larger intervals.

In the large files group, reducing over-provisioning from the factor of 1.6 to 1.1 increases the over-capacity requests more smoothly. At x1.3 over-provisioning, over-capacity requests appear at remapping intervals of 5 minutes and higher. Less slack leads to over-capacity requests at deltas as small as 30 seconds. Again, once appear, over-capacity requests increase for longer the remapping intervals (with one exception in Figure 16(b), which we consider an aberration).

Overall, these results show the intricacies in tuning the system. Clearly, provisioning a larger capacity slack allows one to reduce frequency of remappings: indeed, the proximity factor in remapping decisions does not change, and the load factor becomes less significant with the increased slack. This also results in lower deliver cost, as the system rarely sends traffic to nonproximal servers because of overload. Less slack requires more frequent remappings and can result in higher delivery cost. A proper choice of the remapping interval in this case requires careful analysis of the workload similar to the one performed in this article.

6. RELATED WORK

Our architecture uses IP anycast to route HTTP requests to edge servers, with a subsequent HTTP redirection of requests for particularly large downloads. Our parallel work addresses the increased penalty of disrupted connections in CDNs that deliver streaming content and very large objects [Al-Qudah et al. 2009]. That work proposes to induce connection disruption as a way to reassign a client to a different edge server if load conditions change during the long-running download. That work is complimentary to our present approach: the latter can use this technique instead of HTTP redirects to deal with long-running downloads.

IP anycast has been used in some components of content delivery. In particular, Limelight CDN [Limelight 2010] utilizes anycast to route DNS queries to their DNS servers: each data center has its own DNS server, with all DNS servers sharing the same address. Whenever a DNS query arrives to a given DNS server, the server resolves it to an edge server co-located in the same data center. Thus, even though edge servers use unicast addresses, Limelight sidesteps the need to determine the nearest data center for a client, leveraging the underlying BGP routing fabric for data center selection. However, similar to the DNS-based CDNs, clients are still directed to the data center that is nearest to client's DNS server and not the client itself.

CacheFly [CacheFly] is to our knowledge the first CDN utilizing the anycast technology for content download itself. Our approach targets a different CDN environment: while CacheFly follows the co-location approach with edge servers obtaining connectivity from multiple ISPs, we assume a single-AS CDN where the operator has control over intra-platform routing. No information on which (if any) load reassignment mechanisms CacheFly uses is available.

Utilizing IPv6 for anycast request routing in CDNs has been independently proposed in Szymaniak et al. [2007] and Acharya and Shaikh [2002]. Our work shows that anycast can be used for CDN content delivery even in current IPv4 networks.

We formulate our load balancing problem as *generalized assignment problem* (GAP). One related problem is the multiple knapsack problem (MKP), where we are given a set of items with different size and profit, and the objective is to find a subset of items that allows a feasible packing in the bins without violating bin capacity and maximize the total profit. Chekuri and Khanna [2000] present a polynomial time approximation scheme (PTAS) for this problem. MKP is a special case where the profit for an item is the same for all bins, which cannot handle our setting since a cost of serving a request in a CDN varies depending on the server. Aggarwal et al. [2003] consider the problem of load rebalancing. Given the current assignment of request-server pairs, they focus on minimizing the completion time of queued requests by moving up to k requests to different servers and present a linear-time 1.5 approximation algorithm to this NP-hard problem. While the limited amount of rebalancing is relevant to our case to reduce ongoing session disruptions, our work has a different objective of maintaining server load under capacity.

Another related problem is the facility location problem, where the goal is to select a subset of potential sites to open facilities and minimize the sum of request service cost and opening costs of the facilities [Shmoys et al. 1997]. This problem is more relevant in the provisioning time scale, when we can determine where to place CDN servers for a content group. In our setting, we are given a set of CDN servers and load-balance between them without violating the capacity constraint.

Current CDNs predominantly use DNS-based load balancing, and a number of load-balancing algorithms for this environment have been proposed in research [Biliris et al. 2001; Cardellini et al. 2003; Colajanni et al. 1998; Kwan et al. 1995; Rabinovich et al. 2003] and made available in commercial products [3-DNS Controller 2005; Cisco

2009; ServerIron 2008]. Since load-balancing is done at the application layer, these algorithms are able to make load balancing decisions at the granularity of individual DNS requests. For example, Biliris et al. [2001] uses a simple algorithm of resolving each request to the nearest non-overloaded server, while Colajanni et al. [1998] proposes intricate variations in DNS response TTL to control the amount of load directed to the server. These algorithms are not applicable in our environment where load-balancing decisions are at the drastically coarser granularity of the entire PEs.

Content delivery networks can benefit from peer-to-peer content sharing, which can be used to share cached content either among CDN servers (and thus reduce the need to forward requests to the origin server) [Freedman et al. 2004] or among users' local caches directly [Iyer et al. 2002]. These approaches are complimentary to and can be used in conjunction with our architecture. There has also been a rise in the use of peer-to-peer content delivery as an alternative to traditional content delivery networks, with BitTorrent and other P2P platforms providing examples of this approach. The general scalability of this style of content delivery is considered in Stutzbach et al. [2005]. Our work targets traditional CDNs, which offer their subscribers content delivery from a dedicated commercially operated platform with tight control and usage reporting.

