Architecture for Open Content Delivery Networks

Zvi Rosberg and Rami Mukhtar

ARC Special Research Centre for
Ultra-Broadband Information Networks
Department of Electrical and Electronics Engineering
The University of Melbourne
Victoria 3010, Australia

March 15, 2002

Abstract

We describe a new architecture for an open Content Delivery Network (CDN) built from a single software component, Request-Redirector, collocated with the web server. The Request-Redirector encapsulates all the functions that a CDN requires. In particular, a mechanism to redirect client requests to edge-servers; a local network proximity database used to select the best edge-server; and a mechanism to embed a JavaScript probing program in the response message that reports back network proximity measurements. Conversely to existing solution where edge-servers are tightly coupled with other CDN components, in ours they constitute a confederation of independent servers conforming to a standard and simple interface.

Keywords: Internet, World Wide Web, Content Delivery Network, Caching Proxy, JavaScript

1. Introduction

The World Wide Web (web) is defined by a set of protocols and standards that facilitate the publishing and distribution of multimedia content on the Internet. Publishers place content on web servers (*origin-servers*), and end-users (clients) use web client applications and a variety of tools to retrieve static, dynamic and streaming media content. The growing number of clients,

servers and multimedia applications (e.g., live and streaming audio and video) necessitates a distributed method for content delivery that is efficient and scalable.

Content Delivery Networks (CDN) are an increasingly popular mechanism by which content can be efficiently delivered to the client. A CDN is a network of servers (*edge-servers*), which maintain and serve all or part of the origin-server's content from geographically distributed locations. Edge-servers are placed strategically close to client aggregation centers; hence shortening their routes to the content. Some prominent CDN services include Akamai's CDN [AKAM], Digital Island (a subsidiary of Cable & Wireless Inc., [DGIS]), and others [KWZ]. Most of these services rely on proprietary technologies, and in general do not interoperate.

Existing CDN services are comprised of hundreds to thousands of edge-servers that are distributed amongst many ISPs in multiple Point of Presence (POPs). In order to ensure that client requests are redirected to the edge-server that can best serve the request, it is necessary to maintain an up to date knowledge base, which is dynamically updated using real time measurements.

Until now, the majority of CDN deployments have been in the form of proprietary deployments. The focus of this paper is to present an inexpensive and open CDN architecture based on a novel client side measurement technique. We believe that this architecture will facilitate the deployment of open CDN networks with the same convenience as Peer to Peer networks have enjoyed. We compare and contrast the advantaged of the proposed architecture against other architectures and justify a potential performance improvement using measurements on existing services on the Internet.

2. The Need for a New Architecture

Motivation

Site mirroring and client side caching proxies are still popular alternatives to CDNs. Site mirroring is usually done on a single content publisher basis that replicates its entire origin site into several geographically dispersed web servers called mirror sites. The mirror sites are usually not designed to be shared among several content publishers and therefore considered as an expensive and hard to administer solution.

At first, client side caching proxies appear as an attractive solution, which raises the question why deploy CDNs? The answer lies in the manner by which caching proxies are deployed and operated. While mirror sites operate on behalf of the content publishers and are fully controlled by them, caching proxies operate on behalf of the clients. Caching proxies are usually attached to a client's local network or located at client aggregation centers (ISPs, IAPs and enterprise networks). These locations are as good as a content publisher may hope for. However, the content publishers are totally dependent on the network service providers, who control the cache locations, capacities and content consistency. Some content providers cannot afford to depend on a completely independent third party to administer the delivery of their content.

One solution to this problem is to enhance caching proxies to behave as edge-servers based on service level agreements between the service providers and the content publishers. Although economical incentives do exist, lack of reasonable priced enabling technologies has prevented ISPs to operate edge-servers in this way. This vacuum has been exploited a couple of years ago by the current CDN services that have offered client side web caching services in a form of a

(very complex and expensive) CDNs. It is not clear whether the current CDN solutions serve the best interest of the content publishers (who pay for the service) or the client community. We argue that a new CDN architecture [ROS] is required, which provides ISPs and content providers an effective and economical means to cooperatively offer a high quality content delivery service.

Our architecture is motivated by the observation that most ISPs already have caching proxies that can easily be enhanced to operate as edge-servers. Therefore, by merely enhancing origin-servers to intercept client requests and redirect them to the closest edge-server, one would get a complete CDN solution.

Advantages

Decentralized Architecture

Existing services interact only with their own CDN edge-servers, the proposed architecture uses an open interface that can interact with any edge-server that complies to the interface specification. This facilitates simple CDN peering, enabling any edge server to serve any content provider. Since edge-servers represent the bulk of the investment, this increases the economic viability of CDNs. Furthermore, since existing CDN solutions comprise a vast number of tightly coupled and disperse components, their deployment and management is extremely complex. In our solution, every request redirector and every edge-server joins and leaves a CDN on an individual basis; thus no central deployment and/or management is required. Furthermore, our novel client side measurement scheme can independently measure edge-server performance, and hence does not require any sophisticated schemes to translate an edge servers load (a machine dependent measure) to client access times.

Flexible Subscription

The commitment that current CDN services require from content publishers (especially those that require the replacement of embedded URLs), is entirely alleviated by our architecture. Each content publisher owns its request redirector, and is free to negotiate at any time any service agreement with any of the edge service providers. This also enables a content provider to custom tailor the CDN subscription to specific requirements. For example, a content provider may choose (at a single server granularity) to subscribe to particular edge-servers that best serves that particular clients needs.

