Anycast CDNs Revisited

Hussein A. Alzoubi* Seungjoon Lee† Michael Rabinovich*
Oliver Spatscheck† Jacobus Van der Merwe†
* Case Western Reserve University, Cleveland, OH 44106 hussein.alzoubi@case.edu, misha@eecs.case.edu
†AT&T Labs - Research, Florham Park, NJ 07932 fslee, spatsch, kobusg@research.att.com

ABSTRACT

Because it is an integral part of the Internet routing apparatus, and because it allows multiple instances of the same service to be "naturally" discovered, IP Anycast has many attractive features for any service that involve the replication of multiple instances across the Internet. While briefly considered as an enabler when content distribution networks (CDNs) first emerged, the use of 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. Prompted by recent developments in route control technology, as well as a better understanding of the behavior of IP Anycast in operational settings, we revisit this decision and propose a load-aware IP Anycast CDN architecture that addresses these concerns while benefiting from inherent IP Anycast features. 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 session 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

H.4.m [**Information Systems**]: Miscellaneous; D.2 [**Software**]: Software Engineering; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Performance, Algorithms

Keywords

CDN, autonomous system, anycast, routing, load balancing

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, to more traditional Web objects and streaming events and more recently user generated video content [4]. Because of the often bursty nature of demand for such content [9], and because content owners require their content to be highly available and be delivered in timely manner without impacting presentation quality [13], 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 [2]. 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" [1]. 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 [1]. Indeed most commercial CDNs make use of DNS to perform redirection. DNS based redirection is not a panacea, however. In DNS-based redirection the actual eyeball request is not redirected, rather the local-DNS server of the eyeball is redirected [1]. This has several implications. First, not all eyeballs are in close proximity to their local-DNS servers [11, 14], 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. This problem can be mitigated significantly by having the DNS system make use of very short time-to-live (TTL) values. However, caching of DNS queries by local-DNS servers and especially certain browsers beyond specified TTL means that this remains an issue [12]. Finally, despite significant research in this area [7, 5, 16], the complexity of knowing the "distance" between any two IP addresses on the Internet, needed to find the closest CDN node, remains a difficult problem.

In this work we revisit IP Anycast as redirection technique, which, although examined early in the CDN evolution process [1], was considered infeasible at the time. IP Anycast refers to the ability of the IP routing and forward-

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

ing architecture to allow the same IP address to be assigned to multiple different 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 [8]. 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 a 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 optimal 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 anycast endpoint 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.

In this paper we present our work on a load-aware IP Anycast CDN architecture. 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 to be informed by external intelligence [19, 6, 20]. Second, recent Anycast based measurement work [3] shed light on the behavior of IP Anycast, as well as the appropriate way to deploy IP Anycast to facilitate proximal routing.

When selecting between multiple hosts with the same Anycast IP address, route control mechanisms allow both CDN node load and network conditions to be taken into account, which addresses two of the concerns listed above (i.e., quality of service and load-awareness). Route control also partially deals with the concern about resetting sessions because of route changes. We note that in practice there are two variants 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 of these problems preventing unnecessary switching between Anycast addresses within the network. As for route changes outside of the IP Anycast network, recent Anycast measurement work [3] has shown that most IP prefixes exhibit very good affinity, i.e., would be routed along the same path towards the Anycast enabled network.

The main contributions of our work 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).

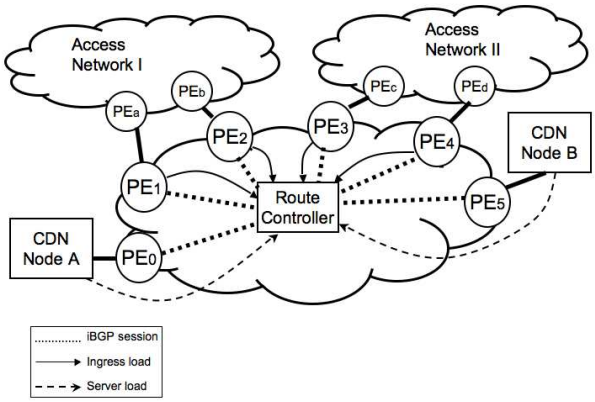


Figure 1: Load-aware Anycast CDN Architecture

- 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 (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 the paper, 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 paper 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 [3] has shown this to be the ideal type of deployment to ensure good proximity properties[2].

Figure 1 also shows the Route Controller function that is central to our approach [19, 20]. The Route Controller exchanges routes with provider edge (PE) routers in the CDN

[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.

provider network. As such, the Route Controller can influence the routes selected by these PEs. For our purposes, the Route Controller takes two other inputs, namely, ingress load from the PEs at the edge of the network, and server load from the CDN nodes for which it is performing redirection.

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 who 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 addresses become reachable via the whole 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). The Route Controller can manipulate ingress load(s) worth of *request* traffic in order to effect server *response* load. In other words, there is an implicit assumption in our approach that there is a direct correlation between the request (ingress) traffic and the resulting response server traffic. Fortunately, as we will show in Section 5, this assumption holds because CDN providers typically group like content together (e.g., large or small objects) so that in the aggregate each request "weighs" more or less the same.

