Effective Peering for Multi-provider Content Delivery Services

Lisa Amini[a,b], Anees Shaikh[a], Henning Schulzrinne[b]
[a]IBM Research, Hawthorne, New York
[b]Columbia University, New York, New York
Email: aminil@us.ibm.com, aashaikh@watson.ibm.com, hgs@cs.columbia.edu

***Abstract*-- Peering allows service providers to handle traffic surges without over-provisioning, reduce the cost of dedicated infrastructure, and leverage the specialization and prices of partner providers. In this paper, we develop a peering system for multi-provider content delivery based on a cost-optimized peer selection algorithm. We formulate a cost model for evaluating competing peering strategies, and use measurement data collected from globally distributed network probe stations, large-scale Web sites, and existing service provider infrastructures to empirically evaluate proposed peering strategies. Our analysis shows that our peer selection algorithm is significantly more efficient than greedy alternatives, in terms of minimizing service cost and respecting network delay and server capacity thresholds, over a broad range of real-world scenarios.**

***Index terms*-- System design, simulations, network measurements, experimentation with real networks/testbeds.**

I. INTRODUCTION

Delivering large-scale network services requires a distributed computing and network infrastructure that provides a consistently high level of performance and reliability in the face of surges in demand, failures, and changes in customer requirements. Service providers who acquire, provision, and manage the service infrastructure must balance these requirements against the high costs of deploying customer-dedicated, over-provisioned resources. Service providers have addressed this problem to some extent by using shared infrastructures to multiplex resources between many customers, and thus improve utilization. To lower costs further, however, providers are increasingly interested in leveraging computing or network infrastructure from partners through peering mechanisms that allow provisioning and sharing of computing resources, and settlement and exchange of the resultant revenue. The customer's view remains that of a single provider (who we refer to as the *primary* provider), which may in fact consist of pooled resources from several providers distributed over a number of locations.

For example, instead of over-provisioning hosting centers for potential surges in demand, or distributing thousands of servers worldwide to minimize network latency, a provider could offload certain client workloads to partners. Cooperatives would also spawn secondary markets for the under-utilized resources of some providers. Federated, multi-provider architectures have been proposed as part of the CDI (Content Distribution Internetworking) effort [8], Grid computing [9], and 3G wireless clearinghouses [15]. Recent research has shown that multi-provider peering approaches offer cost benefits in a variety of scenarios [2], and can significantly improve reliability and performance [6][24].

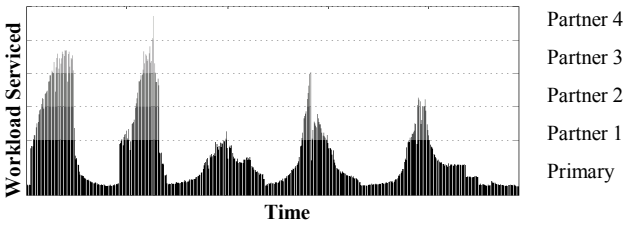


Figure 1: Example of cooperative, multi-service provider approach. While offered workload is within the capacity of the home infrastructure, the hosting service provider processes it. During peak periods, workload in excess of the home capacity is offloaded to one or more partner service providers.

In this paper, we focus on the use of cooperatives for content delivery services. In this scenario, the primary service provider may deploy a limited content hosting infrastructure in a single (or few) data centers, and at the same time, strike agreements with other content distribution service providers (CDSPs) who agree to serve customer content on behalf of the primary provider for a share of the revenue. As long as the client load can be handled by the primary provider, requests will be served from the origin since this maximizes revenue for the primary provider. If the actual or predicted workload exceeds the capacity of the primary's servers, the excess requests are offloaded to the partner servers. The choice of which clients are offloaded to which partners may be based on provider cost and capacity, expected performance delivered to the client from a given provider, or policies that prefer one provider over another. Figure 1 shows a simple example of how workload might be managed from the primary provider's servers under normal conditions, and partially offloaded to partner servers during peak periods.

Such a peering system allows a service provider to handle surges without over-provisioning, reduce infrastructure costs, and leverage the prices and specialization (e.g., specific geographic regions) of other providers. While peering for content delivery is appealing for all of these reasons, there are a number of challenges in designing a system that virtualizes multiple providers, and directs clients to different providers based on cost, performance, and expected load. Our

contribution lies in designing and implementing a peering system that minimizes cost while also respecting client performance requirements. This is in contrast to peer selection in traditional peer-to-peer networks where the focus is primarily on efficiently finding a nearby peer with the desired content.

We evaluate our strategy against alternative schemes using real traffic data from large Web sites, request-routing data gathered from measurements of operational CDSPs, and network location data collected from a set of globally distributed probe stations. Specifically, we propose solutions for the following key issues:

- *When to offload*? We adopt a proactive approach in which client load is continuously monitored at the primary provider's servers in order to predict future demand and trigger offload when load exceeds available capacity.
- *Who to offload*? We propose and evaluate a novel client clustering scheme that balances network proximity and load prediction accuracy. Small clusters enable more accurate estimation of client location, but larger clusters generate more aggregate traffic, thus improving traffic prediction capabilities.
- *Whom to offload to*? When choosing a partner provider for a group of clients, it is necessary to consider the cost as well as the expected performance. This is complicated by separate administration and limited information sharing between individual providers. We leverage redirection models developed in [3] to predict performance of partners.
- *How to offload*? Our peering system is based on DNS-based request-routing, which is the *de facto* standard used by CDSPs. Domain Name Service (DNS) provides a relatively transparent way to direct clients between service providers, and to recover them when appropriate.

In evaluating our peering strategy we show that our proposed peer selection algorithm is significantly more efficient than greedy alternatives, which average a factor of 1.5 times higher cost. We demonstrate that this performance is consistent over a range of operating environments, including different Web sites, pricing structures, and expected delay thresholds.

The remainder of this paper is organized as follows. In the next section we provide background information. In Section III, we provide a more formal statement of the problem our peering strategies address. We present our peer selection algorithm, experimental methodology, and data collection procedures in Sections IV and V. We detail the results of our analysis in Section VI. We summarize the paper in Section VII.

II. BACKGROUND

Our peering system uses state of the art techniques to estimate client location and predict performance from partner providers. In this section, we give a brief overview of how we leverage existing work in designing our system. In the next section, we provide a more formal statement of the problem addressed in this paper, using terminology and notation introduced in this section.

A. Network Distance Inference

We use the terms network delay and distance interchangeably to refer to the round trip time (RTT) between a pair of IP addresses. A number of techniques have been proposed to estimate delays in the Internet [10][13][18]. A recent empirical evaluation of distance inference techniques [2] shows that the absolute coordinates scheme proposed in [18] performs well, and generally provides low absolute and relative error in estimating delay. We summarize the absolute coordinates technique here, as it is used in later sections to infer distances between Internet hosts.

Geometric heuristics for delay estimation were originally proposed for inter-domain routing by Hotz [10], and recently applied to estimating distances between arbitrary Internet hosts [18]. Hotz defined the relative location of a given host, $\mathcal{H}_i$, as an N-tuple,

$$V'(\mathcal{H}_i) = < d(\mathcal{H}_i, P_1), d(\mathcal{H}_i, P_2), d(\mathcal{H}_i, P_3), \ldots, d(\mathcal{H}_i, P_N), >$$

where each element represents the distance between $\mathcal{H}_i$ and the corresponding probe station (i.e., P_n in position n). This N-tuple, $V'(\mathcal{H}_i)$, is often referred to as the relative coordinates vector of a host. Through the use of global minimization techniques [18], these relative coordinates can be translated into absolute coordinates, $V(\mathcal{H}_i)$. The distance between a host pair, $(\mathcal{H}_1, \mathcal{H}_2)$, is computed as the Euclidean distance between $V(\mathcal{H}_1)$ and $V(\mathcal{H}_2)$.

