

# CDN Brokering\*

Alexandros Biliris, Chuck Cranor, Fred Douglis, Michael Rabinovich,  
Sandeep Sibal, Oliver Spatscheck, and Walter Sturm

AT&T  
Florham Park, NJ

In *Proceedings of the 6th International Workshop on Web Caching and Content Distribution*, Boston, MA, June, 2001

## Abstract

*Content distribution networks* (CDNs) increase the capacity of individual Web sites and attempt to deliver content from caches that are located “closer” to end-users than the origin servers that provide the content. *CDN brokering* provides CDNs a way to interoperate by allowing one CDN to intelligently redirect clients dynamically to other CDNs. This paper describes the goals, architecture, and performance of a CDN brokerage system. Our system has been deployed on the Internet on a provisional basis, and our architectural ideas have helped advance the evolution of Internet standards for interoperating CDNs.

## 1 Introduction

As the scale and use of the Internet increase, Web content providers find it increasingly difficult to serve all users wishing to access their content with an appropriately low response time, especially in the face of unexpectedly high loads (often called “flash crowds”). Content distribution networks (CDNs) have become a popular approach to address this problem; they typically deploy multiple caches throughout the Internet, allowing them to offload the management of these caches from individual content providers. This offers economies of scale and greater abilities to respond to high loads. CDNs also try to decrease latency for individual clients by satisfying client requests from nearby caches.

Despite these advantages, CDNs have their own limitations. It is extremely capital-intensive and operationally complex for CDNs to achieve the scale necessary to be present at the edge of every network or region of the world. Even where they have cache space close to a particular network, demand may outstrip capacity from time to time or failures may occur. Clearly the ability for multiple CDNs to interoperate can offer additional

flexibility by letting one CDN direct clients to a different CDN in order to improve scale, fault tolerance and performance. This is sometimes referred to as “Content Distribution Internetworking” (CDI), the subject of efforts within the IETF.

We define **CDN Brokering** as the ability of one CDN to redirect clients dynamically among two or more CDNs. While over the past few months the Content Alliance has put forth several documents [2, 7, 8, 9, 13] describing in general terms how one might let CDNs interoperate, this paper describes an architecture and implementation of a specific realization of this idea, which has been operational since July, 2000. We call our DNS-based system the *Intelligent Domain Name Server* (IDNS). IDNS responds to DNS requests intelligently based on a dynamic, load-sensitive configuration rather than using static information.

The rest of this paper is organized as follows. Section 2 discusses background and motivation. Section 3 describes CDN brokering in general terms and Section 4 addresses our specific software architecture. Section 5 discusses the performance of our system. Section 6 covers related work, and Section 7 concludes.

### 1.1 Terminology

Content provider companies that sign up for a CDN service are known as *customers* and Web clients (users or client proxies) that access customers’ content are called *clients*.

Clients typically look up DNS information via name servers in their own network that act on behalf of a collection of computers; these client-side name servers are called *Client DNS servers*.

## 2 Background and Motivation

Not all CDNs are created equal or have the same goals. Some are small, regional services that target a particular segment of the Internet, for instance a country of moderate size. Others have a large presence within a particular network, but are limited to the

---

\* Contact address: [icds-broker@research.att.com](mailto:icds-broker@research.att.com)

reach of that network unless they have extensive BGP peering relationships [14] or other connectivity that provide high bandwidth to many other networks. Still other CDNs are worldwide and attempt to encompass a large fraction of the Internet. However, deploying a CDN in many edge locations and managing those servers is an enormous commitment in capital and labor.

In fact, a CDN of any scale might find it financially advantageous to have an arrangement that allows it to wholesale capacity to other CDNs. Similarly, the ability to *buy* capacity at wholesale from another CDN in those rare cases when one's own CDN is overloaded or (partly) unavailable would provide extra insurance. Finally, a CDN that wishes to expand its presence might resell capacity from other CDNs on an ongoing basis, rather than under exceptional circumstances.

CDN brokering involves a mechanism that allows the broker to dynamically decide on the fraction of requests to be served from its own CDN, and to split the remaining requests among other partner CDNs, again according to some dynamically decided percentage.

### 3 Brokering Components

In this section we describe in general terms how CDN brokering is accomplished. A brokering system needs a mechanism to select which CDN is the best to serve a particular request (Section 3.1), a way to direct clients to that CDN (Section 3.2), a way for a partner CDN to map requests back to an origin server to obtain content (Section 3.3), and an accounting mechanism to bill for traffic when appropriate (Section 3.4).

#### 3.1 Selection

Selecting to which CDN a client should be directed requires two things. First, it requires a mechanism to identify where a request came from. Second, it requires information about suitable locations to which the client can be directed at any given instant. Generally speaking, the biggest decision in this regard is whether to serve the request within the brokering CDN or to redirect to another CDN. We discuss CDN selection in detail in Section 4 in the context of our architecture.