2.2 Objectives and Benefits

Given the architecture described above, the goals of the redirection algorithm driving the Route Controller are as follows: (i) To utilize the "natural" IP Anycast proximity properties to reduce the distance traffic is carried in the network. (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 redirection granularity offered by our Route Control approach is at the PE level. For large Tier-1 ISPs the number of PEs typically count 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 benefit in practice.

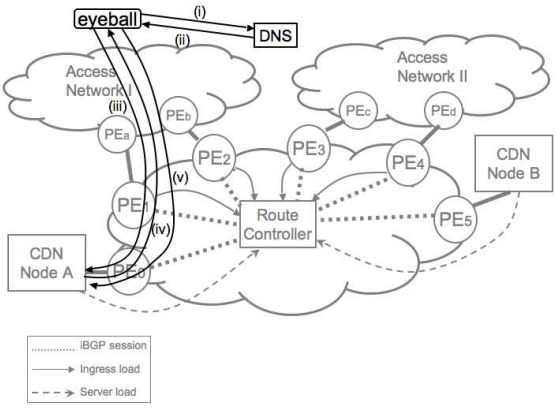


Figure 2: Application level redirection for long-lived sessions

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 paper.

2.3 Additional Functions

Dealing with long lived sessions: Despite increased distribution of rich media content via the Internet, the average Web object size remains relatively small [10]. 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, e.g., streaming or large file download [18]. (Both of these expectations are validated with our analysis 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 which 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) [17] 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 small Web objects, it is less of a concern for long lived sessions such as large file download. This is why we use HTTP redirection only for large file downloads so that the startup overhead is amortized.

Dealing with network congestion: As described above, 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 [19]. 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 [19]. We leave the full development of these mechanisms for future work.

3. REDIRECTION ALGORITHM

As described above, our redirection algorithm 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 requests per time unit. A request enters the system through one of n ingress PEs, and each ingress PE j contributes r_j amount of requests per time unit. 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 Program [15].

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

where indicator variable x_{ij}=1 iff server i serves PE j, and x_{ij}=0 otherwise. 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 [15]. Recall that an α-approximation algorithm always finds a solution that is guaranteed to be at most a times the optimum.

Shmoys and Tardos [15] present an approximation algorithm (called ST-algorithm in this paper) for GAP, which involves a relaxation of the integrality constraint and a rounding based on a fractional solution to the LP relaxation. Specifically, given total cost C, their algorithm decides if there exists a solution with total cost at most C. If so, it also finds a solution whose total cost is at most C and the load on each server is at most $S_i + \max r_j$.

3.2 Minimizing Cost

Note that ST-algorithm can lead to server overload, 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). In our system, we use the following post-processing on the solution of ST-algorithm to off-load overloaded servers and find a feasible solution without violating the constraint.

We first identify most overloaded server i, and then among all the PEs served by i, find out the set of PEs F (starting from the smallest in load) where server i's load becomes below the capacity S_i after off-loading F. Then, starting with the largest in size among F, we off-load each PE j to a server with enough residual capacity, as long as the load on server i is above S_i. (If there are multiple such servers for j, we choose the minimum-cost one in our simulation although we can employ different strategies such as best-fit.) We repeat this if there is an overloaded server. 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. Note that the load comparison between i and t ensures the maximum overload in the system decreases over time, which avoids cycles.

3.3 Minimizing Connection Disruption

While the above algorithm 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 in which we attempt to reassign PEs to different servers *only* when we need to off-load one or more overloaded servers. Specifically, we only consider overloaded servers and PEs assigned to them when calculating a re-assignment, while keeping the current mapping for PEs whose server is not overloaded. Even for the PEs assigned to overloaded servers, we set priority such that the PEs prefer to stay assigned to their current server as much as possible. Then, we find a set of PEs that (1) are sufficient to reduce the server load below maximum capacity and (2) minimize the disruption penalty due to the reassignment. We can easily find such a set of PEs and their new server by fixing some of input parameters to ST-algorithm and applying our post-processing algorithm on the solution.

Since this algorithm reassigns PEs to different servers only in overloaded scenarios, it can lead to sub-optimal operation even when the request volume has gone down significantly and a simple proximity-based routing yields a feasible solution with lower cost. One way to address this is to exploit the typical diurnal pattern and reassign the whole set of PEs at a certain time of day (e.g., 4am every day). Another possibility is to compare the current mapping and the best possible mapping at that point, and initiate the reassignment if the difference is beyond a certain threshold (e.g., 70%). In our experiments, we use employ the latter approach.

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. METHODOLOGY

4.1 Data Set

We obtained server logs for a weekday in July, 2007 from a production CDN which we utilized in a trace driven sim-

ulation of our algorithms. For our analysis we use two sets of content groups based on the characteristics of objects the servers serve: one set for *Small Web Objects*, and the other for *Large File Download*. Each log entry we use has detailed information about an HTTP request and response such as client IP, web server IP, request URL, request size, response size, etc. 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.