We use this scheme to assign absolute coordinates to thousands of IP addresses, based on measurements from globally distributed probe stations. We can then infer the distance between clients and servers belonging to partner providers to enable the peering system to direct clients to providers that are expected to meet the performance requirements. We detail the collection of the network probe data, as well as other data sets used in our study, in Section V.

Note that while we limit our attention to distance as the key performance metric (particularly for Web content delivery) other metrics such as optimizing cache hit rate may also be useful [25].

B. Predicting Performance from Partner Server Sets

Large-scale Web sites often serve content from multiple, geographically distributed server sets that may consist of mirrored servers or be part of a content delivery network (CDN). The most common approach for directing clients to a specific server is via DNS request routing. DNS-based techniques may also be used to redirect clients from the primary provider's hosting infrastructure to a partner who performs its own server selection [6].

Predicting the performance a partner server set will provide to a given client requires some knowledge of how the partner selects servers for different clients. The actual server selection criteria used in CDNs are generally considered proprietary information, and may be based on complex and dynamic metrics [5][17], making prediction difficult. However, a recent measurement-based study [3] of

commercial server set deployments showed that, despite the use of a variety of selection criteria, these schemes can be modeled with reasonable accuracy. The study determined that it was possible to predict the distance between clients and the chosen server to within 20 ms in 90% of the cases with a few simple models. The study proposed a distance metric that is the average delay from a client to each server in the set, weighted by the probability that the client is directed to a particular server.

The models developed in the paper included: *i)* selecting the closest server to the client in terms of network delay (referred to as MIN), *ii)* a uniform choice between the first and second closest (MIN2), *iii)* load-balancing with uniform server selection (LBP), and *iv)* a policy in which 80% of the traffic is directed to the closest server while 20% is uniformly distributed among the remaining servers (WGT).

We use information such as the number of servers, locations of servers, and redirection policies discovered in [3] to evaluate our peering strategies. Additionally, when inferring the distance between clients and server sets, we use the same weighted average distance metric. Our peering strategy is described in Section IV and the server set data used for its evaluation is detailed in Section V.

III. Problem Formulation

Our goal is to minimize the cost of hosting a workload on a federated infrastructure comprising sets of servers deployed by a primary service provider and its partners. The primary service provider selects amongst primary and partner server sets; each server set provider performs server selection within its own server set. Server sets may be globally or regionally deployed, and server set coverage may be overlapping since multiple providers may deploy servers in the same network or geographical region. Examples of server sets include surrogate servers deployed by CDSPs, mirrored servers deployed by a content hosting entity, and caching servers deployed in access networks. We focus on workloads (such as Web content delivery) where the primary goal is to service client requests by delivering data to the client.

We perform our analysis using a fee structure representative of the content delivery environment [21]. Specifically, the cost of servicing a workload is the sum of the fees charged by the server sets participating in servicing the workload, plus any penalties due to violations of specified network delay or capacity thresholds. Server set fees are bandwidth-based, that is, the fee is a charge per megabit of content served per second. Fees are assessed according to the 95th percentile of the maximum bandwidth utilized in any measurement period during the service interval. A measurement period, for example, might be 60 seconds whereas a service interval is on the order of 1 month.

Our problem can be formulated as follows: Let [0, T) denote the service interval. Further, let $\boldsymbol{S}$ denote the set of participating server sets where $i \in \boldsymbol{S}$, and $\boldsymbol{K}$ denote the set of clients, $k \in \boldsymbol{K}$. Each server set i has a positive fee denoted by $p_i \geq 0$, which is charged when servicing all client requests within an agreed upon capacity $r_i \geq 0$. The network delay function, $\lambda(k, i)$, specifies whether the client k and the server set i are within the distance threshold, D. That is, $\lambda(k, i) = 1$ if the distance $d(i, k) < D$; $\lambda(k, i) = 0$ otherwise.

Provider i's workload, $X_i(t)$, is the sum of the workloads imposed by each client k, i.e., $X_i(t) = \sum_k X_i^k(t)$. The excess demand of i is $\delta(X_i, r_i)$ [1]:

$$\delta(X_i, r_i) = \left(\sum_k X_i^k(t)(1-\lambda(k,i))\right) \quad (1)$$

$$+\left[\left(\sum_k X_i^k(t)\lambda(k,i)\right) - r_i\right]^+ \quad (2)$$

which is the portion of i's workload that cannot be served by i due to either network delay (1) or server capacity (2) constraints.

If $\delta(X_i, r_i) > 0$, provider i will incur congestion costs. Congestion costs are charges incurred when the offered workload exceeds a provider's capacity. Examples of congestion costs include monetary penalties or loss of goodwill due to not achieving customer service level agreements. If a provider with excess demand chooses to redirect excess demand to partner providers, congestion costs would include peering fees paid to partners. If provider i redirects excess demand to one or more partners, the assignment policy determines the fraction, α_{ij}, directed to a given partner j; this fraction can be expressed relative to individual clients, $\alpha_{ij} = \sum_k \alpha_{ij}^k$.

Client requests that are directed to an overloaded server or to a server for which the network delay exceeds D are said to be *misdirected* by the assignment policy. We use $\xi(X_i^k, \alpha_{ij}^k)$ to denote the portion of the workload that is misdirected. A provider incurs a penalty, γ_i, for workload that is not serviced according to service criteria (either because it was not serviced or because it was misdirected).

Thus, total congestion costs C_i for a single provider i are:

$$C_i(r) = \sum_j \phi\left(\sum_k \delta(X_i^k, r_i)\alpha_{ij}^k\right)p_j \quad (3)$$

$$+\gamma_i\phi(\delta(X_i, r_i)(1-\sum_j \alpha_{ij})) \quad (4)$$

$$+\gamma_i\xi(X_i^k, \alpha_{ij}^k)] \quad (5)$$

where $\phi(\boldsymbol{X})$ is the 95th percentile of a workload $\boldsymbol{X}$. The terms in the definition of C_i represent the cost of offloading traffic to other providers (3), the penalties incurred because the resources required to meet the service level objectives were not available when required (4), and the penalties incurred because client requests were incorrectly directed to a server

[1] In equation (2) we use the notation $x^+ := \max\{x, 0\}$.

that did not meet the service level objectives (5). Note that the cost includes terms related to performance as well as business-related considerations.

Our goal is to develop an algorithm capable of finding the assignment $\alpha_{ij}^{k}(t)$ for all $t \in T$ that minimizes the congestion costs (specified above in terms (3, 4, 5)) incurred by a single provider i. We do not consider optimizations achieved by changes in the capacity (i.e., the values in $\boldsymbol{r}$ are static), since workload assignment algorithms affect only the effective allocation of the capacity of i and its peers. Also, we assume the workload $X_i(t)$ and the price charged by provider i for servicing $X_i(t)$ are not affected by the assignment policy. This allows a simplified cost-based formulation (as opposed to a profit-based formulation).

While the solution is deterministic when the workload, $X(t)$, is known, $X(t)$ for the full interval T is not known at the time the selection is made. More importantly, finding the optimal assignment is not computationally feasible (since there are hundreds of thousands of clients in the environment we wish to address). Our goal is not to solve this problem analytically, but instead, to develop a system that uses heuristics to approximate the optimal solution, given the information available at the time of selection. We use the congestion costs defined in this section to compare our strategy to greedy alternatives. We also describe how our algorithm is incorporated into a more comprehensive peering system, capable of managing workload assignments in a large-scale commercial website.