#### 3.2 Redirection

Figure 1 depicts a redirection mechanism for a brokering CDN, which we refer to as *B*. The diagram also shows two other "partnering" CDNs, *G* and *H*. The edge servers are shown as rectangles and the DNS servers are shown as small ovals. Each of the CDNs has its own DNS server(s) that it uses to direct clients to its own servers. Of particular importance are the brokering DNS servers that allow *B* to redirect clients not just to one of its

own servers, but to any other partnering CDN as well. This is accomplished through another layer of indirection as explained below.

An underlying assumption is that the redirection mechanism is using DNS and that *B* is authoritative for the hostnames its customers are having it serve, either directly or via a CNAME redirection. (There are other possible methods for what the Content Alliance refers to as "request-routing" [2]; we compare our approach to some of them in Section 6 below.) DNS systems use time-to-live (TTL) values to control the granularity of host assignments, and the selection of an appropriate TTL is important to balance the overhead of brokering against the impact of sending the clients of a particular client-side DNS server to one cache or CDN.

Assume that a customer has signed up for service with *B*, and a client requests content provided by this customer. The sequence of steps for a client to retrieve that content is as follows:

1. The client attempts to resolve the domain name in the URL of the content piece hosted by *B* by sending a lookup request to the local Client DNS, unless the client has an unexpired response already cached.
2. The Client DNS eventually contacts one of the brokering DNS servers (BDS) that is authoritative for that domain.
3. The BDS then selects one of the following possibilities depending on the control policy that it supports:

**Serve externally:** A DNS-based brokering mechanism forwards the request to another CDN via DNS CNAME or NS response. The client DNS server resolves this response further to retrieve an A record. This approach is described further in Section 3.3.1.

**Serve internally:** If the BDS selects *B*'s own CDN, it has several options depending on whether it can assign a specific (virtual) cache directly:

- Return an A record that contains the IP address of a server within its own CDN. This is the most efficient option, but it requires the closest cooperation between the BDS and the rest of the brokering CDN, *B*.
- Perform a triangular resolution; i.e., forward the DNS query to a DNS resolver for *B*'s CDN, and let that DNS server respond directly to the client DNS server.
- Redirect the client DNS server explicitly to *B*'s internal DNS server, using the same techniques as for external CDNs.

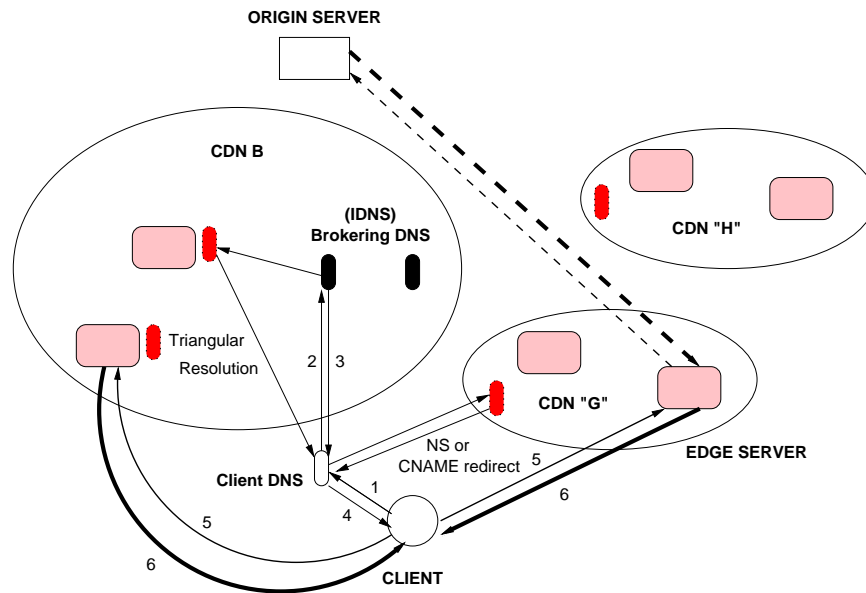


Figure 1: High level schematic of the IDNS redirection mechanism

4. The Client DNS forwards the IP address (DNS A record) of the selected edge server to the client.
5. The client sends the request for the content to the selected edge server.
6. The edge server satisfies the client request either by serving it from its cache or from the origin server.

### 3.3 Naming

The previous subsection described how a client is directed to a server that will provide the requested content. Here we discuss how that server is selected within the partner network and how it knows what content to provide. These two issues are related, as they determine the naming conventions that different CDNs can use to interconnect.