One set of inputs required by our algorithm is the request load arriving from each ingress PE. We use the number of ongoing requests (being served by a server) that have arrived from each ingress PE j as request load r_j. To determine the number of request counts arriving from each PE, 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. Then, we use the service response time to determine whether a flow is currently being served.

One of our objectives is to minimize the network bandwidth usage by service requests and responses. To reflect this, we first obtain the distance matrix d_{ij} between server i and ingress PE j in terms of air miles (which is known to be highly correlated with link usage within an autonomous system). Then, we use the product $r_j d_{ij}$ as the cost c_{ij} of serving requests from PE j at server i. To determine the server capacity in our experiments, we first analyze the log to determine the maximum load during the entire time period in the log. Then, we use a value slightly larger than the maximum load and divide it by the number of servers. This leads to a high-load scenario for peak time, while we have sufficient server capacity to handle all the requests. Another interesting aspect is to understand how each scheme performs in scenarios with different loads (e.g., light or medium load), which we plan to investigate in our future work.

4.2 Simulation Environment

We used CSIM (`http://www.mesquite.com`) 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 ingress PE flows, which are implemented as CSIM processes. For each flow that arrives we determine the ingress PE j, the response time t, and the response size l. We assume that the server responds at a constant rate calculated as the response size divided by the response time. In other words, each flow causes a server to serve data at the constant rate of l/t for t seconds. Multiple flows from the same PE j can be active simultaneously on server S. Furthermore, multiple PEs can be served by the same facility at the same time.

Our servers (facilities) are configured to have infinite capacity and very large bandwidth. This means that our servers in the simulation can handle any size of load. The redirection algorithm decides whether or not servers are overloaded, depending on the number of active connections monitored at the time the algorithm starts. We keep track of the number of active connections at the server side and ingress PEs side to calculate the current server load and PE load r_j.

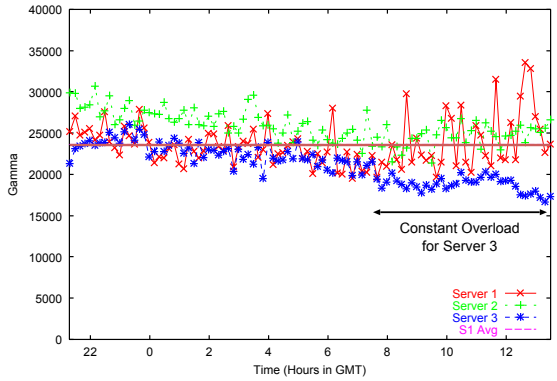


Figure 3: Correlation between server load and incoming traffic volume. The straight line is the average value for Server 1.

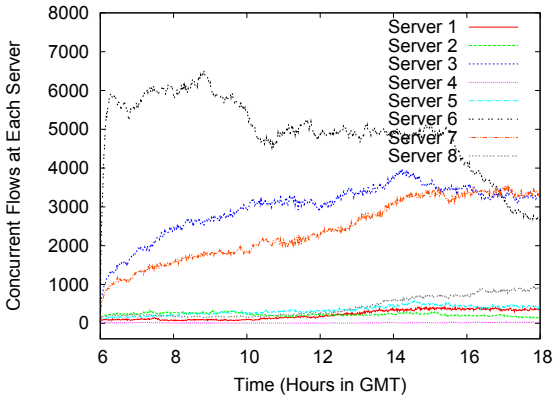
4.3 Schemes and Metrics for Comparison

We experiment with the following schemes and compare the performance:

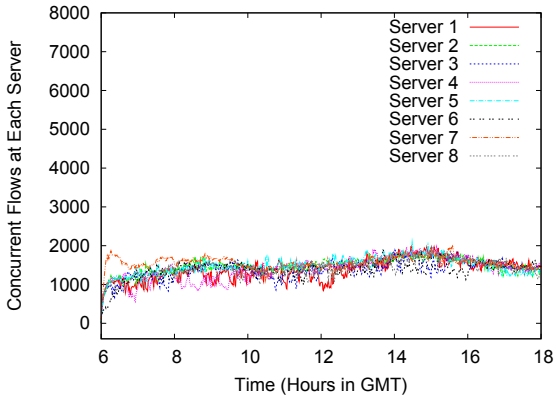
- Simple Anycast (SAC): This is "native" Anycast, which represents an idealized proximity routing scheme, where each request is served at the geographically closest server.
- Simple Load Balancing (SLB): This scheme minimizes the difference in load among all servers without considering the cost.
- Advanced 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 only reassigns PEs currently mapped to overloaded servers to minimize connection disruptions as described in Section 3.3.

In SAC, each PE is statically mapped to a server, and there is no change in mapping. SLB and ALB-A recalculate the mapping every two minutes based on the current load. While ALB-O also analyzes the current load and mapping every two minutes, it re-calculates the mapping (usually only for a subset of PEs) only if an overloaded server exists.