IV. Peer Selection Algorithm

In this section, we detail our proposed algorithm for cost-optimized peering, which we refer to as COP. We begin by introducing a novel client clustering technique and then describe how COP uses it. We contrast our proposal with two alternative greedy strategies. In later sections, we will compare the performance of these strategies on workloads generated from two world-wide sporting event Web sites, using server sets representative of commercially deployed content delivery infrastructures, and client location data collected from globally distributed network probe stations.

A. IP Address Clustering

COP does not attempt to assign individual client IP addresses to server sets, but instead operates on clusters of network proximal clients. Workload statistics are tracked and server set assignments are performed on a per-cluster basis. While clustering improves the computational efficiency of the assignment algorithm, we also use it to mitigate some drawbacks of DNS-based redirection.

DNS redirection is a highly scalable, but coarse-grained mechanism for directing clients to network service points. Since DNS responses may be cached at clients and local DNS (LDNS) servers, multiple requests from potentially different clients may use the result from a single DNS resolution. Setting the expiration of a DNS response to a small value can limit, but not eliminate, this effect. We propose a traffic clustering technique that enables clients to be grouped for effective network location and traffic estimation, and thus helps to mitigate some of the issues that arise from the coarse-grained client-to-server assignments common to DNS-based redirection.

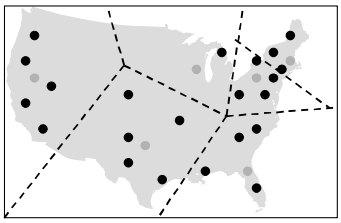


Figure 2: Client IP address clustering via spatial partitioning of the plane in which a number of stations (shown in gray) are designated and probed IP addresses (shown in black) are assigned to the nearest station.

Our clustering approach combines techniques from network-aware clustering, location inference, and spatial analysis. We use network probe data described in Section V to assign absolute coordinates to IP addresses. IP addresses that differ only by the last octet (i.e., /24 address subnets) are grouped together and assigned a single set of coordinates. Our decision to group IP addresses according to the /24 address is based on findings in [2] indicating a small variance in delay measurements to IP addresses in the same /24 subnet. The clustering process begins by selecting a representative set of station coordinates, as depicted in Figure 2, and assigning IP addresses to the nearest station. The station coordinates, which are the centroids of the coordinates of IP addresses in the cluster, serve as the location estimate for all IP addresses within the cluster. In Section VI, we provide a comparative evaluation of station selection alternatives, including human population and Web traffic centers. We also evaluate COP over a range of cluster sizes.

B. Cost-optimized Peering (COP)

The COP algorithm functions by accepting three primary inputs: server set data, workload data, and IP address coordinate data. It produces an assignment of client workload to different server sets, including the primary provider.

The data for each server set i, is the contracted capacity, r_i, the unit price of service, p_i, and the absolute coordinates, $V(s_{ij})$, of the IP addresses of each server s_{ij} in server set i. The server set IP addresses could either be provided by the partner or inferred using techniques similar to those described in [3].

The workload input is a traffic log in which each entry represents a client request and includes a timestamp, client IP address and number of bytes sent in response to the request. The workload, $X^k(t)$, is tracked for each client cluster in terms of number of requests and number of bytes in the response. The absolute coordinates of each client cluster, $V(k)$, are also computed and stored. The output of COP is an assignment of the percentage of requests from each client cluster that should be assigned to each server set.

The COP algorithm casts the server set selection problem as a minimum cost network flow optimization problem. Min-cost flow formulations are a common tool for solving

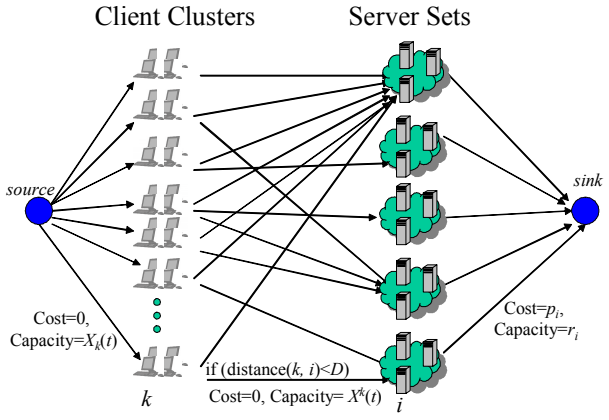


Figure 3: Formulation of server set selection problem as bipartite, minimum cost flow graph. Note that edges link a given client cluster *k* and server set, *i* only if the expected network delay between *k* and *i*, is less than the distance threshold.

distribution, resource management and capacity planning problems [1], and have even been applied to server selection within a single server set [4]. We translate the components of our server set selection problem into a flow graph as follows.

Referring to Figure 3, client clusters are connected to server sets in a bipartite graph. An edge connects cluster *k* to server set *i*, if the distance between *k* and *i* (as defined in Section II.B) is less than the network delay threshold, *D*. The costs of all edges are 0, except those leading from server sets to the sink. These bear the fee associated with the server set. The capacity of the server set to sink edges are set to the contracted capacity of the server set. The capacity of edges from source to client cluster, and client cluster to server sets is set to the workload generated by the corresponding client cluster. COP implements the network simplex algorithm [1] to find the flow values, f_{ki} for each (*k*, *i*)-tuple, that meet the min-cost, max-flow objective.

At administratively defined intervals, COP produces a map, which specifies the fraction of the workload $f_{ki}/X^k(t)$ from *k* that is to be directed to *i* in interval *t*. The map is stored in an expanded trie data structure designed for fast IP routing table lookups. Our peering DNS protocol engine, pDNS, performs a lookup using the client IP address when a client request is received to determine the appropriate server set assignment.

C. Alternative Strategies

Although selection criteria for multi-provider content delivery have been proposed, none address the criteria we wish to evaluate (namely, minimizing service delivery costs while respecting service level objectives). For example, the Content Serving Utility described in [11] allows an administrator to configure the primary provider's authoritative DNS server to redirect clients to a partner CDN based on a few static policies, but does not take into account cost or workload.

A second example, the CDN Brokering system [6], implements a DNS-based request-routing system and an accounting mechanism to bill for traffic when appropriate. The CDN Broker's assignment strategy classifies each client according to the "region" of its IP address, where region is defined as a BGP cluster. The CDN Broker also does not address the optimization criteria we consider, but instead uses a table in which each region has a list of CDNs serving that region. Each CDN is assigned a weight for the regions in which it is listed. However, the details of how these weights are derived, or whether they are ever updated, are not described.

Since none of the published multi-provider request-routing systems address the cost and service level objectives we wish to target, we compare COP with two greedy peering strategies, which we refer to as DualGreedy and MinCost.

The DualGreedy bases its assignment on clients belonging to /24 clusters; it does not use the client clustering described in Section IV.A. When a request from client *k* is received, pDNS searches the list of server sets, and selects the lowest cost server set *i* that meets the delay threshold *D,* and for which the workload assigned so far in the current measurement period *t*, is less than the capacity, r_i. The DualGreedy algorithm is likely to perform well in terms of assigning a server set within the network delay threshold. However, it may not fair as well in minimizing overall costs, since it makes only local (greedy) decisions regarding cost.