#### 3.3.1 DNS mapping

In Section 3.2, we described a choice in which *B* redirects a client DNS server to *G* using CNAME or NS, but did not describe what hostnames would be used or what *G* would do once it receives a request. For NS redirection, *G* is made authoritative for the lowest level of the domain name; since it sees the original

hostname for the customer, it knows exactly which host's content is being requested. For CNAME redirection, however, the client DNS server is given a new DNS name in *G*'s domain. If *G* treats all servers equivalently, it needs no special information at the time of the DNS query: it will resolve to a cache, and use the HOST header of the HTTP request to identify the content. But if *G* serves content for this customer only from some of its caches, it must identify the customer at the time of the DNS lookup, and *B* and *G* must agree on the format of that name.

We use *semantic mapping* to exchange that information. *B* transforms hostnames automatically when redirecting, for instance:

```
a.example.com -> a.example.com.B.G.com
```

In this case, the presence of CDN *B* in the domain name in *G*'s DNS namespace implies that *B* has redirected the request to *G*. Although this semantic mapping does not prevent the encoding of individual object IDs in the DNS name we do not encourage such an encoding. Such an encoding would increase the number of DNS names used and consequently the load on the DNS system.

#### 3.3.2 Obtaining Content

A second problem is allowing *G* to serve the requested content.

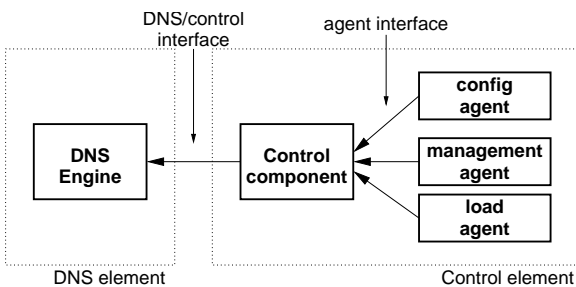


Figure 2: Architectural overview of IDNS

To go to the origin server,  $G$ 's cache must be able to find the IP address for a host serving the content of `a.example.com`. Typically, CDNs statically configure caches to have a “back door” from which to obtain content on a miss in order to avoid loops that would redirect queries for the original hostname back into the CDN itself. They might identify hosts by a specific IP address (bypassing DNS) or a hostname that does not resolve via the CDN. This static configuration must be done for each brokered site.

### 3.4 Accounting

Currently, CDNs charge their customers based on usage. They are responsible for collecting usage information and for making that information available to the customers when they issue bills. CDNs may bill each other explicitly for usage, or perform other forms of settlement, but regardless there must be a mechanism to exchange that information. Both the Content Alliance and the Content Bridge organizations have efforts underway to standardize this exchange [3, 13], and we expect to adopt those methods once they are standardized.

## 4 IDNS Architecture

Figure 2 shows the high-level IDNS architecture. IDNS consists of two main elements: a DNS element and a control element. The first contains the DNS engine process which receives and answers DNS queries from the network. The control element maintains configuration information, receives live feedback and updates the tables used by the DNS engine to resolve requests. Communication between the two elements is done using a TCP socket based DNS/control interface.

IDNS's control element is the heart of the IDNS system. It consists of a control component and a set of agents. The agents are used to configure, manage, and obtain load feedback information from remote systems. The control component contains

a load balancing algorithm that takes input from the agents and produces a set of tables (or a set of changes to the current tables) that are suitable for download into the DNS engine. These tables should contain the best distribution of CDN resolutions for a set of client regions, based on a given policy, that can be computed efficiently.

IDNS' configuration tables map client DNS server IP addresses to *regions* [11] and are used to select between CDNs serving a client's region by assigning a set of probability distributions. A region consists of a name and a set of IP prefixes associated with that region. Regions can be defined in many ways. For example, IP addresses can be grouped by IP-backbone connectivity or by country. Currently we use Krishnamurthy and Wang's [11] approach to clustering clients into regions. The control component passes the region definitions to the DNS element using a control update message. The DNS engine then uses an IP address longest prefix match subsystem to map a DNS server address to a region.

In the remainder of this section we examine the IDNS architecture in more detail.

### 4.1 DNS Element

The DNS engine is a special-purpose DNS server, and was designed to be small and robust. Since it is a stand-alone process, other components of the IDNS system can fail and/or be restarted while the DNS engine will continue to resolve client DNS requests without interruption using its current configuration. The DNS element supports atomic updates to its database, and it continues to respond to DNS requests even when a database update is in progress. These features are achieved by taking advantage of the copy-on-write memory semantics provided by the UNIX kernel's virtual memory system.

#### 4.1.1 DNS Engine Operations

At startup, the DNS engine initializes its tables, creates two sockets, and then enters its main loop. One socket is used to receive UDP DNS requests. The other is used for TCP DNS/control interface requests. These requests are processed one-at-a-time in the order they are received (requests received while an update is in progress are queued in the socket listen queue until they can be serviced). The TCP control connection has an inactivity timeout to prevent hung control processes from blocking access to the server.