7. FUTURE WORK

The approach we discuss in this article leads to a number of open questions for future work. The focus of this article is on making the case that anycast CDNs are a practical alternative to a DNS CDN. One obvious question, which we could not answer with the data available, is the quantitative side-by-side comparison between the two mechanisms. To answer this question, one needs to be able to evaluate the difference between the "external" proximity from the client to the end of the CDN network given in the CDN and anycast CDN, and how it affects the performance from the client perspective.

A consequence of our network provider perspective is that our cost metric reflects the desire to optimize request processing within the CDN's network. An important question for future work is how this affects the performance from the application perspective. In particular, as mentioned in Section 4.1, the remapping algorithm could take into account the overall reverse routing path to the clients, but the current approach can do so only at the granularity of the entire PEs. In principle, repinning could be done at the individual ingress interface level of the PEs; this however requires more complexity in the PEs, as well as significant increase of the search space for the optimization algorithm. A detailed study would be required to understand the potential benefits and tradeoffs involved.

The remapping algorithm in our architecture runs periodically, and we examined implications of various values of the remapping interval. However, our work assumes that the interval is statically configured. An interesting question is if any benefits could be derived from adjusting this interval dynamically, perhaps in response to the observed instability of the demand.

Another question has to do with the limitation that our current architecture targets CDNs that belong to a single autonomous system. In an ongoing work, we are currently extending our architecture to CDNs that spans multiple autonomous systems. In this work, we are defining APIs between the CDN and the ISPs it utilizes; the CDN then employs a combination of a set of anycast addresses and the APIs to control traffic distribution between the autonomous systems. Finally, our current approach focuses on small object downloads and file transfers. The applicability of IP anycast to other content types, notably steaming media, is another direction for future investigation.

8. CONCLUSION

New route control mechanisms, as well as a better understanding of the behavior of IP Anycast in operational settings, allowed us to revisit IP Anycast as a CDN redirection mechanism. We presented a load-aware IP Anycast CDN architecture and described algorithms that allow redirection to utilize IP Anycast's inherent proximity properties, without suffering the negative consequences of using IP Anycast with session based protocols. We evaluated our algorithms using trace data from an operational CDN and showed that they perform almost as well as native IP Anycast in terms of proximity, manage to keep server load within capacity constraints and significantly outperform other approaches in terms of the number of session disruptions.

In the future we expect to gain experience with our approach in an operational deployment. We also plan to exploit the capabilities of our architecture to avoid network hotspots to further enhance our approach.

REFERENCES

3-DNS Controller. 2005. F5 Networks. http://support.f5.com/kb/en-us/archived_products/3-dns/.

ACHARYA, A. AND SHAIKH, A. 2002. Using mobility support for request routing in IPv6 CDNs. In *Proceedings of the 7th International Web Content Caching and Distribution Workshop (WCW)*.

AGGARWAL, G., MOTWANI, R., AND ZHU, A. 2003. The load rebalancing problem. In *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'03)*. ACM, New York, 258–265.

AL-QUDAH, Z., LEE, S., RABINOVICH, M., SPATSCHECK, O., AND VAN DER MERWE, J. E. 2009. Anycast-aware transport for content delivery networks. In *Proceedings of the International Conference on the World-Wide Web (WWW)*. 301–310.

ATT ICDS 2010. Intelligent content distribution service. http://www.business.att.com/enterprise/Service/digital-media-solutions-enterprise/content-distribution-enterprise/content-distribution-service-enterprise/state=Ohio/.

BALLANI, H., FRANCIS, P., AND RATNASAMY, S. 2006. A measurement-based deployment proposal for IP anycast. In *Proceedings of the ACM/USENIX Internet Measurements Conference*.

BARBIR, A., CAIN, B., DOUGLIS, F., GREEN, M., HOFMANN, M., NAIR, R., POTTER, D., AND SPATSCHECK, O. 2003. Known content network (CN) request-routing mechanisms. RFC 3568.

BILIRIS, A., CRANOR, C., DOUGLIS, F., RABINOVICH, M., SIBAL, S., SPATSCHECK, O., AND STURM, W. 2001. CDN brokering. In *Proceedings of the 6th International Workshop on Web Caching and Content Distribution*.

CACHEFLY. CacheFly: Besthop global traffic management. http://www.cachefly.com/video.html.

CARDELLINI, V., COLAJANNI, M., AND YU, P. S. 2003. Request redirection algorithms for distributed web systems. *IEEE Trans. Parall. Distrib. Syst. 14,* 4, 355–368.

CHA, M., KWAK, H., RODRIGUEZ, P., AHN, Y.-Y., AND MOON, S. 2007. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proceedings of the ACM/USENIX Internet Measurement Conference*.

CHEKURI, C. AND KHANNA, S. 2000. A PTAS for the multiple knapsack problem. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*. ACM, 213–222.