The MinCost strategy also does not use client clustering, nor does it use delay threshold. When a request from client *k* is received, pDNS searches the list of server sets, and selects the lowest cost server set *i* for which the workload assigned *i* in the current measurement period *t* is less than r_i. Thus, the MinCost strategy will minimize peering charges (term 3 from Section III) but is more likely to incur penalties (term 5 from Section III) due to missed service level objectives.

In Section VI, we compare the ability of all three strategies to minimize the overall service cost for content delivery in the multi-service provider scenario.

V. Experimental Methodology

A. Evaluation Platform

Our software-based evaluation platform consists of three components: trafficGenerator, pDNS, and statLogger. These components are depicted in Figure 4. The trafficGenerator reads Web server logs, in which each entry contains a timestamp, client IP address, and a count of the bytes served as a result of the client's request. The trafficGenerator generates a DNS request for each entry and sends this request, with the timestamp included in the additional data field, to the pDNS daemon. The pDNS daemon performs server set selection (using the configured peering strategy, COP, DualGreedy, or MinCost), and responds with an answer to direct the client to the assigned server set (i.e., using a CNAME record). The trafficGenerator forwards the request data and response to the statLogger.

The statLogger maintains a serverSet object for each server set. The serverSet object maintains the state of each server set, including the current workload assigned, the server selection policy, and a list of server IP addresses. The serverSet object is capable of responding to statLogger

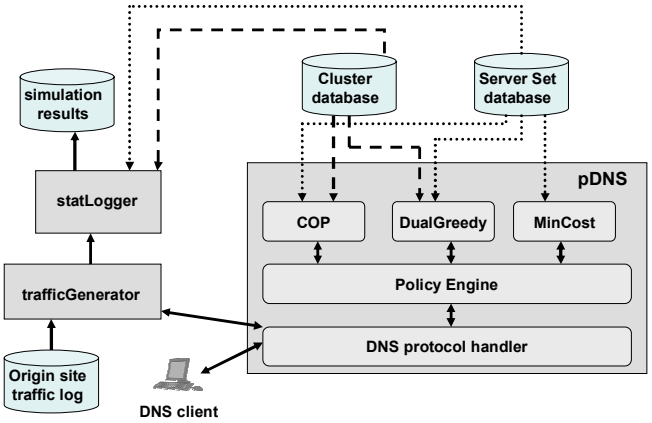


Figure 4: Peering DNS (pDNS) evaluation environment. The pDNS daemon initializes policies according to the configuration. The policies we report on are COP, DualGreedy, and MinCost. The trafficGenerator reads Web server logs to generate DNS transactions and then sends those transactions to the pDNS daemon using the same protocol as a DNS client. The trafficGenerator forwards the results to the statLogger, which tracks and logs performance.

requests for mapping a client IP address, by returning the IP address of the server selected to service the client according to the provider's inferred selection policy. For each transaction received from the trafficGenerator, the statLogger uses the absolute coordinates of the client and assigned server to calculate the expected network delay. If the expected delay exceeds the delay threshold D, a penalty is imposed. At the end of each measurement period, the statLogger requests utilization statistics from each serverSet, and calculates and logs the total usage and penalty fees for the measurement period. Note that penalties due to server set overload are assessed according to the contracted capacity r_i of a server set. That is, since provider i has agreed that r_i capacity is available for use by the primary server set, the load imposed by other customers of provider i need not be considered in the penalty calculations for the primary provider.

B. Data collection

Our evaluation platform employs real-world data collected from three sources: commercially deployed server sets, Web logs from major sporting events sites, and network location data collected from globally distributed network probe stations. In this section, we describe the how these datasets were collected.

1) Web Traffic Logs

We collected Web cache logs from the official Web sites of two major sporting events. The sporting events are hosted on an infrastructure comprising Web servers and caches in multiple, distributed hosting centers. Client requests are directed to one of the hosting locations according to expected network delay and available server resources. The request logs collected from the June/July 2002 tournament, which we refer to as *W1*, represent an aggregate of the requests received by the Web caches located at the four sites indicated in Figure 5. The *W2* data set is also the aggregate of requests received at the same four sites; however the *W2* dataset represents a different event during August/September 2002.

Name	Event	Cache Locations	Captured
W1	Wimbledon [23]	4	6/27/02 - 7/7/02
W2	US Open [22]	4	8/27/02 - 9/5/02

Table 1: Summary of Web cache data sets. Official site information is provided in [22][23].

These data sets are representative of the large-scale distributed network services that motivate our work. For example, the caching infrastructure for the *W2* event serviced over 217 billion requests from clients accessing the servers from over 365 thousand /24 IP address prefixes. Over three terabytes of content were served during this event. Table 1 provides a summary of these two data sets.

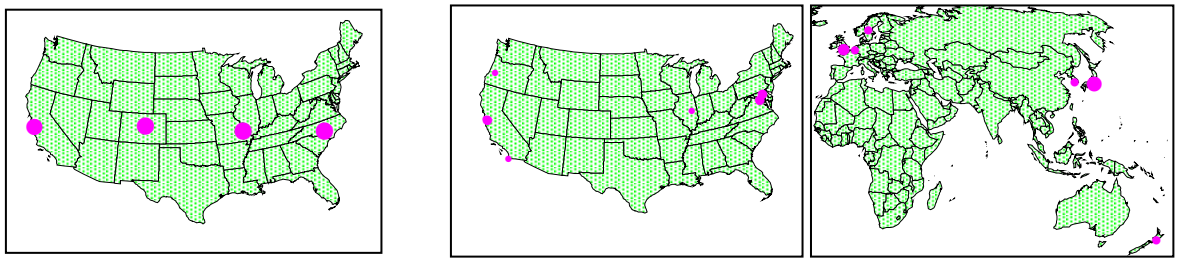
2) Network Probe Data

The *D1* dataset was collected from 21 monitors managed by CAIDA and deployed in geographically and topologically diverse locations [7]. Figure 5 depicts the locations of the 21 monitors. Each monitor continuously probes IP addresses using one of four lists. Three of these lists are designed to cover significant portions of the routable IP Version 4 (IPv4) address space. The fourth comprises hosts taken from DNS root server logs. The number of IP addresses in each list ranged from 125,000 to 825,000.

For each of the 21 probe sites, we collected a list of the IP addresses probed, and the minimum RTT observed for that IP address. We considered only addresses for which at least ten RTT measurement samples had been collected, and grouped addresses according to their 24-bit network address prefix, as described in Section IV.A.

There were about 130,000 /24 clusters for which we had ten RTT samples from all 21 monitors. We noted that some of the probed addresses had very large delays (e.g., greater than 300 ms) from all monitors. Minimum delays of greater than 300 ms across all monitors are likely to be associated with, for example, satellite connections or dial-up connections. Since the distance function evaluation in [2] was unable to verify the accuracy of known inference techniques for such connections, we restricted the *D1* dataset to only those /24 addresses that were within 300 ms of at least one probe station.

In summary, to avoid skewing our analysis, we limited the *D1* dataset to only those /24 addresses for which we had at least ten RTT measurements from each of the monitors, and for which at least one monitor had a minimum RTT of less than 300 ms. Under these restrictions, we selected a set of monitors that would provide the largest pool of /24 addresses for which we had distance measurements. We found that with a subset of 11 monitors, we had a usable pool of 71,000 /24 addresses in *D1*. We assigned absolute coordinates to these /24 addresses according to the scheme described in Section II. A.



a) 4 $\mathcal{W}1$, $\mathcal{W}2$ collection sites *b) Skitter ($\mathcal{D}1$) collection sites*

Figure 5: Collection sites for $\mathcal{W}1$, $\mathcal{W}2$, $\mathcal{D}1$ data. a) The $\mathcal{W}1$ and $\mathcal{W}2$ data sets were collected from the same four sites. b) The $\mathcal{D}1$ data was collected under the Skitter project [7] from 21 sites. In some cases, multiple sites were too close to plot separately. These sites are depicted as a single, larger marker. The size of the marker is indicative of the number of collection sites at the indicated location.