Improved Routing Decisions

A further advantage of our CDN architecture is the novel measurement subsystem. Existing solutions use either static measures (e.g., route hop-count) or ping-based estimators to select the closest edge-server. We propose a metric based on the loading times of test objects that are embedded in the content measured by the requesting clients. Since static proximities are not sensitive to traffic fluctuation, and ping-based measurements are not reliable due to the "ping shaping and blocking" mechanisms in routers and servers, we argue that our method selects the edge-server based on a metric that is closely correlated with the user perceived loading time. The loading time measurement applies an idea presented in [RaEl] in which a client side JavaScript program sends object loading requests and tracks their loading times.

3. A New CDN Concept

The Request-Redirector

The main component of our architecture is the *request-redirector*; a single device collocated with the origin-server that encapsulates all the functions of a conventional CDN. It intercepts client requests arriving at the origin-server and decides how to process them based on their requested content, source IP address, and possibly other information they may carry. Selected requests are redirected to one out of multiple edge-server candidates distributed across the Internet. To ensure optimal redirections, the Request-Redirector manages a local database mapping client clusters to edge-server candidates.

The Request-Redirector can be packaged in several forms depending on the setup at the content publisher site. One form is an auxiliary software *plug-in* installed in the same host that runs the web server program. This form does not require recompilation of the web server program and is suitable for e.g., Microsoft's IIS and Netscape Enterprise web server programs. Another form is an extension module added to a web server program. This form does require re-compilation of the web server program (e.g., the Apache web server). In a third form, the Request-Redirector runs as a separate reverse proxy process (in the same or another machine) and gets all requests first.

Ignoring the transactions required to resolve a domain name into an IP address, Figure 1 illustrates all of the application layer transactions required for a client to fetch content from a server using a protocol such as HTTP, HTTPS, RTSP or MMS. Figure 2 illustrates the transactions required to fetch content from a CDN network employing *request-redirection*. Request redirection increases the number of transactions required to fetch content. However, the origin server is alleviated from the burden of subsequent requests. The interception, redirection and edge-server selection are the main roles of a CDN. We now describe key components of our architecture that enable a *request-redirector* to redirect the client to the most appropriate edge server.

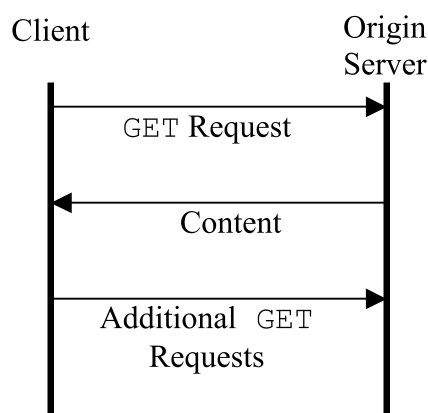


Figure 1: Traditional client-server transaction.

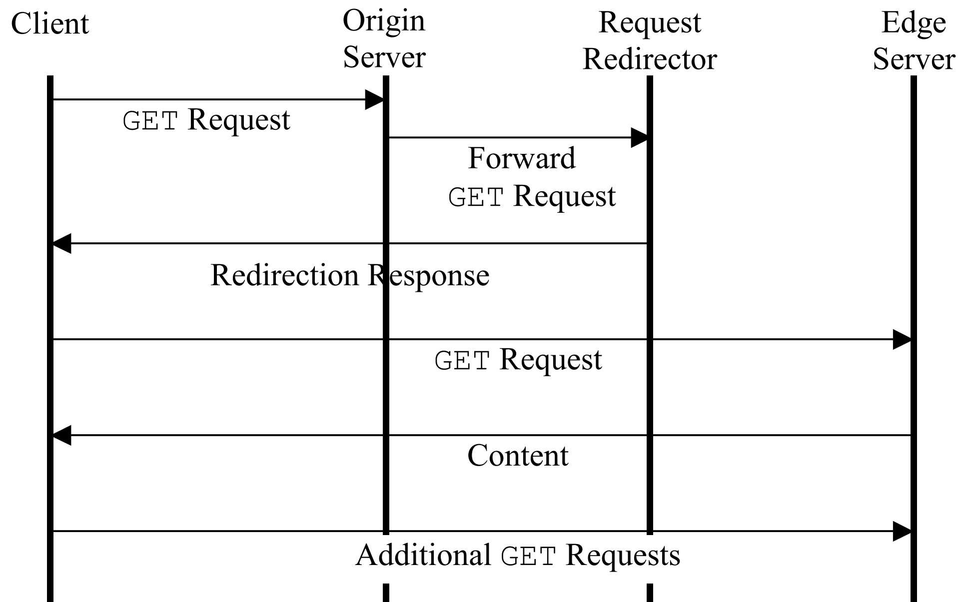


Figure 2: CDN client-server transaction using the simplest form of request redirection.

Client Clusters

A major challenge is to select the best edge-server for every particular client request. Maintaining a database that contains an entry for each of the 2^{32} possible clients is unfeasible. Hence, clients must be mapped into a relatively small number of *network-vicinity* groups; a set of IP addresses that are close to each other with respect to some network distance measure. Furthermore, since not all edge-servers may serve the same content types, network-proximity groups must be further refined into *request-groups* that differentiate client requests by their requested content and access protocol. Given the request-groups, edge-server selection is a dynamic mapping from a request-group to an edge-server that is dependent on network congestion and edge-server loads.