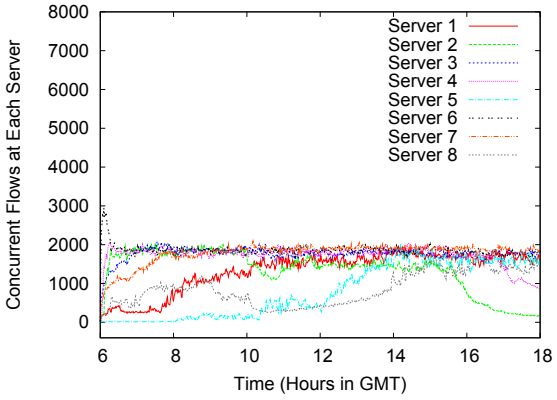
For performance comparison, we first use the *number of ongoing connections* and *service data rate* at each server. A desirable scheme should keep the number below the capacity limit all the time. We also examine the *average miles* a request traverses within the CDN provider network before reaching a server. A small value for this metric denotes small network link usage in practice. We finally use the *number of connection disruptions* due to re-mapping. SAC uses a static mapping and does not experience disconnections due to re-mapping. However, since it is not load-aware, the number of connections for one server can go over the maximum limit. In this case, we count the number of dropped connections. 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.



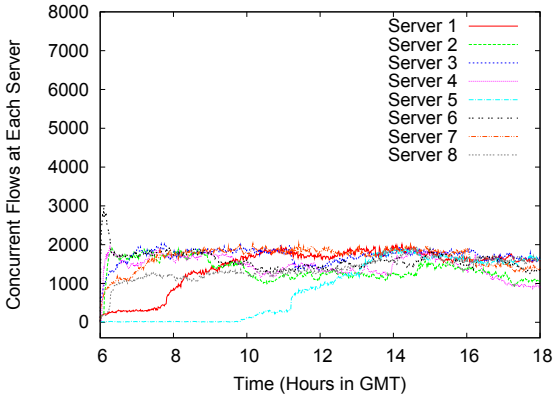
(a) SAC



(b) SLB



(c) ALB-A



(d) ALB-O

Figure 4: Number of concurrent connections for each scheme (Large file download)

5. EXPERIMENT RESULTS

In this section, we present our simulation results. We first investigate the validity of our correlation assumption between request load and response load in Section 5.1. We present the evaluation of our redirection algorithms in Section 5.2.

5.1 Request Load vs. Response Load

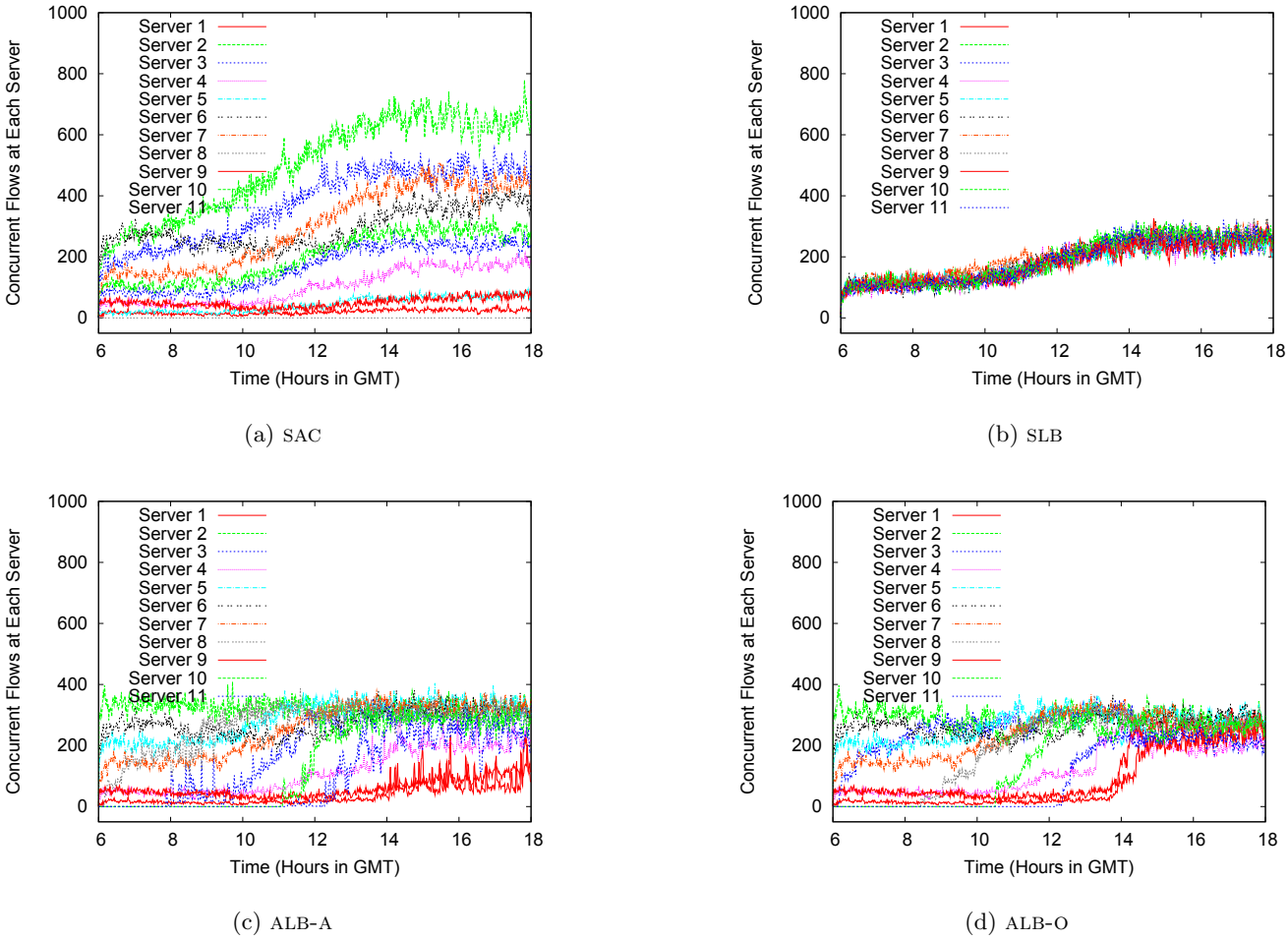
As explained in Section 2, our approach relies on redirecting traffic from ingress PEs to effect the load on CDN servers. As such it relies on the assumption that there exists a linear correlation between *request load* and the resulting *response load* on the CDN server. The request packet stream takes the form of requests and TCP ACK packets flowing towards the CDN node.[3] Since there is regularity in the number of TCP ACK packets relative to the number of response (data) packets that are sent, we can expect correlation. In this subsection, we investigate data from an operational CDN to understand whether this key assumption holds.