3) Server Set Statistics

Server set statistics were collected as follows. A set of 23 server sets was compiled by visiting the Web sites of five known CDN service providers to retrieve a customer listing, and augmenting this list with popular Web sites using a recognized rating site [12]. Selected server sets were assigned a policy, based on initial measurements from eight geographically distributed clients. The policy assignments were then verified by collecting measurement data from 38 other, non-overlapping systems [20]. The details of this measurement study are provided in [3].

We wished to have our evaluation platform reflect the policies and server locations of commercially deployed server sets. We used the IP address lists and policies collected in [3] to create the serverSet objects implemented in our evaluation platform. The characteristics of the serverSet objects are summarized in Table 2.

Each serverSet was also assigned a peering price, p_i. Intuitively, a server set is likely to be priced according to variety of properties, such as the number of server locations, capacity, and network bandwidth. We did not have access to the actual rates or capacities of these server sets, so we adopted a simple model to enable comparative analysis. For the rate, we assigned $p_i = b + \mathcal{N}_i \times m$, where b is a base price assigned all server sets, $\mathcal{N}_i$ is the number of servers in the set, and m is a per-server multiplier. In addition to testing a range of values for b and m (including $m=0$), we also tested pricing based on total bytes served (as opposed to 95th percentile of maximum bandwidth served).

ID	Policy	#IP Addrs
home	MIN	4
SS0	MIN	4
SS1	MIN2	92
SS2	MIN2	98
SS3	MIN2	91
SS4	MIN	24
SS5	LBP	3
SS6	MIN2	41
SS7	MIN	4
SS8	WGT	4
SS9	LBP	2

Table 2: Characteristics of serverSets used in our system. Each serverSet, except SS0 and home, were modeled after a server set in [3] by adopting the server selection policy and server IP address list of the server set. In some cases, network probe data was not available for a server IP address. For example, only 24 IP addresses were used for SS4, since location data was not available for four of the 28 addresses discovered. The "home" server set uses the IP addresses of the origin servers in $\mathcal{W}1$. SS0 was created to represent a server set that is deployed at the same locations.

We set the capacity of each server set relative to the number of servers in server set. That is, we started by computing a multiplier, ω, which was equal to one half of the average load observed under $\mathcal{W}1$. We set $r_i = \omega$ for server sets with $N_i < 5$ servers, $r_i = 2\omega$ for server sets with $5 \le N_i < 24$ servers, $r_i = 3\omega$ for server sets with $25 \le N_i < 50$ servers, and $r_i = 20\omega$ for server sets with $N_i > 50$ servers. The only exception was the home server set (i.e., the primary provider), which was assigned a capacity equal to *average* load observed under $\mathcal{W}1$.

The serverSet objects instantiated by the statLogger were also separately instantiated by the pDNS daemon. The difference however, was that pDNS had access only to the *expected* delay for a given client, whereas statLogger evaluated results based on the distance between the client and the *actual* server assigned. Additionally, COP (as part of pDNS) bases client assignments on *expected* workload for each client cluster, whereas statLogger evaluated results based on the *actual* workload directed to each serverSet.

VI. Evaluation Results

We compared the ability of our peer selection algorithm to minimize the cost of content delivery services in a multi-provider environment. We also performed an empirical analysis of the sensitivity of these results to a number of factors, including target distance threshold, pricing scheme, and client cluster sizes. We evaluated the peer selection algorithms for both the $\mathcal{W}1$ and $\mathcal{W}2$ events. Finally, we considered operation in an environment requiring predictive peer selection to proactively respond to anticipated surges in demand. In this section we present these results and provide insights into why different results were achieved.

A. Cluster Identification and Evaluation

The COP algorithm makes decisions based on client clusters (as opposed to individual clients), so our first task was to isolate and evaluate IP address clusters from the CAIDA network probe data. Minimizing the maximum intra-cell width for a given number of stations or minimizing the number of

stations for a given maximum cell width are NP-complete problems. Thus, we used heuristics to choose an initial (or seed) set of stations, and apply iterative partition-based clustering to refine this set of stations. Iterative partition-based clustering refers to the process of selecting a set of seed stations, generating a partitioning by assigning each probed /24 address to its closest station, and then generating new station coordinates as the centroids of the current partitions. The centroid is computed based on the network distance from each /24 to its assigned station. This partitioning, assignment, and coordinate update sequence was repeated until there was no change in the coordinates over two cycles, or a maximum number (we used 2000) of iterations were completed.

We chose two independent methods for selecting the initial seed stations and compared these to random seed selection. The first method is based on geographic location, and the second on regions in which we had high observed traffic volumes.

The geographic (geo) heuristic entailed mapping each /24 cluster to a geographic location via IxMapper/GLS [14]. We then created a list of all the locations at which we found at least two /24 clusters that were within 5 ms of each other. We chose 5 ms as the threshold because previous work had shown the typical maximum latency within a single metropolitan area is approximately 5 ms [19]. We calculated the centroid of the /24 clusters for each location that met this criteria. If a cluster was within 5 ms of another cluster, we selected a single representative location by calculating the centroid of addresses in both locations. We found 157 such locations to serve as seeds. We repeated this test by searching for locations with two or more /24 clusters within 10 ms of each other. We found 2615 such locations. We refer to these sets as small#geo and large#geo, respectively.

For the traffic volume heuristic, we rounded the elements of the absolute coordinates vector to the nearest multiple of 5 ms; we refer to these as the quantized coordinates. The traffic generated by clients for each /24 group was assigned to the nearest quantized coordinates so the coordinates could be ranked by traffic volume. The top 157 and 2615 coordinates formed the seed stations for small#vol and large#vol, respectively. Note that these numbers of clusters are chosen to match the numbers found above in the geo heuristics to allow comparison. We also randomly selected a set of 157 and 2615 coordinates to form the small#ran and large#ran seed sets. We refined the coordinates of these six seed sets as described above using iterative partition-based clustering to produce the final set of station addresses. The number of final stations, and thus clusters, resulting from each seed set is less than the number of seeds, since clusters are merged if their station coordinates are within 5 ms. For example, the set of 2615 seeds initially in large#vol resulted in a partitioning with 880 clusters.

In Figure 6, we plotted the distribution of traffic generated by /24 clusters, with respect to the distance from their assigned station. For example, using clusters formed from the large#ran seed set, over 70% of the client requests were generated from

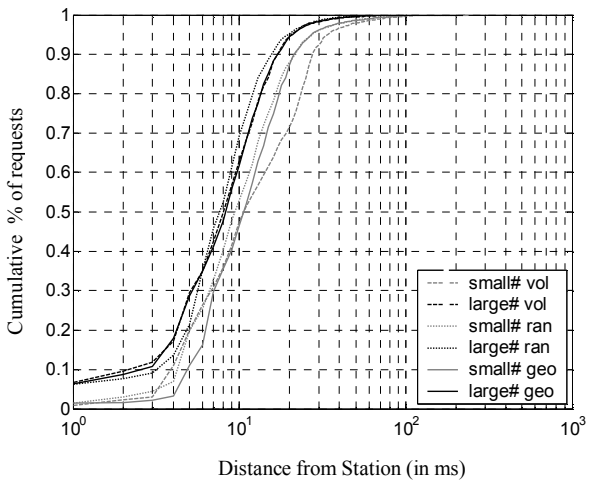


Figure 6: Comparison of cumulative distribution of client requests (traffic volume) to network distance. The x-axis is in log scale for improved readability. The plots are labeled according to the number of initial stations (small=157, large=2615) and the method used to select the initial stations (geo=geographic, vol=traffic volume, ran= random). The plots for large#vol and large#geo are overlapping in many areas.

locations within 10 ms of their assigned station, and over 96% from within 20 ms. Clusters resulting from the large#vol and large#geo seed sets had similar proximity results. Expectedly, fewer clusters (as formed from small#geo, small#ran, small#vol), resulted in wider clusters. For example, approximately 87% of the client requests under small#geo and small#ran were generated from locations within 20 ms of their assigned station.