Client Probe Program

A key part of the proposed CDN architecture that distinguishes it from other implementations, is a Novel client side measurement scheme. On making a request for content, a client may be randomly selected to measure the loading times of objects embedded into the content, which are retrieved from different edge-server candidates. The results of the measurements are then returned to the request-redirector using a *cookie*. By this mechanism, clients are directly used to measure which edge-server provides the best service at any instant. A standard *web-cookie* [COO] may be sufficient for this purpose. However, they have two limitations: (i) they can be disabled in the client browsers; and (ii) they are returned only in subsequent message requests. These two limitations are overcome by the use of the *instant-cookie* method that is described in Section XX. We shall show in Section XX that the loading times are highly correlated with both the edge-servers load and the level of the network congestion.

Virtual CDNs

Since our architecture is open and distributed, there is no notion of edge-servers belonging to a particular CDN service provider. Virtual CDNs are implicitly created by edge-servers subscribing to provide content distribution services to an origin server on an individual basis. In order to facilitate this interaction, transactions need to be standardized such that transactions between the origin-server and edge-server are secure and compensation agreements are facilitated. In very large systems, a broker may be required to provide Authentication, Accounting and Authorization (AAA) services. We describe how these issues are resolved in Section XX.

4. Implementation

The Request-Redirector

A schematic representation of the Request-Redirector is presented in Figure 3. The client-probe-program generator creates and appends a JavaScript program to selected client response messages. The cookie and measurement handler receives the measurements and maintains the response measurement database that is in turn used by the edge-server selector for optimal redirection decisions. The network proximity measurement mechanism obtains information on the clustering of IP addresses; this information is then used to maintain the Network Proximity Database. Figure 4 illustrates typical transactions between components of the request-redirector.

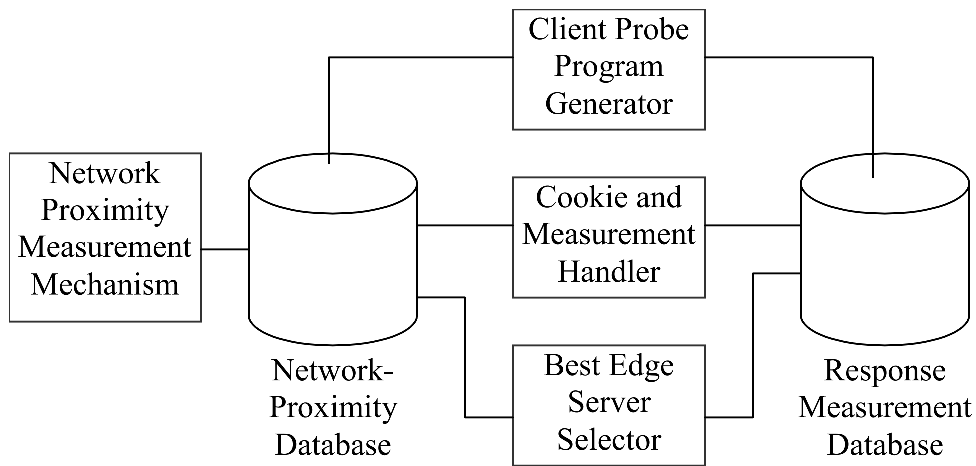
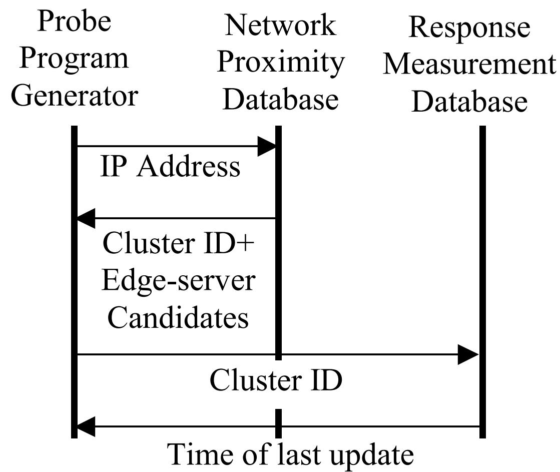
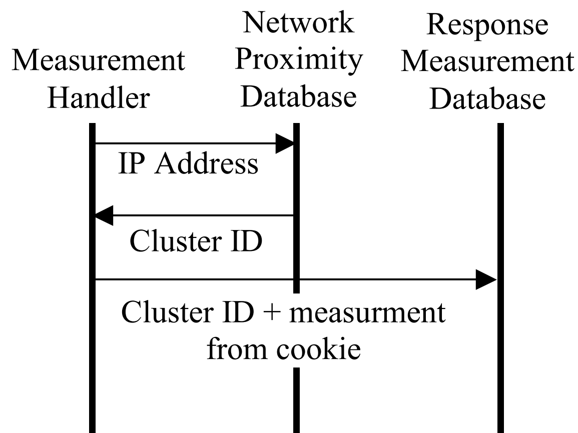


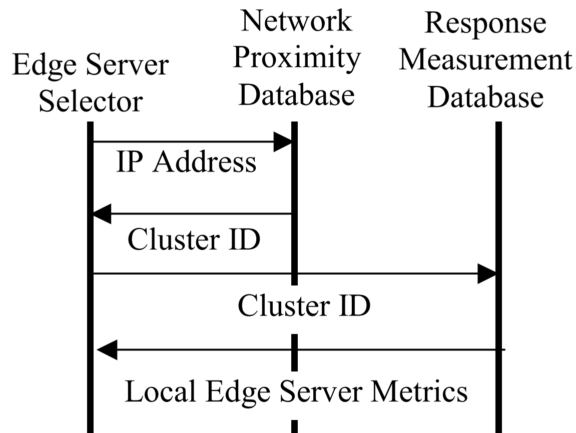
Figure 3: Request-Redirector sub-components.