We study a CDN content group for large file downloads and obtain per-server load data from the CDN nodes in question. For the network ingress load data, we obtained NetFlow records to count the number of packets from each ingress PE to content servers for the corresponding period of time. (Due to incomplete server logs, we use 16+ hours of results in this subsection.)

Consider the following value $\gamma = \alpha S/r$, where S is the maximum capacity of the CDN node (specified as the maximum bytes throughput in the configuration files of the CDN), α is node's utilization in [0,1] (as reported by the CDN's load monitoring apparatus and referred to as server load below), and r is the aggregate per-second request count from ingress PEs coming to the server. Note that a constant value of γ signifies a perfect linear correlation between server load α and incoming traffic volume r.

In Figure 3, we plot γ for multiple servers in the content group. We observe that γ is quite stable across time, although it shows minor fluctuation depending on the time of day. We further note that server 2 has 60% higher capacity than the other two servers, and the server load ranges between 0.3 and 1.0 over time. This result shows that the maximum server capacity is an important parameter to determine γ. Although Server 3 exhibits a slight deviation after 36000 seconds due to constant overload (i.e., $\alpha = 1$), γ stays stable before that period even when α is often well

[3]We limit our discussion here to TCP based downloads.



(a) SAC

(b) SLB

(c) ALB-A

(d) ALB-O

Figure 5: Number of concurrent connections for each scheme (Small web object)

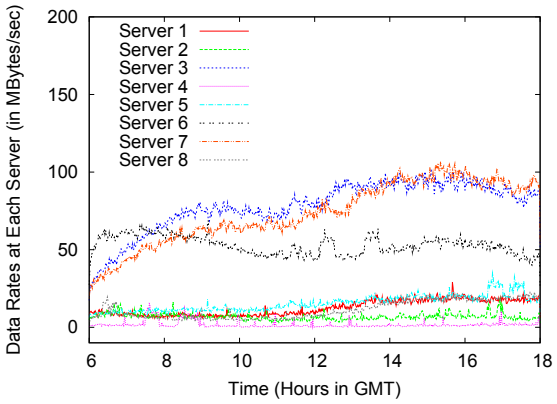
above 0.9. The data therefore indicates that our assumption of linear correlation holds.

5.2 Redirection Evaluation

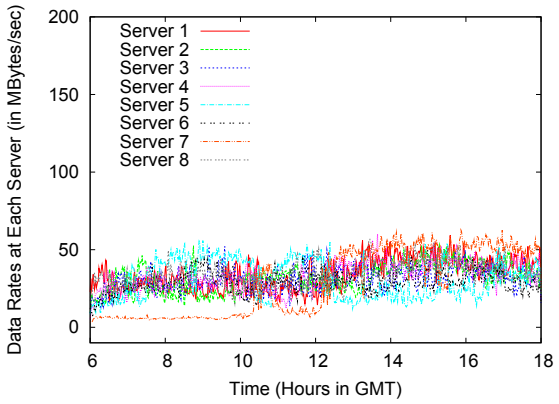
In our evaluation we consider in turn the server load, the number of miles traffic were carried and the number of session disruptions that resulted for each of the redirection schemes presented in Section 4.3.

Server Load: We first present the number of connections at each server using the results for large file downloads. In Figure 4, we plot the number of concurrent flows at each server over time. For the clarity of presentation, we use the points sampled every minute. Since SAC does not take load into account, but always maps PEs to a closest server, we observe from Figure 4(a) that the load at only a few servers grows significantly, while other servers get very few flows. For example, at 8am, 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. In Figure 4(b), we observe that SLB evenly distributes the load across servers. However, SLB does not take cost into account and can potentially lead to high connection cost.