We will begin our peering strategy comparison using the coordinates identified from the large#vol seeds. In later sections, we will consider the sensitivity of these results by comparing other clusterings.

B. Peering Strategy Comparison

Using the evaluation platform described in Section V.A, we compared the performance of COP and two greedy alternatives on the workload generated in *W1*. We measured the cost achieved by setting the distance threshold to 50 ms, 100 ms, 150 ms and 200 ms. This comparison, in relative units, is illustrated in Figure 7.

The decrease in cost as the distance threshold increases is as expected. COP is significantly more efficient than the greedy alternatives, with alternatives averaging a factor of 1.5 times more costly than COP. We also computed the relative costs when using pricing based on total bytes served. The performance under the bytes-served schemes was consistent, with greedy schemes costing 1.3 to 2.1 times more than COP.

We also evaluated the cost achieved by these three schemes on the *W2* event. Again, the cost achieved under the greedy algorithms averaged 1.5 times the cost achieved when using COP.

Increased costs under the greedy schemes could be due to a number of reasons. Penalties incurred by exceeding the capacity or network delay thresholds are the most likely reasons. Both COP and DualGreedy use absolute coordinates

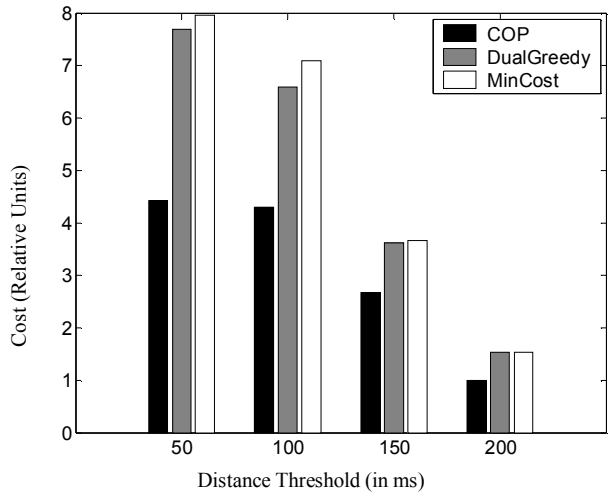


Figure 7: Comparison of relative costs achieved by COP, DualGreedy, and MinCost algorithms over a range of values used as the targeted distance threshold (*D*).

data as a basis for expected network delay. However, DualGreedy assigns low cost server sets on a first-come-first-serve basis. This strategy could result in "filling" the capacity of certain server sets with clients that could be served by many server sets, at the expense of clients that are within the delay threshold of only a small number of server sets. This does not explain why, for example, MinCost achieved similar results to DualGreedy, even though it does not consider network delay thresholds and therefore is more likely to incur high penalties. To gain insights into these and other issues, we plotted the distribution of traffic over the course of the *W1* event.

In Figure 8, we show the breakdown of requests serviced on the primary provider's infrastructure, versus those serviced on peer infrastructures, as well as on peer infrastructures in excess of the contracted capacity. We also plotted a comparison of requests served within the targeted delay threshold, versus those exceeding the network delay threshold or overloading peers' servers.

A prominent feature of this set of plots is that COP, in contrast to DualGreedy and MinCost, did not assign load to peers in excess of their contracted capacity. COP does assign load in excess of the primary, or home, infrastructure capacity during all four of the major surges. This is an indication that the peers with available capacity were outside the network delay threshold. These requests would have incurred penalties regardless of whether they were serviced from the primary, or a peer's infrastructure, so they were serviced on the home infrastructure to avoid peer charges.

The diagrams in Figure 8 also illustrate other subtleties. For example, consider that MinCost is competitive with DualGreedy for most distance thresholds (Figure 7). By comparing Figure 8c to 8e, we see MinCost had higher proportions of the load serviced under high delay conditions (resulting in higher penalties). This is not surprising since MinCost does not consider expected network delay when making assignments. However, because MinCost directed more load to the primary server set than to peers (compare 8b to 8d), these penalties were partially offset by lower costs due to peers. The captions in Figure 8 discuss additional comparisons.

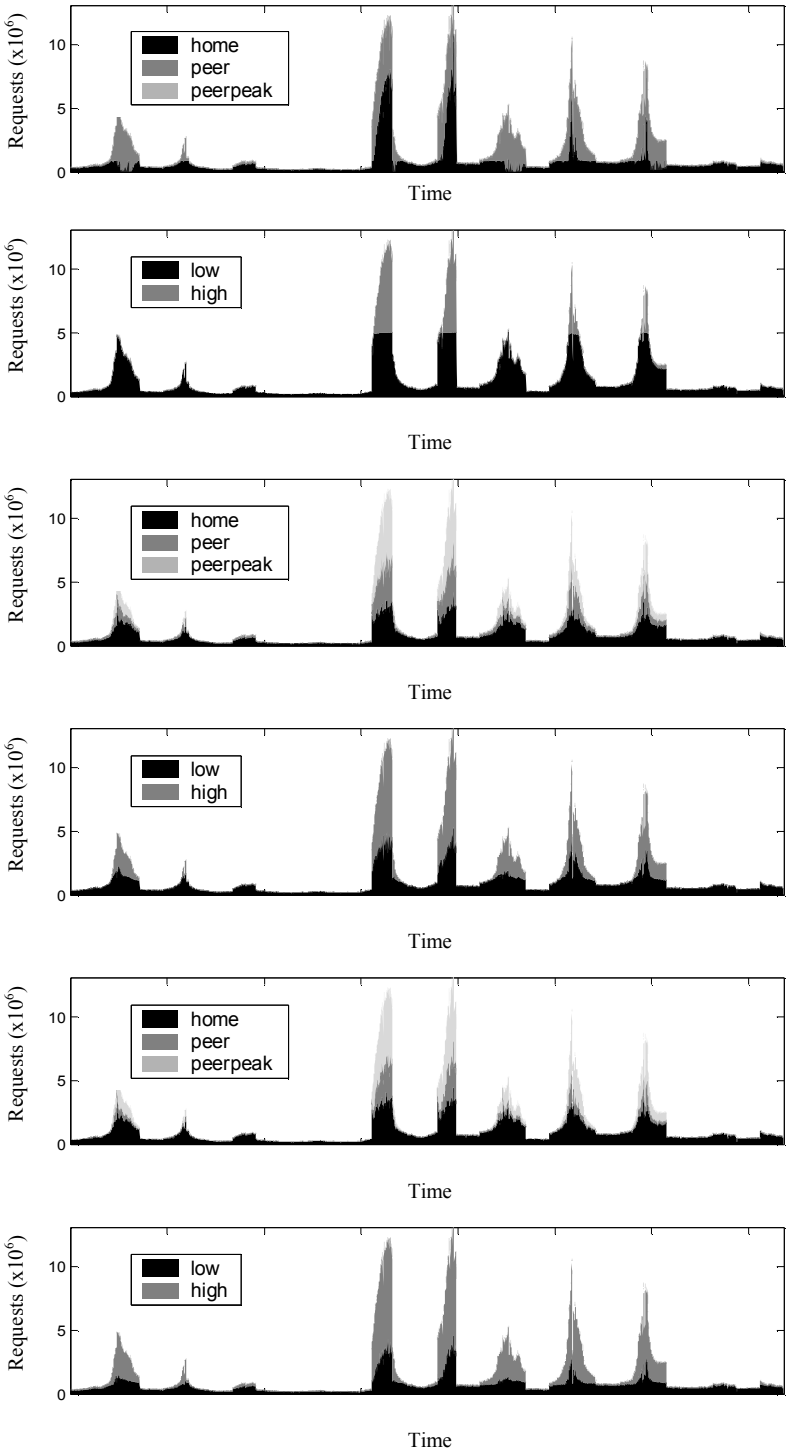
C. *Effects of Client Clustering on Traffic Prediction*