(a) Generation of a probe program in response to a client request



(b) Updating the response measurement database using content in a client cookie



(c) A client request redirection

Figure 4: Typical Transactions between components of a Request-Redirector

Request Processing

The Request-Redirector module intercepts requests arriving at the origin-server and decides how to process them. Client requests are intercepted by the Request-Routing module and are classified into one out of the following types based on the content identifier, cookie content and

the client IP address. The request types are illustrated in Figure 5, requests of type 1-3 are used for CDN management and data collection, whereas requests of types 4 and 5 deal with clients request for content. Each request type is described in detail below.

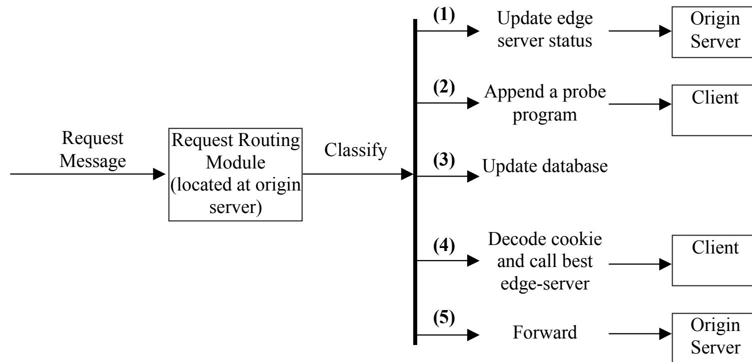


Figure 5: Request-Routing processing.

Requests of Type 1 update the edge-server status and are then forwarded to the origin-server. Note that such requests relieve the Request-Routing from monitoring the edge-servers since edge-servers must occasionally refresh their caches using Type 1 requests. Furthermore, by controlling the expiry times of the response headers (“`Pragma: no-cache`“; ”`Cache-Control: max-age=0`” and “`Expires:`”, see [1]), the edge-servers can be forced to send type 1 requests sufficiently often.

Responses to requests of Type 2 are appended with a probe-program implemented with JavaScript or as a Java applet. This program is then interpreted by the client browser, which measures loading times from every candidate. Measurements are then reported to the Request-Routing host either by a standard web-cookie or by requests of Type 3.

Requests of Type 3 carry measurement information decoded from their URI string and/or from the message body (if exists) and update the local database.

Request of Type 4 are processed as follows. Firstly, existing cookies are decoded and update the local database. Then the best edge-server is selected based on the request cookie (if it exists) and/or the local database. Last an HTTP or RTSP redirection response message ("302" status code) is sent to the client.

Requests of Type 5 are simply forwarded to the origin-server that processes them as if the Request-Routing module is not present. Figure 2 illustrates the transactions required for a typical client request.

The Edge Server

Each edge-server provides a request-redirector with an initialization file containing IP address clusters it offers to serve, along with their associated average route hop-count to the edge-server. By setting and monitoring the Time To Live (TTL) fields in probe packets, an edge-server can obtain hop counts to each IP address cluster.

Other proximity measures such as round-trip-time and packet loss can be added to the initialization files. In this case they can be merged by prioritizing each measure, or by averaging them into a common utility scale.

When an edge-server intercepts a redirected request for content it must ascertain from which origin-server the request originated. This can be achieved, if we use a standard URI format in the `"Location:"` header field of the redirection response message. E.g., `<scheme://><edge_server>/<origin_server>/<abs_path>`, where `<scheme://>` is the protocol name, `<edge_server>` is the edge-server address, `<origin_server>` is the origin-server address, and `<abs_path>` is the absolute path to the content. Each redirected request that arrives to an edge-server will then contain the address of the origin-server.

An alternative method for encoding the origin server's identification into the request is to set the `<edge_server>` in the `"Location:"` header to an alias name that uniquely identifies its origin-server. Since most clients set the alias name in their request `"host:"` header field, the origin-server could then be identified by each edge-server. Note that this option is more appealing than the first one since the redirected URI would not need encoded data. Moreover, since most existing servers support the notion of "virtual-hosts" and "proxy-pass" configuration parameter (see Apache server), they would not need any enhancements – just configuration.
Also note that the HTTP 1.1 standard requires the `"host:"` header field, therefore it is a matter of short time until the third requirement will become void.