CISCO. 2009. Cisco GSS 4400 series global site selector appliances. http://www.cisco.com/en/US/products/hw/contnetw/ps4162/index.html.

COLAJANNI, M., YU, P. S., AND CARDELLINI, V. 1998. Dynamic load balancing in geographically distributed heterogeneous web servers. In *Proceedings of the IEEE International Conference on Device Circuit and Systems (ICDCS)*. 295–302.

DUFFIELD, N., GOPALAN, K., HINES, M. R., SHAIKH, A., AND VAN DER MERWE, J. E. 2007. Measurement informed route selection. In *Proceedings of the Passive and Active Measurement Conference.* Extended abstract.

FREEDMAN, M. J., FREUDENTHAL, E., AND MAZIÈRES, D. 2004. Democratizing content publication with coral. In *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 239–252.

HARDIE, T. 2002. Distributing authoritative name servers via shared unicast addresses. IETF RFC 3258.

IYER, S., ROWSTRON, A., AND DRUSCHEL, P. 2002. Squirrel: A decentralized peer-to-peer web cache. In *Proceedings of the ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*. 213–222.

JUNG, J., KRISHNAMURTHY, B., AND RABINOVICH, M. 2002. Flash crowds and denial of service attacks: Characterization and implications for CDNs and web sites. In *Proceedings of 11th World Wide Web Conference*.

KING, A. 2006. The Average web page. `http://www.optimizationweek.com/reviews/average-web-page/`.

KWAN, T. T., MCCRATH, R., AND REED, D. A. 1995. NCSA's world wide web server: Design and performance. *IEEE Comput. 28,* 11, 68–74.

LIMELIGHT. 2010. `http://www.limelightnetworks.com/platform/cdn/`.

MAO, Z., CRANOR, C., DOUGLIS, F., RABINOVICH, M., SPATSCHECK, O., AND WANG, J. 2002. A precise and efficient evaluation of the proximity between web clients and their local DNS servers. In *Proceedings of the USENIX Annual Technical Conference*.

PANG, J., AKELLA, A., SHAIKH, A., KRISHNAMURTHY, B., AND SESHAN, S. 2004. On the Responsiveness of DNS-based network control. In *Proceedings of the Internet Measurement Conference (IMC)*.

RABINOVICH, M., XIAO, Z., AND AGGARWAL, A. 2003. Computing on the edge: A platform for replicating Internet applications. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*.

REIBMAN, A., SEN, S., AND VAN DER MERWE, J. 2004. Network monitoring for video quality over IP. In *Proceedings of the Picture Coding Symposium*.

SERVERIRON. 2008. ServerIron DNSProxy. Fountry networks. `http://www.brocade.com/products/all/switches/index.page`.

SHAIKH, A., TEWARI, R., AND AGRAWAL, M. 2001. On the effectiveness of DNS-based server selection. In *Proceedings of the IEEE Annual Conference on Computer Communications (INFOCOM)*. 1801–1810.

SHMOYS, D. AND TARDOS, E. 1993. An approximation algorithm for the generalized assignment problem. *Math. Prog. 62*, 461–474.

SHMOYS, D. B., TARDOS, E., AND AARDAL, K. 1997. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing (STOC'97)*. ACM, New York.

STUTZBACH, D., ZAPPALA, D., AND REJAIE, R. 2005. The scalability of swarming peer-to-peer content delivery. In *Proceedings of Networking*. 15–26.

SZYMANIAK, M., PIERRE, G., SIMONS-NIKOLOVA, M., AND VAN STEEN, M. 2007. Enabling service adaptability with versatile anycast. *Concurr. Computat. Practice Exp. 19,* 13, 1837–1863.

VAN DER MERWE, J., SEN, S., AND KALMANEK, C. 2002. Streaming video traffic: Characterization and network impact. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution (WCW)*.

VAN DER MERWE, J., GAUSMAN, P., CRANOR, C., AND AKHMAROV, R. 2003. Design, implementation and operation of a large enterprise content distribution network. In *Proceedings of the 8th International Workshop on Web Content Caching and Distribution*.

VAN DER MERWE, J., CEPLEANU, A., D'SOUZA, K., FREEMAN, B., GREENBERG, A., KNIGHT, D., MCMILLAN, R., MOLONEY, D., MULLIGAN, J., NGUYEN, H., NGUYEN, M., RAMARAJAN, A., SAAD, S., SATTERLEE, M., SPENCER, T., TOLL, D., AND ZELINGHER, S. 2006. Dynamic connectivity management with an intelligent route service control point. In *Proceedings of the SIGCOMM Workshop on Internet Network Management (INM'06)*. ACM, New York, 29–34.

VERKAIK, P., PEI, D., SCHOLL, T., SHAIKH, A., SNOEREN, A., AND VAN DER MERWE, J. 2007. Wresting control from BGP: Scalable fine-grained route control. In *Proceedings of the USENIX Annual Technical Conference*.

Received October 2009; revised April 2010; accepted September 2010