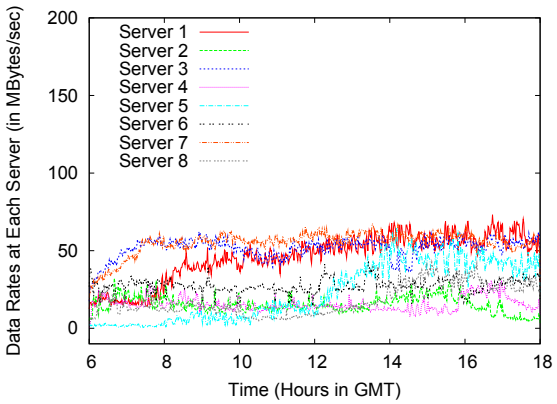
In Figure 4(c), we present the performance of ALB-A. Based on the maximum total number of concurrent requests, we set the capacity of each server to 1900. Unlike SLB, ALB-A (as well as ALB-O) does not balance the load among servers. This is expected because the main objective of ALB-A is to find a mapping that minimizes the cost as long as the resulting mapping does not violate the server capacity constraint. As a result, in the morning (around 7am), 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 3pm), 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 4(d)), while the change in connection count is in general more graceful. However, the difference becomes clear after the traffic peak is over (at around 4pm). 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 3pm). This property of ALB-O leads to less traffic disruption at the expense of increased overall cost. We further elaborate on this aspect later in this section.



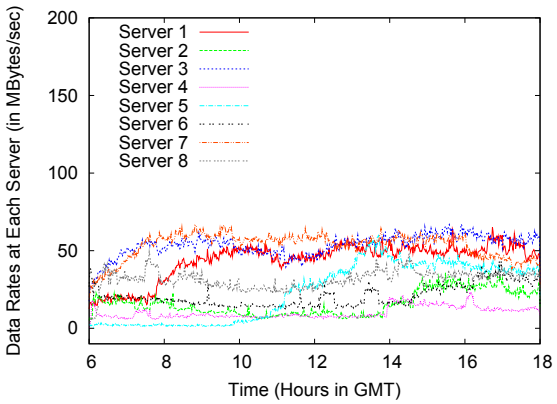
(a) SAC



(b) SLB



(c) ALB-A



(d) ALB-O

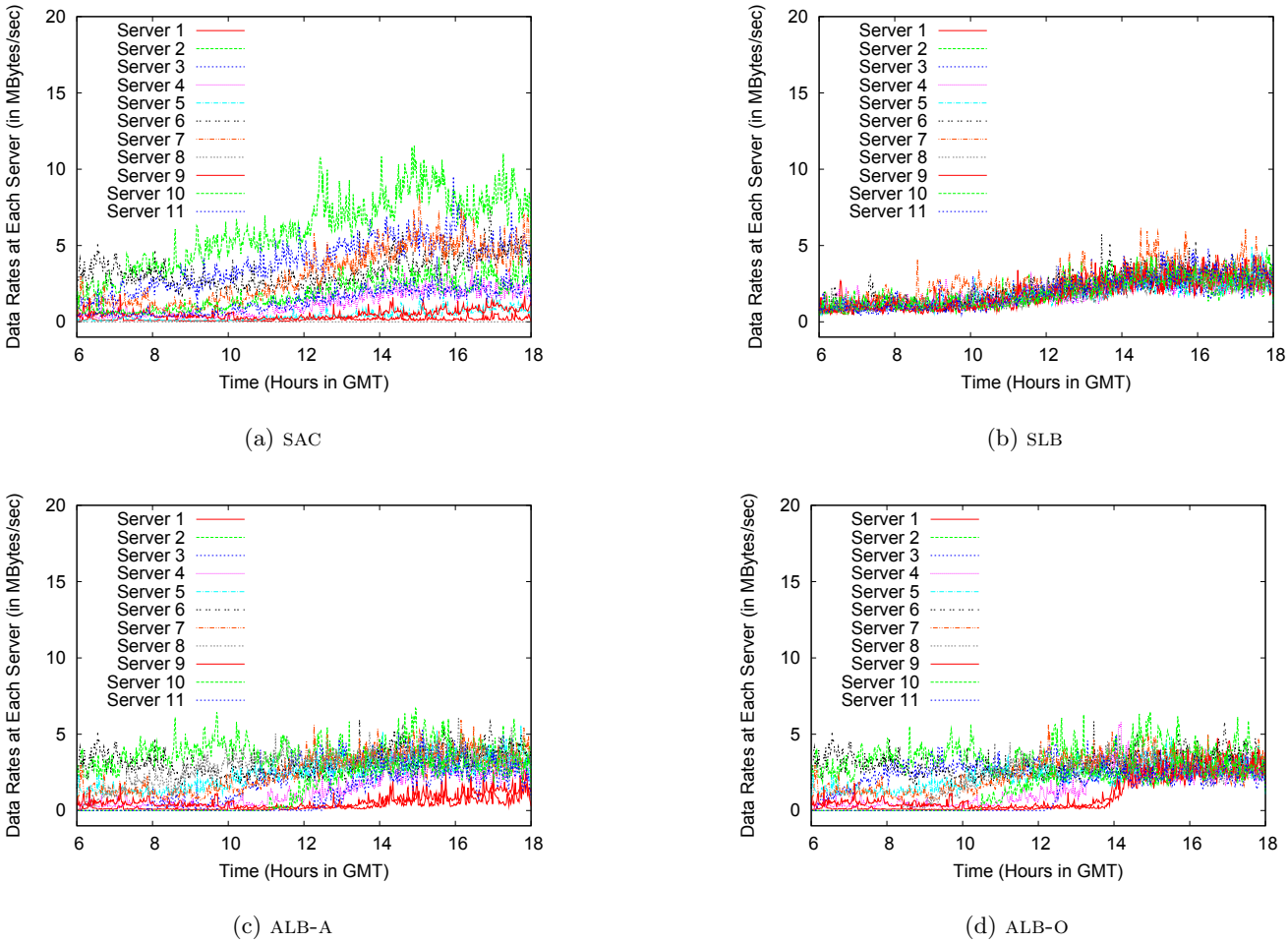
Figure 6: Service data rate for each scheme (Large file download)