The Client-Probe-Program

A fundamental problem in CDN is to collect and maintain measurements about network congestion and edge-server loads. Existing CDN solutions place special monitoring devices at the edge-servers and throughout the Internet, and digest their measurements in a central database. In our system, they are the clients who arrive at each Request-Redirector that silently collect the measurements and report them back to their activating Request-Redirectors. Thus, each Request-Redirector maintains its own local database for its clients and their corresponding edge-server candidates.

```
<HTML>
<HEAD>
<title>Loading Time Measurement Test</title>
<SCRIPT language=JavaScript>
<!--
// 5.1
var start;
var stop;

// 5.2
function noteTime() {
stop = new Date()
diff = stop.getTime() - start.getTime();
alert(diff+' milliseconds');
}
// -->
</SCRIPT>
</HEAD>

<BODY>
<P><B>Clear the local memory and disk cache of your browser and mouse over the
links below to measure its loading time.</B><HR>
```

```
<P>
<CENTER>
<!— 5.3 -->
<IMG OnLoad='noteTime();' src="" alt="loading_image" name="image_one" width="150"
height="100"><BR>
</CENTER>
<HR>
<!— 5.4 -->
<A onmouseover="start = new Date();
<!— 5.5-->
image_one.src='http://edge.domain1.com/test.gif';"
href="'http://edge1.domain1.com/test.gif'">Cache Proxy 1</A><BR>
<HR>
<A onmouseover="start = new Date();
<!— 5.6 -->
image_one.src='http://edge2.domain2.com/test.gif';"
href="'http://edge2.domain2.com/test.gif'">Cache Proxy 2</A><BR>
<HR>
<A onmouseover="start = new Date();
<!— 5.7-->
image_one.src='http://edge3.domain3.com/test.gif';"
href="'http://edge3.domain3.com/test.gif'">Cache Proxy 3</A><BR>
<HR>
</BODY>
</HTML>
```

Figure 6: A JavaScript code example to measure image loading times from multiple locations

The measurement device is the client-probe-program. It is written in a scripting language or implemented as an object code that can be embedded into a response message. When the client browser loads the response, this program gets executed in the client host and measures the loading times of some pre-configured objects from a designated set of edge-server candidates. In the preferred scenario this program attempts to store the measurement data onto the client hard drive using a standard web-cookie. In addition or alternatively, the program sends the measurement data back to its activating Request-Redirector using the instant-cookies described below.

Note that each client may have a different set of edge-server candidates (which may also change in time). Therefore those candidates are set in the client-probe-program on-the-fly. Also, to cope with measurements from different time zones and unsynchronized clocks, the Request-Redirector sets its local time in each client-probe-program, which is later used as the measurement time stamp.

Figure 6 demonstrates a client-probe-program implemented with JavaScript. The program measures image loading times from three different URI locations (defined by 5.5, 5.6 and 5.7 in the figure) when the user passes the mouse over the link defined by 5.4 in the figure. On mousing the link, the start time is set to `var start` (defined by 5.1 in the figure) and when the image loading completes, the event handler (defined by 5.3 in the figure) calls the function `noteTime()` (defined by 5.2 in the figure) that computes the loading time.

Redirection Cookies

We use two methods to pass the measurements from client-probe-programs to their activating Request-Redirectors. One is a standard web-cookies and the other is a new mechanism (instant-cookie) that resolves two limitations cookies have.

Web-cookies

A web-cookie, [COO], is a piece of information sent from a web server to a web browser that the browser is expected to save and to return to the web server in subsequent requests. Depending on the browsers' settings, the browser may accept or reject the cookies, and may save them onto its hard disk for a specified amount time after which they expire. Cookies can be sent in the header of HTTP, HTTPS and RTSP messages; and can be set, modified and retrieved by client-side scripts or object codes embedded in the response messages. A cookie contains a directory path to the origin-server that tells the browser where to send it, and information written in a series of "NAME=VALUE" pairs.

The Request-Redirector uses cookies (in the headers and embedded JavaScript programs) to set the following redirection data:

- Edge-server candidates for the respected request-group.
- Loading times from each candidate as measured by the client-probe-program.
- Request-Redirector time stamp.
- Cookie type that distinguishes between local measurements and estimated values based on other client measurements.

Figure 7 illustrates a JavaScript sample code (along with self-explained comments) that silently sets a standard web-cookie in the client browser hard drive.

```
// server time set by the Request-Redirector
// Date(yr_num, mo_num, day_num, hr_num, min_num, sec_num)
var server_time = new Date(01, 12, 09, 13, 27, 45);
// Edge-server proxy candidates set by the Request-Redirector
var urls = new Array("http://cache1.domain1.com/test.gif",
                     "http://cache2.domain2.com/test.gif",
                     "http://cach3.domain3.com/test.gif");
// Edge-server preference order initialized with zero.
// Set by the Client-Probe-Program timing function.
// E.g., ("3","1","2") indicates that edge2 is the best,
// edge3 is second best and edge 1 is the worst.
var url_order = new Array("0","0","0");
// Loading times (ms) from each edge server. Initialized with zero
// and set by the Client-Probe-Program timing function.
var loading_times = new Array("0","0","0");
// Cookie path and cookie domain. Set by the Request-Redirector.
var path   = "; path=/;";
var domain = "; domain=origin_server_domain.com;";

function setCookie()
{
    // concatenate cookie values
    //
    // Add cookie type and separate with ":".
    var value = ""LOCALLY_MEASURED:";
    // Add server time and separate with  ":".
    value = value + server_time.getTime() +":";
    // Add edge preference order and loading times with a separating ":".
    // Edge server loading times are separated with "#".
    for (i=0; i<urls.length; i++) {
        value = value + urls[i]+"#"+url_order[i]+"#"+loading_times[i] +":";
    }
    // Build and set the cookie
    //
    // Set the "NAME=VALUE" pair.
    var the_cookie = "REDIRECTION=" + escape(value);
    // get current date and time
    var cookie_expire_date = new Date();
    // add 30 days
    cookie_expire_date.setTime(cookie_expire_date.getTime()+ 2592000000)
    // add expires field
```

```
    the_cookie = the_cookie + "; expires=";

the_cookie = the_cookie + cookie_expire_date.toGMTString();
// add path and domain fields
    the_cookie = the_cookie + path + domain;
// set cookie property of this document
    document.cookie = the_cookie;
}
```