In the preceding evaluations, we used a large number of client clusters. Specifically, using traffic volume heuristics to select 2615 seed stations, we used iterative partitioning to isolate 880 clusters. While using a large number of clusters can improve the estimation accuracy of client location, it also reduces the cluster client population, and may make traffic prediction more difficult.

Table 3 helps to illustrate this trade-off. We created a number of partitions, using different seed heuristics and a range of initial stations. We then used standard linear prediction on the numbers of requests generated by each of the clusters and calculated the relative deviation between the actual and predicted number of requests for each interval in *W1*. The linear prediction used a sliding window of ten 60-second intervals to predict the expected number of requests for the next 60-second interval. We have tested other prediction schemes, such as exponentially weighted moving averages and best linear fit, but found that simple linear prediction, with three coefficients performs well in this environment.

Table 3 shows that as the number of clusters (column 3) increases, the percentage of requests generated from locations within 10 ms of the assigned station increases, but the relative deviation for traffic predictions also increases. We were interested in the effect this may have on our peering algorithm. We selected three sets of clusters to test the sensitivity to cluster size: *i)* the set of 46 clusters (row 1 of Table 3), *ii)* the set of 310 clusters (row 5), and *iii)* the set of 1756 clusters (row 10). We chose these sets to cover a range of cluster sizes.

We compared the costs achieved using these cluster sets and linear prediction to those achieved earlier for 880 clusters, which had simply used the number of requests generated by a client cluster in the previous interval to estimate the number of requests that would be received in the next interval. Both the set with 310, and the set with 1756 provided an improvement over the baseline set with 880 clusters. The set with 310 clusters achieved a cost of 95% of that achieved by the set with 880 clusters. Additionally, the set with 1756 clusters achieved a cost of 79% of the cost under 880 clusters. The cost achieved by the set of 47 clusters was within 0.1% of the cost achieved using 880 clusters. These results indicate the cost is affected by the selected clustering, but that a range of cluster sizes work reasonably well in practice. In future work, we intend to further explore the process for choosing the most appropriate number of clusters.



←(a) COP algorithm; comparison of requests serviced on home infrastructure, versus peers and peers in excess of contracted capacity. Unlike DualGreedy (c) and MinCost (e), COP did not assign load to peers in excess of the contracted capacity.

←(b) COP algorithm; comparison of requests serviced within the delay threshold versus those exceeding delay threshold or contracted capacity. Plot shows number of requests serviced with low delay is more than for DualGreedy (d) or MinCost (f) during same traffic surge periods.

←(c) DualGreedy algorithm; comparison of requests serviced on home infrastructure, versus peers and peers in excess of contracted capacity. Load served on home infrastructure was slightly less than under MinCost (e), resulting in higher peer costs.

←(d) DualGreedy algorithm; comparison of requests serviced within the delay threshold versus those exceeding delay threshold or contracted capacity. Load serviced under low delay conditions was slightly higher than MinCost (f), resulting in higher penalties due to exceeded network delay threshold.

← (e) MinCost algorithm; comparison of requests serviced on home infrastructure, versus peers, and peers in excess of capacity. MinCost directed more of the load to the primary server set than either COP (a) or DualGreedy (c). The result was to lower fees paid to peers, but increased penalties due to exceeding contracted capacity and network delay thresholds.

←(f) MinCost algorithm; comparison of requests serviced within the delay threshold versus those exceeding delay threshold or contracted capacity.

Figure 8: Comparison of offload and delay characteristics for COP, DualGreedy and MinCost algorithms over the course of the *W1* event. The diagrams in (a), (c), (e) compare the requests serviced on the home infrastructure, versus peers, and peers in excess of contracted peer capacity. The diagrams in (b), (d), (f) compare the requests serviced within low delay, versus requests that were serviced under high server or network delay conditions.

#Seeds	Seed Policy	#Clusters Isolated	%Reqs <10ms	%Reqs <20ms	Avg Rel Dev
157	vol	46	47.5	71.5	0.019
157	geo	131	46.5	87.2	0.235
157	ran	153	52.9	87.9	0.220
500	vol	145	54.1	90.0	0.257
1000	vol	310	58.8	91.8	0.268
500	ran	412	58.1	91.3	0.261
1000	ran	721	58.9	93.3	0.265
2615	vol	880	62.8	94.4	0.259
2615	geo	1687	61.8	94.4	0.250
2615	ran	1756	69.2	95.1	0.266

Table 3: Effects of seed policy and number of clusters isolated on cluster properties. Smaller numbers of clusters tend to be wider (i.e., have smaller percentages of requests within 10 or 20 ms of assigned station), but more accurately modeled by simple linear prediction techniques (i.e., the predicted request rates have lower average relative deviations).

Our peering system is designed so that this trade-off can be dynamically managed, according to the needs of a particular application. For example, during non-peak periods (or in environments where server overload is unlikely), a clustering with very large numbers of small, more fine-grained clusters (e.g., /24 clusters) could be used by the DualGreedy algorithm to achieve higher client location accuracy. Under surge conditions (or whenever server overload is more likely), a clustering (such as large#vol) with fewer numbers of clusters, and thus better traffic prediction, can be used by the COP algorithm to protect against both server overload and assignments exceeding the delay threshold. The hybrid approach is used only if the system is configured to do so, and the switch between the COP and DualGreedy modes can be triggered according to traffic volume.

D. Discussion

Our analysis has demonstrated the value of intelligent peer selection over a broad range of operating scenarios. We have focused on relative performance, using a cost-based formulation that takes into account expected network delay and server capacity thresholds. Our COP approach does have some drawbacks, however, which we discuss below.

COP relies on extensive network probing to build a database of absolute coordinates in order to infer network distances. (The DualGreedy algorithm also uses this network distance database.) We use the minimum measured delay to estimate the raw propagation delay, which is not expected to change frequently. Hence, the probing can be done infrequently to refresh the delay database. Alternatively, we could use a "lazy" approach in which we probe only active clients (i.e., those that initiate requests) to avoid probing portions of the IP address space that do not contain clients of interest. This has the potential drawback of initially misdirecting clients for which no location information is yet available.