The DNS element provides transactional update semantics to the control element. We use two processes, communicating via a pipe and sharing physical memory in a copy-on-write fashion, to ensure that exactly one process (with either the original or modified control tables) serves requests.

### 4.1.2 Table Management

Internally, the DNS element stores its configuration in three types of data structures.

**Region table:** The region table, defined above, clusters all IP addresses into a set of regions. The DNS engine allows the fast matching of a given IP address into the region with the longest prefix match. A default region is used to handle IP addresses that cannot be mapped to any other region.

**Distribution tables:** There are multiple distribution tables, each being representative of a particular set of CDNs used to offer a particular service. Each of those distribution tables contains one entry per region, specifying what type of resolution should be used and to which CDN a DNS request should be resolved with which probability.

**Customer table:** The customer table associates customers defined by their DNS name with a particular distribution table. The reason for this level of indirection is that a distribution table is determined by the CDNs involved and that the same set of CDNs is likely to be used by multiple customers, though not every customer will use every CDN.

### 4.1.3 DNS Engine Lookup

To resolve a DNS request, the DNS engine first finds the client DNS server's region by searching the region table using longest prefix matching. It then maps the request to a distribution table using the requested DNS name and the customer table. Next, it determines a particular set of probabilities by using the region as an index in the distribution table. Using those probabilities, one of the CDNs in the selected entry of the distribution table is used to resolve the request. Therefore, the overall lookup overhead consists only of one longest prefix match, one hash table lookup, one array element lookup, and one simplified random number generation.

## 4.2 Control Element

In this section we describe the control element data structures, load balancing, and control agents.

### 4.2.1 Control Data Structures

The control element's three main data structures used to describe brokering configuration are regions, customers and CDNs. The region data structure, as described in section 4.1.2, is used to cluster IP addresses into a more manageable set of regions. Customer entries consist of a brokered domain name, an origin domain name, and a set of CDNs that the customer may use. CDN

entries consist of a name, capacity and current usage (expressed as bandwidth), the quality with which a CDN covers a given region, and a redirection method. Each region is assigned a coverage value indicating how well the region is covered by that CDN and this information is used for load balancing as described in the next section. The redirection method must be one of "address," "NS," or "CNAME."

Generally speaking, the control element configuration described by these data structures is dynamically adjusted by the control component using input from the control agents (see Section 4.2.3).

### 4.2.2 Load Balancing

The main purpose of the control component is to asynchronously recalculate and update the tables used by the DNS engine according to current loads and a given policy.

The load balancing algorithm has to determine the mapping of  $\langle \text{region}, \text{set of CDNs} \rangle$  to a probability distribution. This means that a client in the specified region will be redirected to a particular CDN with the given probability if the client requested the Web site of a customer which uses a particular set of CDNs. To provide protection against flash crowds while generally directing clients to nearby caches, the probability distribution has to be chosen carefully—considering how well a CDN covers a particular region and how much capacity is available.

Given the number of variables, the lack of predictability of client demand, and the possibility of stale and incomplete feedback, the design of a robust and scalable control algorithm is a non-trivial task. Currently, our algorithm takes the following steps when considering where to send clients from a given region:

1. Eliminate all CDNs that are overloaded
2. From the remaining CDNs, keep those that serve this region best as determined by their coverage values
3. For the CDNs identified in step 2, distribute the load among them based on any available information on current capacity.

While we do not claim that this algorithm is optimal it is unclear how much improvement would be obtained from more sophistication unless the load feedback mechanism (described next) is able to provide more precise and immediate load information than is currently available. Finding such an optimal algorithm is outside the scope of this paper.

### 4.2.3 Agents

As shown in Figure 2, the Control component receives input from the following three agents:

**Configuration Agent:** initializes the Control component with the appropriate configuration.

**Management Agent:** has a Web-based GUI interface that allows run-time editing of various aspects of the control component.

**Load Agent:** provides load information retrieved from participating CDNs. We use a structured feedback protocol, over a secure channel, to report load-related information to the load agent. Load reports include request rates and bandwidth consumption, as well as available capacity, both aggregated and on a per-region basis.

## 5 Performance

The costs and benefits of CDN brokering in general, and our IDNS implementation in particular, can be grouped into three categories. The first category is the performance of the brokering DNS server. The performance of a single DNS server directly influences the number of DNS servers required to support a particular site and, therefore, the capital cost of the deployment. The second category is the delay imposed by the additional level of redirection necessary to support brokering. This additional delay is the major potential cost of CDN brokering in terms of overall latency as perceived by a client. The third category consists of the potential benefits of CDN brokering: increased capacity, reduced cost, higher fault tolerance and better performance. Since we were not able to test the first three benefits in our live network, we will concentrate on a case study of the possible performance benefits.