In Figure 5, 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 service rate of each server in Figure 7 (and Figure 6 for large file download). We observe that there is strong correlation between the number of connections (Figures 4 and 5) and data rates (Figures 6 and 7).

Connection Disruption: SLB, ALB-A, and ALB-O can disrupt active connections due to reassignment. In this subsection, we investigate the impact of each scheme on connection disruption. We also study the number of dropped connections in SAC due to limited server capacity. In our experiments, we use sufficiently large server capacity for SAC so that the aggregate capacity is enough to process the total offered load: 2500 for large file download scenario, and 500 for small object scenario. Then, we count the number of connections that arrive to a server and find the server is operating at its capacity, which would be dropped in practice. While using a smaller capacity value in our experiments will lead to more connection drops in SAC, we illustrate that SAC can drop a large number of connection requests even when servers are relatively well provisioned.

In Figure 8, we present the percentage of disrupted connections for each scheme. For large file downloads, the service response time is longer, and in case of SLB, ALB-A, and ALB-O, a flow is more susceptible to re-mapping during its lifetime. This confirms our architectural assumption concerning the need for application level redirect for long lived sessions. From the figure, we observe that ALB-O clearly outperforms the other two by more than an order of magnitude. However, the disruption penalty due to re-mapping is much smaller than the penalty of connection drops due to overload in SAC. Specifically, SAC drops more than 18% of total requests in the large files scenario, which is 430 times higher than the number of disruptions by ALB-O. For small object contents, the disruption is less frequent due to the smaller object size and thus shorter service time. ALB-O again significantly outperforms the other schemes.

Average Cost: We present the average cost (as the average number of air miles) of each scheme in Figures 9 and 10. 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). SLB balances the load among servers without taking cost into account and leads to the highest cost. We observe in Figure 9 that ALB-A is nearly optimal in cost when the load is low (e.g., at 8am) because we can assign each PE to a closest server that



(a) SAC

(b) SLB

(c) ALB-A

(d) ALB-O

Figure 7: Service data rate for each scheme (Small web object)

has enough capacity. 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 re-mapped 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 8). This restricts the solution space for ALB-O compared to ALB-A, which subsequently increases the cost of ALB-O solution.

6. 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 which 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 near future we expect to gain operational 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.

Acknowledgment. We thank the reviewers for their valuable feedback. The work on this project by Alzoubi and Rabinovich is supported in part by the NSF Grant CNS-0615190.

7. REFERENCES

[1] A. Barbir et al.. Known CN Request-Routing Mechanisms. RFC 3568, July 2003.

[2] Alex Biliris, C. Cranor, F. Douglis, M. Rabinovich, S.Sibal, O. Spatscheck, and W. Sturm. CDN Brokering. Sixth International Workshop on Web Caching and Content Distribution, June 2001.

[3] H. Ballani, P. Francis, and S. Ratnasamy. A Measurement-based Deployment Proposal for IP Anycast. In *Proc. ACM IMC*, Oct 2006.