Figure 7: A JavaScript example to a set a Redirection-Cookie in the client hard disk.

Observe that HTTP specification requires from intermediary proxy servers not to cache messages containing cookies, but to propagate them to the client or server. Thus, exchanging messages containing cookies guarantee that new measurements (rather cached ones) will always reach to the Request-Redirector.

Also notice that using the redirection cookie above bears a great performance advantage since it contains the most recent and particular measurements required for edge-server selection. Thus, a Request-Redirector can respond immediately without further searching the database.

Instant-cookies

As explained in Section XX, web-cookies have two limitations: (i) they can be disabled by the clients; and (ii) they may be returned only after some significant delay. The instant-cookie described below addresses both limitations.

The instant-cookie can be implemented in any client-probe-program, but here we illustrate only a JavaScript implementation. Observe that the information available to a JavaScript program is strictly limited: it can only read images and JavaScript programs from the origin-server, and cookies from the client host. Furthermore, it can only write or modify cookies in the origin-server and the client host. Thus sending information from the client to the activating Request-Redirector can only be made by requesting certain images.

In Figure 5 we demonstrate how a JavaScript program requests a specific image. It can either be done by using the `IMG` tag of HTML, or by assigning the image URI to a JavaScript Image object. An `IMG` tag has a `SRC` attribute and a JavaScript Image object has a "src" property that reflects the `SRC` attribute of the `IMG` tag. Setting the "src" property begins loading the object referenced by the URI into the image. Thus, by encoding the information into the requested URI, that information would be transferred from the client to the Request-Redirector.

A simple encoding is to append the cookie value to a pre-prepared URI string as follows. Suppose that the pre-prepared URI string is `"http://211.172.100.1/notify/"`, where `"211.172.100.1"` is the IP address of the activating Request-Redirector host, and `"/notify/"` indicates measurement data (messages of type (t3) above). Then by setting `"http://211.172.100.1/notify/<value>.gif"` to a "src" property of an Image object in the client-probe-program, `<value>` is sent to the Request-Redirector. Obviously, `<value>` does not reference any image file in the origin-server. The Request-Redirector that intercepts the request just decodes the URI string without fetching the file.

Other encoding techniques (not discussed here) may trade between efficient encoding and efficient decoding. Also, client-probe-programs implemented by an embedded object code (e.g., Java applet, ActiveX object) may open another HTTP connection to the activating Request-

Redirector through which they could send the measurements using HTTP `GET`, `POST` or `PUT` methods (see [HTTP]). The reason for using HTTP rather another protocol is to penetrate firewalls.

Request-Groups

Request-groups are disjoint sets of IP addresses that remain constant over a period of weeks to months. They are built on top of network-vicinity groups that (loosely speaking) are sets of client IP addresses that are close to each other with respect to some network proximity metric.

A convenient way to cluster IP addresses is to mimic the clustering done by BGP (Boundary Gateway Protocol) routers to simplify their routing tables. By their protocol convention, BGP routers exchange routing records that contain the feasible path to each autonomous network. It can be assumed that an autonomous network represents a cluster of IP addresses that are closely located in network hops.

Selection algorithms

All algorithms use either the data from the request cookie, if it is statistically more recent than the data in the Request-Redirector database; otherwise from the database.

The following priority regime is employed. Firstly, all non-operational candidates are excluded. Then, the remaining candidates are ordered by their utility measures and the one with the highest utility is selected, provided its utility is greater than a pre-configured threshold. If no one is selected, the request is forwarded to the origin-server. We use utility thresholds to protect the edge-servers from being overloaded. Also note that only edge servers that support the protocol corresponding to the client's request are considered (e.g. HTTP, HTTPS, MMS, RTSP RealMedia, and RTSP QuickTime media).

It is has been observed in distributed environments that similar priority regimes result in too frequent imbalanced edge-servers. The reason is that under-loaded edge-servers attract many redirected requests from different Request-Redirectors, which in turn overload those servers. Since overloaded servers are relieved slowly, those servers stay overloaded for a long while after which they become under-loaded again.

To prevent this from occurring, an edge-server can be selected with some probability proportional to its utility measure. Alternatively, it can be selected in a weighted round-robin fashion. That is, during each round, edge-servers with higher utility measures are selected more often than those with lower measures. The probabilities or the weights above are tuned based on actual performance measurements that are taken in a deployed CDN. Note that both adjustments occasionally refrain from selecting the best edge-server - a property known to reduce the likelihood of imbalanced edge-servers. Nevertheless, at a longer time scale the two adjustments still preserve the preferences induced by the utility measures.