### 5.1 IDNS Performance

Since there is no established benchmark for evaluating the performance of DNS servers we will focus on the rate of DNS requests the server can answer successfully to measure its performance. This rate determines how many IDNS servers are required to support a given client population.

#### 5.1.1 Test Setup

To measure DNS request rates, we connected an IDNS server running FreeBSD 5.0 on a Dell PowerEdge 2450 Server with one 730MHz Pentium III processor and 1GB of RAM to a set of five DNS client simulators via gigabit Ethernet. To determine

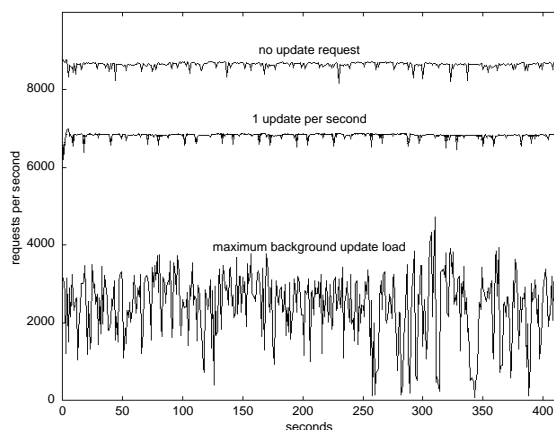


Figure 3: IDNS performance

the best performance with an idle control component and an idle set of agents, we measured the maximum number of requests the server can sustain while supporting a configuration of five CDNs, 50 customers, and 250 regions.

In addition to performance without update load, we repeated the experiment while generating one load agent, capacity agent, or management agent update per second on the control component. Note that in this test we did not allow updates to the DNS engine to be aggregated (this would have reduced the load). The generated management updates randomly changed the number of CDNs, customers, and regions in the range from 1-10 CDNs, 1-100 customers, and 1-500 regions. This experiment represents the behavior of an extremely busy IDNS server. Note that IDNS scales to a real “user community” by deploying a number of servers in proportion to the customer load.

We also examined the worst-case behavior of the IDNS server by using a control component that generates updates as fast as possible. In this test, the number of CDNs, customers, and regions used is in the same ranges as above.

#### 5.1.2 Results

Figure 3 shows the rate of successful DNS requests as a function of time. Both the experiment with no control updates and with one update per second show little variance and can sustain 8700 and 6800 requests per second respectively. Not surprisingly, the maximum control load experiment shows high variance, but the average rate is still on the order of 2000 requests per second.

Overall, the rate of 6800 requests per second in a realistic setting is fast enough to support multiple brokered sites. At that

rate if a DNS timeout of five minutes is used by IDNS, an IDNS server can handle 2,040,000 client DNS servers (more than currently exist in the Internet). Since RFC1034 [12] requires two DNS servers for any site on the Internet, a realistic setup could serve up to 4,080,000 client DNS servers. The load on an individual IDNS server is even further reduced by the fact that some major browser implement in-memory DNS caching with essentially no timeouts.

## 5.2 Redirection Overhead

Our second set of measurements help determine the overhead of additional DNS lookups imposed by the extra level of redirection necessary to support brokering. To evaluate this overhead we used six test Web sites and measured their performance from the view of almost 25,000 client DNS servers. (This list of client DNS servers was acquired using packet traces.)

### 5.2.1 Test Setup

The test setup consists of six test sites and a measurement machine. On the measurement machine we used the `dig` DNS tool to issue requests to a list of 24,712 DNS servers. The six test sites fall into the following three categories:

**Regular Site:** One test site is a regular Web site included as a base case. The DNS server for this site directly returns an A record of a single Web server.

**CDN Sites:** Four of the test sites each use their own real-life CDNs to serve their content. These sites are used to show the typical DNS delay across four regular CDNs. Each of these sites use a CNAME record to redirect the DNS resolution of a client DNS server to their CDN. The CDNs then resolve the CNAME in accordance to their DNS-based redirection mechanism. Our results are averaged across the four CDNs measured.

**Brokered Site:** The final test site is brokered using IDNS. On this site, a client DNS server first requests the resolution from the authoritative DNS server of the site which redirects the client DNS server to the IDNS server using a CNAME. The IDNS server in turn redirects the client DNS server to one of four CDNs chosen randomly from the four CDNs measured above. Thus, the average of the four CDN sites is directly comparable to the results returned by the brokered site.

To measure the DNS delay of resolving the six test sites, we request a recursive resolution of the test site domain name twice in short succession from each client DNS server on our list. The

Measurement	Success Rate	Average Delay	Standard Deviation
Regular site	99.3%	260 ms	36 ms
CDN average	96.0%	842 ms	22 ms
Brokered site	92.7%	993 ms	81 ms

Table 1: DNS delay with cold client DNS server cache

Measurement	Success Rate	Average Delay	Standard Deviation
Regular site	99.8%	218 ms	29 ms
CDN average	97.3%	756 ms	83 ms
Brokered site	93.3%	843 ms	103 ms

Table 2: DNS delay with lukewarm client DNS server cache. The deviation was measured over ten runs.

first request triggers a resolution of the domain name using the remote DNS server as client DNS server. The second request is fulfilled from the cache on the remote DNS server and is used to calculate the delay from our measurement machine to the remote DNS server being probed. Thus, the delay imposed by the DNS resolution as seen by the remote DNS server is the difference between the first and second measurements.