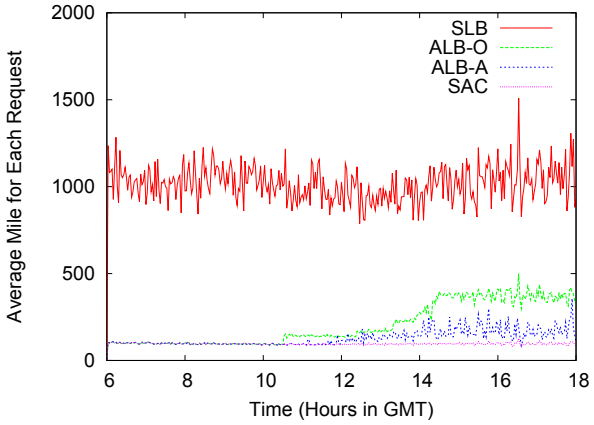


Figure 9: Average cost for each scheme (Small web object)

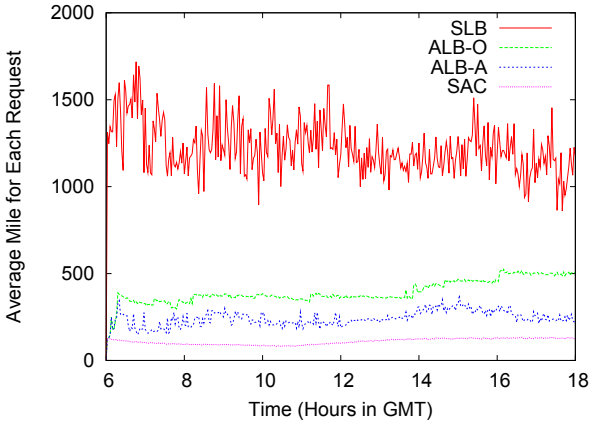


Figure 10: Average cost for each scheme (Large file download)

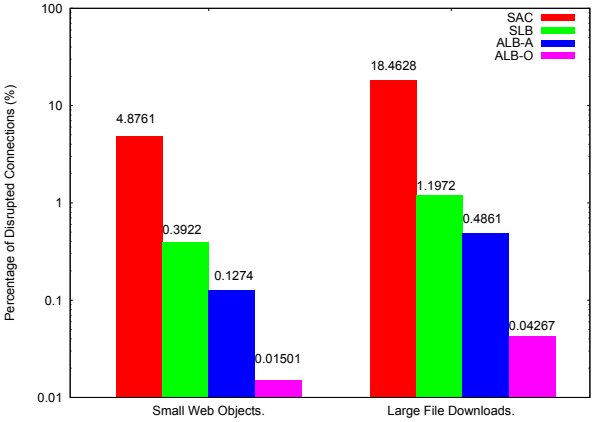


Figure 8: Connection disruption for each scheme (Y-axis in log scale)

[4] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. ACM/USENIX Internet Measurement Conference*, 2007.

[5] R. Cox, F. Dabek, F. Kaashoek, J. Li, and R. Morris. Practical Distributed Network Coordinates. In *ACM HotNets Workshop*, 2003.

[6] N. Duffield, K. Gopalan, M. R. Hines, A. Shaikh, and J. E. Van der Merwe. Measurement Informed Route Selection. Passive and Active Measurement Conference, April 2007. Extended abstract.

[7] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *SIGCOMM Internet Measurement Workshop (IMW 2002)*, 2002.

[8] T. Hardie. Distributing Authoritative Name Servers via Shared Unicast Addresses. IETF RFC 3258, 2002.

[9] J. Jung, B. Krishnamurthy, and M. Rabinovich. Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites. In *11th World Wide Web Conf.*, 2002.

[10] A. King. The Average Web Page. http://www.optimizationweek.com/reviews/average-web-page/, Oct. 2006.

[11] Z. Mao, C. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity between Web Clients and their Local DNS Servers. In *USENIX Annual Technical Conference*, 2002.

[12] J. Pang, A. Akella, A. Shaikh, B. Krishnamurthy, and S. Seshan. On the Responsiveness of DNS-based Network Control. In *Internet Measurement Conference (IMC)*, October 2004.

[13] A. Reibman, S. Sen, and J. Van der Merwe. Network Monitoring for Video Quality over IP. Picture Coding Symposium, 2004.

[14] A. Shaikh, R. Tewari, and M. Agrawal. On the Effectiveness of DNS-based Server Selection. In *IEEE INFOCOM*, pages 1801–1810, 2001.

[15] D. Shmoys and E. Tardos. An Approximation Algorithm for the Generalized Assignment Problem. *Mathematical Programming*, 62:461–474, 1993.

[16] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting Behind Akamai (Travelocity-Based Detouring). In *Proceedings of ACM SIGCOMM*, Sept 2006.

[17] J. Van der Merwe, P. Gausman, C. Cranor, and R. Akhmarov. Design, Implementation and Operation of a Large Enterprise Content Distribution Network. In *Proceedings of 8th International Workshop on Web Content Caching and Distribution*, Sept 2003.

[18] J. Van der Merwe, S. Sen, and C. Kalmanek. Streaming Video Traffic: Characterization and Network Impact. In *Proceedings of 7th Int. Workshop on Web Content Caching and Distribution (WCW)*, 2002.

[19] J. E. Van der Merwe Dynamic Connectivity Management with an Intelligent Route Service Control Point. In Proceedings of ACM SIGCOMM INM, Oct. 2006.

[20] P. Verkaik, D. Pei, T. Scholl, A. Shaikh, A. Snoeren, and J. Van der Merwe. Wresting Control from BGP: Scalable Fine-grained Route Control. In *2007 USENIX Annual Technical Conference*, June 2007.