Measurement update

It is vital to update the utility measures when new measurements arrive at the Request-Redirector. The update procedure is described below. Recall that measurements arrive in cookies and in message requests of Type 3, and have a time stamp that has been set by the Request-Redirector.

If the difference between the current time and the measurement time stamp is less than a pre-specified threshold (i.e., the new measurements have not been aged) then the new loading times are averaged into the database utilities; otherwise they are ignored. Averaging is done using some pre-specified weights that depend on the difference between the measurement time stamp and the last update time of the database value.

If the database is updated, the measurement time stamp becomes the update time of the updated value. Upon completion, the utility measures are recomputed and the candidates are re-ordered.

5. AAA Broker

The definition of a standard interface is required to facilitate the independent deployment of edge-servers. The interface specification should include the following items for the purpose of administration:

- A secure means for edge-servers to authenticate themselves to request-redirectors.
- Standardization of information exchange for accounting and billing purposes.
- An authorization mechanism for new edge servers to offer services.

We collectively refer to these requirements as Authentication, Accounting and Authorization (AAA). In a small, or private CDN deployment, where the relationship between edge-server and content providers is intimate, AAA transactions can occur directly between the two parties. However, larger deployments that involve thousands of edge servers and content providers may justify the establishment of AAA brokers that serve to aggregate accounting and revenue collection as well as providing facilities for secure transactions.

AAA brokers will also provide a catalyst for the deployment of this architecture, as it provides a clear business incentive.

6. Experimental Results

Client side measurements are a key component of our architecture, they facilitate an open architecture by abstracting edge-server measurements from implementation details. In order to verify the validity of our client side measurement technique as a edge-server performance metric, we conducted experiments that involved downloading web pages distributed over the Internet. We then compared the total time required to download the page against our proposed measurement metric and other well known metrics including, round trip time and packet loss rate.

The two dominating factors that determine a page loading time are network latency (or RTT) and server load. The objective of our experiments was to demonstrate that our image loading time measurement is the best predictor for page loading time and therefore the best metric for edge-server selection.

Experimental Setup

Measurements were obtained using a *perl* script that in turn called *curl* [17] to perform DNS name lookup and HTTP transfer, as well as to provide detailed timing of the process. Given a URL, the script would call curl to obtain the HTML content. All of the “img” tags were then extracted by parsing the HTML text. Curl was then successively called to obtain each image object in the page. Our script was limited to only transferring HTML pages and image objects. This was to avoid any noise that may have been introduced by any delays incurred by sending HTTP requests to other servers such as advertisement servers or page hit counters.

Curl provided sufficiently detailed timing information to enable us to differentiate between the time required to resolve the domain name, establish the TCP connection and transfer the object. We deemed the web page download time to be the sum of all of the object transfer times, plus the time required to perform a name lookup for each unique domain name, and the time required to establish a single TCP connection. We felt that this best emulates the transfer time that would be experienced when using a web browser to access a HTTP 1.1 compliant web server.

For the sake of comparison, we collected four metrics: 1) RTT, 2) Packet loss rate, 3) Image transfer time, and 4) Hop count. RTT and packet loss rate were obtained using the *ping* program. Immediately preceding the page transfer, we sent 100 ICMP probe packets, separated by 100ms intervals, to the IP address of the server. The RTT of each probe packet was measured, and the average computed. The proportion of lost packets was deemed to be the loss rate. The number of hops between the client and server was determined using the "*traceroute"* program. The download time of the largest image was also recorded. This time was measured from the successful `GET` request until the last byte of the object has been received, and it does not include the time to resolve the host name or to establish the TCP connection.

The Effect of RTT

Due to TCP's window update algorithm, TCP throughput is adversely affected by RTT. Accordingly, the focus of this experiment was to demonstrate that our image loading time metric sufficiently captures the effect of RTT. Thus, there is no need for explicit RTT measurements.

In order to verify this claim, we ran our script on a list of 75 URLs that pointed to almost identical pages, located at geographically diverse mirror sites. The only differences between these pages were slight variations in the images. The geographic diversity of the sites ensured that the RTT to each individual site was significantly distinct.

As is clearly indicated by Fig. 1 and Fig. 2, both the image loading time and the RTT metric, are highly correlated with the total page loading time. However, packet loss rate (Fig. 3) and hop count (Fig. 4), are not at all correlated with page loading time. Given this evidence we suggest that the image loading time is the best metric for selecting an edge-server for a particular client request, since it also captures the server load (as demonstrated by our next experiment).

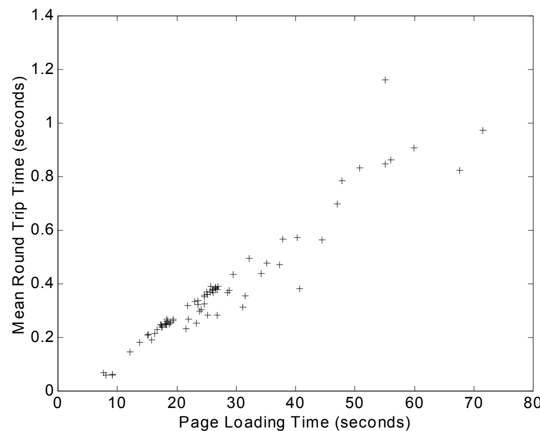


Fig. 1: Mean RTT vs. total image loading time.