We performed this experiment in two settings:

**Cold Client DNS:** In the cold client DNS experiment, we waited for more than two days between each run of the experiment to ensure that it was unlikely that the remote DNS server had any parts of the DNS records required to fulfill the resolution cached. A delay of two days is necessary because the root and top-level name servers specify a two-day TTL on the top-level DNS records.

**“Lukewarm” Client DNS:** In the “lukewarm” client DNS experiment we performed each experiment twice, first to prime the caches of the client DNS servers, then 70 minutes later to measure the “lukewarm” performance. A delay of 70 minutes times out the CNAME and A records; however, it most likely retains the supporting DNS records in the cache of the client DNS server.

The first experiment represents the behavior for a first hit to a site from a DNS server. The second experiment represents a subsequent request to the site and is therefore more representative of what a real user would perceive on a busy site.

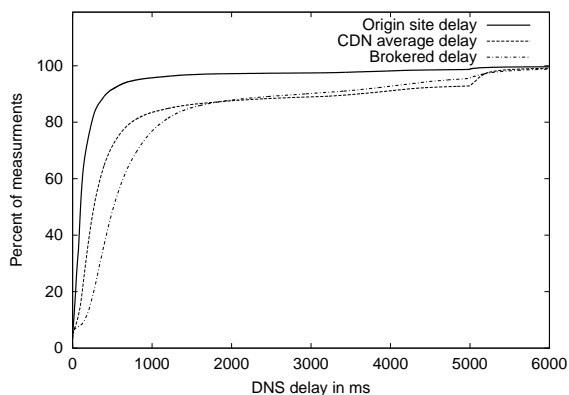


Figure 4: DNS delay with cold DNS cache

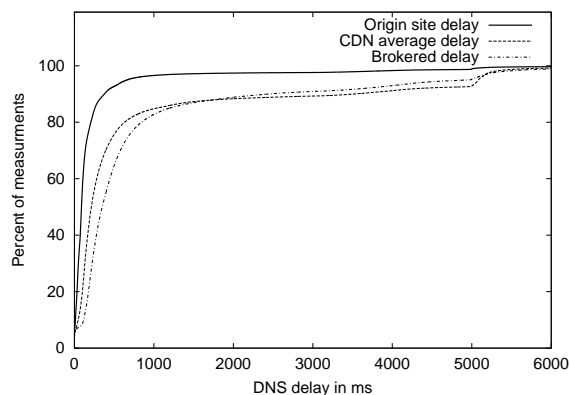


Figure 5: DNS delay with lukewarm DNS cache

### 5.2.2 Results

Table 1 and Table 2 summarize the results of the DNS delay measurements.

The first column indicates the percentage of DNS resolutions for which `dig` returned a valid result. `dig` may not succeed if too many of the DNS datagrams required to recursively resolve the DNS name are dropped by the network. Another source of unsuccessful requests occurs when the request sent from the measurement machine to the remote DNS server is dropped. It is not surprising that the more complex resolution of a DNS name pointing to a CDN has a higher chance of a unsuccessful resolution.

Overall, the delay added by brokering on cold client DNS servers is on the order of 150 ms. In the more realistic case of lukewarm client DNS servers, the delay added (about 87 ms) is substantially smaller. It is interesting to observe that the gap between the CDN accelerated Web site and the origin site on a cold client DNS server is the order of 582 ms and in the case of lukewarm client DNS servers, the overhead imposed by CDNs is on the order of 538 ms.

Figure 4 and Figure 5 show the cumulative delay distribution of the cold and hot client DNS server experiment. In both figures the delay distributions for the CDNs and the brokered site show a bend at the five second mark. This represents a DNS timeout triggered by the loss of one or more DNS datagrams. Comparing the delay distributions, the higher average delay of the CDN and brokered sites seem to originate from the larger number of such DNS timeouts and the associated resolution time of more than five seconds.

## 5.3 Overall Performance

In addition to the potential benefits of increased CDN capacity, reduced cost and better fault tolerance, brokering may also offers the opportunity to improve the performance of a Web site. To illustrate this point we performed a case study on two real CDNs: *G* and *H*. We refer to these CDNs anonymously because our purpose is to demonstrate the merits of brokering rather than to comprehensively evaluate specific CDNs (as in [10]). Note that neither CDN is owned by AT&T.