COP also uses more complex computations in arriving at its solution than the other techniques we evaluated. The network simplex flow algorithm that COP uses, however, was able to compute flow graphs for hundreds of client clusters in less than a second on a 900 MHz Intel processor. Moreover, these computations are done independently of the pDNS server "fast path" which must respond quickly to client DNS queries. COP monitors the workload and computes client cluster-to-server set mappings asynchronously, and periodically generates updated maps, which the pDNS server consults using an efficient IP routing table lookup scheme.

Some of the component techniques used in COP can also be further improved. For example, the redirection models do not perform as well for some server sets that use dynamic selection policies that react frequently to changing network conditions or server load. Also, though the client clustering technique used in COP seems to work reasonably well in practice, the process of choosing the correct number of stations is still not well-explored.

Finally, scenarios in which multiple providers contend for the same cooperative resources may occur in practice. In such scenarios, the resources available from partners might vary over time. Although our evaluation used static capacities, COP minimizes the service delivery cost according to the available capacities, even if the available capacities vary over time. However, it is possible that also employing a model to predict how capacities could be expected to vary over time (due to the interplay of multiple providers vying for resources) would enable further optimizations.

There are also a number of advantages to the COP approach that go beyond the performance benefits described in the previous sections. In particular, our peering solution is highly flexible. For example, a primary provider typically hosts several applications, or Web sites, using the same shared, multi-provider infrastructure. This scenario can be accommodated in COP by creating a source node in the flow graph for each application, and connecting the source to each of the cluster nodes. COP also provides flexibility in partitioning server sets. If a partner provider implements a tiered pricing structure, multiple server set nodes can be created in the flow graph, each with a cost and capacity to reflect the pricing tiers.

Additionally, client affinity to server sets is useful to direct requests for certain types of objects to particular sever sets, especially in scenarios where requested objects (e.g., streaming media files) are large. This avoids the situation in which many of the partners must deliver bandwidth-intensive objects or services (possibly increasing the cost to the primary provider). Also, client affinity can provide a more consistent response time by avoiding directing clients to multiple server sets that have varied performance characteristics. Client affinity can be accomplished by using the same flow values calculated for a previous flow graph in the current interval.

VII. SUMMARY

In this paper, we propose an effective peering system for content delivery workloads in a federated, multiple service provider infrastructure. The core component in the system is a novel peering algorithm, COP, which directs client requests to partner providers such that cost is minimized and performance targets are respected. We evaluate the algorithm against greedy strategies using real traffic data from large-scale Web sites and request-routing data gathered from server sets deployed by commercial CDSPs. We also use network location data collected from globally distributed probe stations. Our evaluation shows that the proposed cost-optimized peering algorithm is significantly more effective than greedy alternatives, which incurred costs of 1.3 to 2.1 times more than COP over a variety of network delay thresholds, pricing schemes, and client cluster sizes.

In addition to refining the server set models and client clustering techniques used in this study, we are also investigating interesting related questions [2]. For example, the question of how to provision primary resources relative to the capacity contracted from partner providers or the peering strategy of partners is virtually unexplored. We are also exploring strategies for how to price a virtual network service, comprised of multiple providers, to potential customers. Finally, we are considering the application of our peering strategy and evaluation methodology in other application contexts, such as Grid and 3G wireless deployments.

ACKNOWLEDGEMENTS

Daniel Bienstock and Olivier Verscheure provided many valuable comments. kc claffy, Daniel Anderson, Andre Broido, and Young Hyun provided access and explanations regarding the Skitter (*D1*) dataset. Paul Dantzig, Brian Snitzer, Herbie Pearthree, Milo Choong, Robert Danford, and Todd Allensworth provided access and key insights into the *W1* and *W2* datasets. We are thankful for all of their help.

REFERENCES

[1] R. Ahuja, T. Magnanti, J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.

[2] L. Amini, "Models and Algorithms for Resource Management in Distributed Computing Cooperatives," PhD Thesis, Columbia University, 2003.

[3] L. Amini, A. Shaikh, H. Schulzrinne, "Modeling Redirection in Geographically Diverse Server Sets," Proceedings of World Wide Web Conference (WWW2003), May 2003.

[4] M. Andrews, B. Shepherd, A. Srinivasan, P. Winkler, F. Zane, "Clustering and Server Selection using Passive Monitoring," *Proceedings Infocom 2002*.

[5] S. Bhattacharjee, Z. Fei, "A Novel Server Selection Technique for Improving the Response Time of a Replicated Service," *Proceedings of IEEE Infocom*, March/April 1998.

[6] A. Biliris, C. Cranor, F. Douglis, M. Rabinovich, S. Sibal and O. Spatscheck, "CDN Brokering," *Web Caching and Content Distribution Workshop*, June 2001.

[7] CAIDA's Internet Topology Data Kit #0204. http://www.caida.org. Funding for this work was provided by DARPA's Network Modeling and Simulation Program N66001-01-1-8909.M. Crovella and R. Carter, "Dynamic Server Selection in the Internet," Proceedings of the Third IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems (HPCS'95), August 1995.

[8] M. Day, B. Cain, G. Tomlinson, P. Rzewski, "A Model for Content Internetworking," Work in Progress, May 2002, http://www.ietf.org/internet-drafts/draft-ietf-cdi-model-02.txt.

[9] I. Foster, C. Kesselman, S. Teucke, "The Anatomy of the Grid," *International Journal of Supercomputer Applications*, November 2001.

[10] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, L. Zhang, "IDMaps: A Global Internet Host Distance Estimation Service," *IEEE/ACM Transactions on Networking*, October 2001.

[11] P. Gayek, R. Nesbitt, H. Pearthree, A.Shaikh, B. Snitzer. "A Web Content Serving Utility," *IBM Systems Journal*, 43(1), 2004.

[12] Hot100. http://www.100hot.com

[13] S. Hotz, "Routing Information organization to support scalable interdomain routing with heterogeneous path requirements," PhD Thesis, University of Southern California, 1994.

[14] IxMapper. http://www.ixiacom.com/products.

[15] R. Katz, A. Joseph, "A Revolutionary Confederated Service Architecture for Future Telecommunications Systems," University of California MICRO Research Proposal, March 2001, http://www.cs.berkeley.edu/~randy/proposals/micro01.pdf.

[16] B. Krishnamurthy, J. Wang, "On network-aware clustering of web clients," *Proceedings of SIGCOMM*, August 2000.

[17] B. Krishnamurthy, C. Wills, Y. Zhang, "On the Use and Performance of Content Distribution Networks," *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW'2001)*, November 2001.

[18] T. Ng, H. Zhang, "Predicting Network Distance with Coordinates-Based Approaches," *Proceedings of Infocom*, June 2002.

[19] V. Padmanabhan, L. Subramanian. "An Investigation of Geographic Mapping Techniques for Internet Hosts," *Proceedings of the ACM SIGCOMM*, August 2001.

[20] Planetlab, http://www.planet-lab.org.

[21] S. Poulouchkine, J. Ravier, "State of the Art on Content Delivery Networks (CDNS)," *survey published by France Telecom Research and Development Division*, July, 2000.

[22] 2002 US Open Tournament Web Site, August 28 – September 5, 2002, http://www.usopen.org, site is no longer available.

[23] 2002 Wimbledon Tournament Web Site, June 27 – July 7, 2002, http://www.wimbledon.org, site is no longer available.

[24] D. Villela, and D. Rubenstein, "A Queueing Analysis of Server Sharing Collectives for Content Distribution," *International Workshop on Quality of Service (IWQoS 2003),* June 2003.

[25] L. Wang, V. Pai, and L. Peterson, "The Effectiveness of Request Redirection on CDN Robustness," *Proceedings of the Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.