### 5.3.1 Test Setup

First we establish the performance of *G* and *H* in different locations. To do this we downloaded two Web pages hosted by the CDNs from eleven different locations. The Web site used for *G* has a size of 39179 bytes and the Web site of *H* contains a total of 43944 bytes. Both pages consisted of one main HTML object containing thirteen embedded images. The test browser in each location first downloaded the HTML page and then downloaded the embedded images using four parallel connections. The experiment was performed at the same time for each CDN and repeated ten times.

Figure 6 shows the results of the experiment. For each CDN, two values are reported. The values reported in Figure 6(a) are the initial delay until the first byte is received, not including the DNS lookup delay. The values reported in Figure 6(b) are the throughput on the application level in KBytes per second. The results show that neither *G* nor *H* performed better than the other in all eleven locations. In fact, in terms of throughput, *G* performed better in five locations whereas *H* performed better in six locations. The data shows that CDN brokering can be used

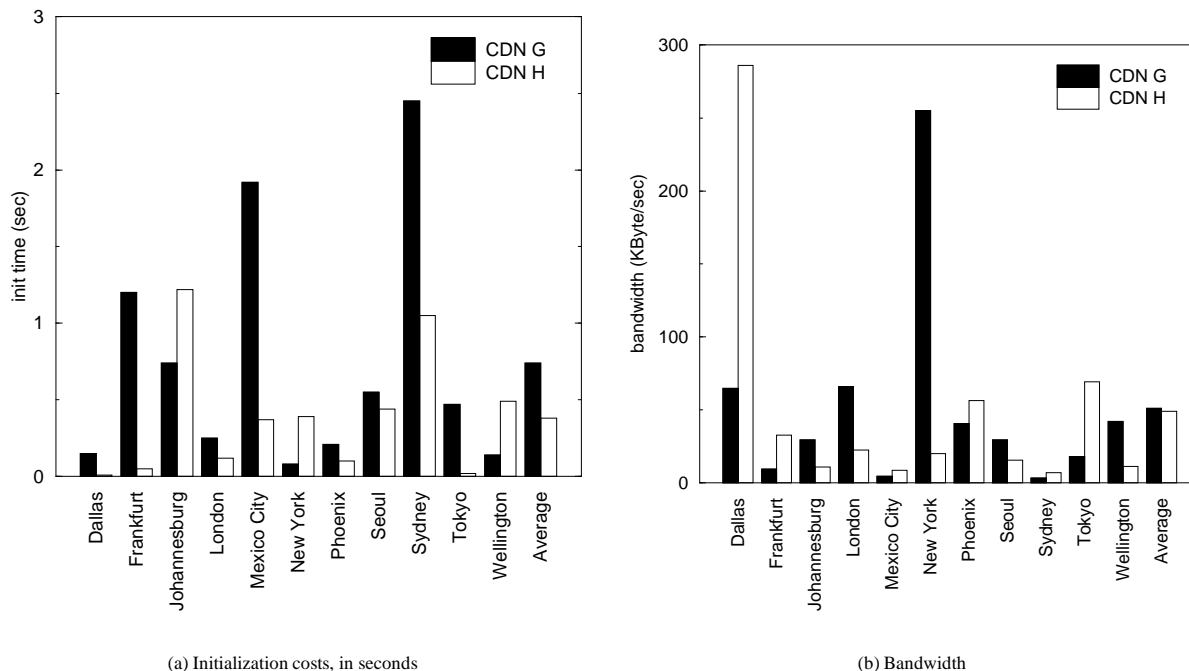


Figure 6: Measurements for each CDN in each location

to leverage the strength of both CDNs. Under the assumption that IDNS redirects each client to the better performing CDN, the average initial delay would decrease to 0.30 seconds and the average throughput would increase to 82,122 bytes per second. Compared to the 50,342 and 52,432 bytes per second each individual CDN can achieve, this is a substantial improvement.

As our case study shows, using two real CDNs measured from eleven locations, CDN brokering may not only provide benefits in terms of overload protection, cost and fault tolerance, but it also can provide performance benefits. However, it should be noted that the case study represents a limited measurement, and different CDNs and/or client sites will undoubtedly provide different results. Over time we expect to support many more types of customers using a variety of partner networks, as we further refine our algorithms and architecture.

## 6 Related Work

Our content brokering proposal involves the symbiotic cooperation of multiple autonomous CDNs. Two recently announced

efforts also aim at cooperation between autonomous content-delivering platforms. Content Bridge [6] is a standardization effort by an alliance of companies. Its goal is to allow cooperation between reverse proxy platforms (such as CDNs) and forward proxy platforms, so that CDNs ensure freshness of their content cached by forward proxies, while forward proxies provide the CDNs with hit statistics of CDNs' content in return. The *content bridge* is a middleware layer between CDNs and forward proxies that facilitates this type of cooperation. In contrast, CDN brokering enables cooperation between CDNs with the ultimate goal of improving access to their content. Thus, the two proposals are orthogonal and complimentary.

The Content Alliance [5] is a standardization effort, organized initially by Cisco, whose goal is to develop standards for inter-CDN cooperation of the kind exemplified by our CDN brokering system. We are actively participating in this effort, which incorporates many of our ideas.

DNS-based load balancing has been widely used for load distribution among Web servers or caches; various products in this area include Alteon's GSLB [1] and Cisco's Distributed Director [4]. These companies disclose few details about their load

balancing algorithms. Unlike these DNS-based implementations, CDN brokering uses DNS to load balance among entire CDNs.

Finally, Shaikh and Tewari [15] consider the problem of measuring the network proximity of client DNS servers with HTTP clients. This issue obviously has direct relevance to CDNs, which crucially rely on the assumption that the origins of a DNS query are indicative of the origins of the corresponding HTTP request. Our implementation of CDN brokering relies on this assumption as well, since it is also DNS-based. If this assumption is definitively shown to not hold, it will provide extra motivation for brokering using alternate mechanisms.

## 7 Conclusion

Content distribution internetworking is a new area, and the techniques and issues involved are actively being explored by ourselves and others. Here we have attempted to motivate the need for CDI: rather than one service trying to be omnipresent, which is demanding in both capital and manpower, a CDN can leverage agreements with other networks to provide good performance at an appropriately low cost. CDN brokering is a form of CDI that uses DNS outsourcing to permit a specialized DNS server to select a CDN at the time of hostname resolution.

IDNS, our patent-pending system for CDN brokering, is the first working example of this functionality of which we are aware. We have measured the scalability of the IDNS DNS server and the overhead of adding an extra level of indirection through the brokerage. We are currently supporting a limited number of AT&T customers using IDNS.

## 8 Acknowledgments

We would like to thank Kobus van der Merwe and Zhen Xiao for reading earlier drafts of the paper and giving us valuable comments. We would also like to thank the anonymous reviewers, who provided valuable feedback.

## References

- [1] Alteon Web Systems, Inc. Enhancing web user experience with global server load balancing White paper. [http://www.alteonwebsystems.com/products/white\\_papers/GSLB/index.shtml](http://www.alteonwebsystems.com/products/white_papers/GSLB/index.shtml).
- [2] A. Barbir, B. Cain, F. Douglis, M. Green, M. Hofmann, R. Nair, D. Potter, and O. Spatscheck. *Known CDN Request-Routing Mechanisms*, Feb. 2001. Work in Progress, draft-cain-cdn-known-request-routing-01.txt.
- [3] B. Cain, P. Rzewski, and N. Robertson. *Cross-Network Accounting for HTTP*, Nov. 2000. Work in Progress, draft-rzewski-cnacct-00.txt.
- [4] Cisco Systems, Inc. DistributedDirector. White paper. [http://www.cisco.com/warp/public/734/distdir/dd\\_wp.htm](http://www.cisco.com/warp/public/734/distdir/dd_wp.htm).
- [5] Content Alliance. <http://www.content-peering.org>.
- [6] Content Bridge. <http://www.content-bridge.com>.
- [7] M. Day, B. Cain, and G. Tomlinson. *A Model for CDN Peering*, Nov. 2000. Work in Progress, draft-day-cdn-model-04.txt.
- [8] M. Day and D. Gilletti. *Content Distribution Network Peering Scenarios*, Nov. 2000. Work in Progress, draft-day-cdn-scenarios-02.txt.
- [9] M. Green, B. Cain, G. Tomlinson, and S. Thomas. *CDN Peering Architectural Overview*, Nov. 2000. Work in Progress, draft-green-cdn-gen-arch-02.txt.
- [10] K. L. Johnson, J. F. Carr, M. S. Day, and M. F. Kaashoek. The measured performance of content distribution networks. In *Proceedings of the 5th International Web Caching and Content Delivery Workshop*, May 2000.
- [11] B. Krishnamurthy and J. Wang. On network-aware clustering of Web clients. In *Proceedings of ACM SIGCOMM 2000*, pages 97–110, Aug. 2000.
- [12] P. V. Mockapetris. RFC 1034: Domain names — concepts and facilities, Nov. 1987.
- [13] R. Nair, D. Gilletti, and J. Scharber. *CDN Peering Authentication, Authorization, and Accounting Requirements*, Nov. 2000. Work in Progress, draft-gilletti-cdn-aaa-reqs-00.txt.
- [14] Y. Rekhter and T. Li. RFC 1771: A Border Gateway Protocol 4 (BGP-4), Mar. 1995.
- [15] A. Shaikh and R. Tewari. On the effectiveness of DNS-based server selection. IBM Research Report RC 21785, IBM Thomas J. Watson Research Center, 2000.