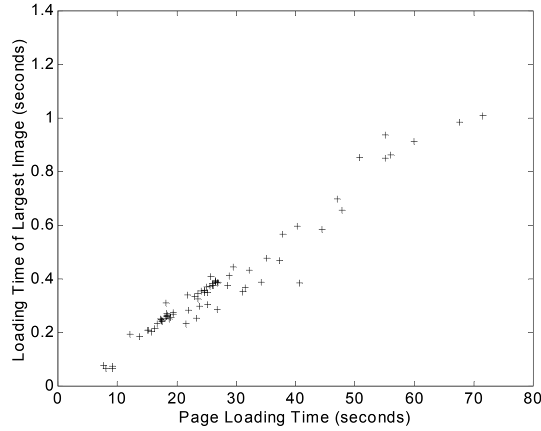


Fig. 2: Loading time of largest image vs. total page loading time.

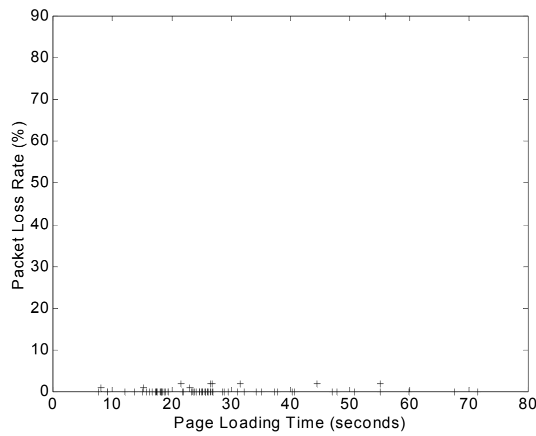


Fig. 3: Packet loss rate vs. total image loading time

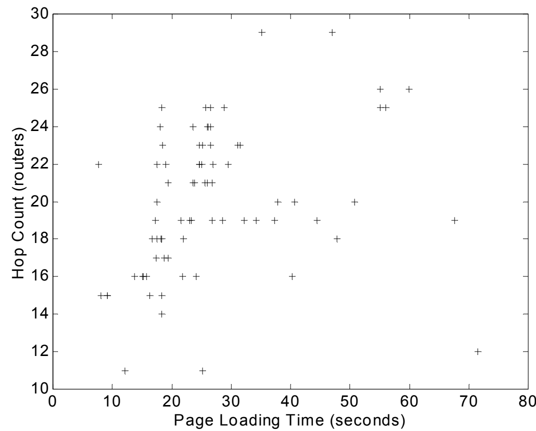


Fig. 4: Hop Count vs. total image loading time

Server Load

The other dominating factor that determines the total loading time of a web page is server load. We have suggested that both server load and RTT are captured in our image loading metric. The focus of this experiment was to verify that our image loading metric provides information on server load.

To verify this proposition, we repeatedly ran our script at 30 second intervals over a period of over 50 hours, on a single web site. Fig. 6 and Fig. 7 compare how the RTT and image loading metric capture the load of the server. As can be seen by comparing Fig. 5 and Fig. 6, the RTT metric has very little correlation with the server load. In fact, the mean RTT is almost constant. Accordingly, we suggest that in this case, the total page loading time is a reasonable reflection of this server's load. The image loading time, Fig. 7, is highly correlated with the page loading time, Fig. 5. This demonstrates that our image loading time metric is capable of capturing server load.

Note the relatively small number of noisy samples in Fig. 7 that correspond to image loading times of approximately 0.23 seconds. We believe that these samples are an artifact of the configuration of this particular server. They would be averaged out by our scheme, since routing decisions are based on many measurements, taken by different clients within a client cluster.

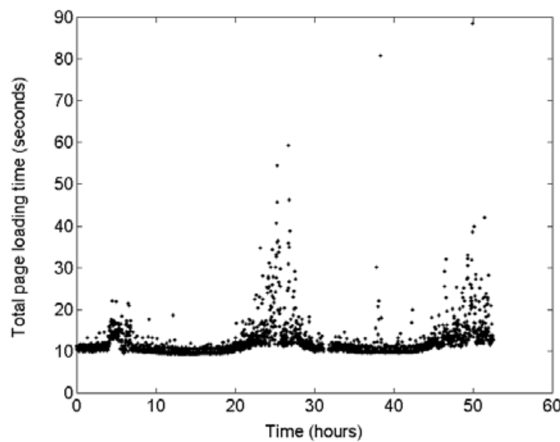


Fig. 5: Total page loading time

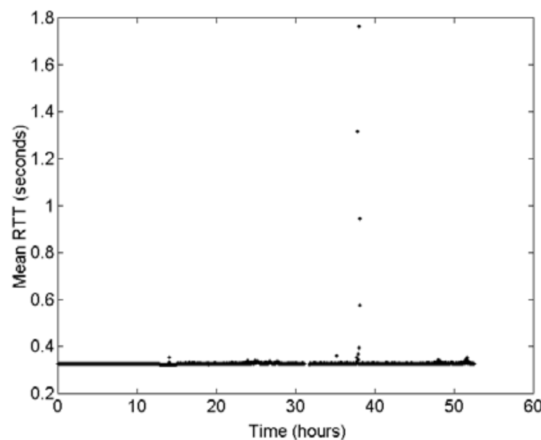


Fig. 6: Mean RTT

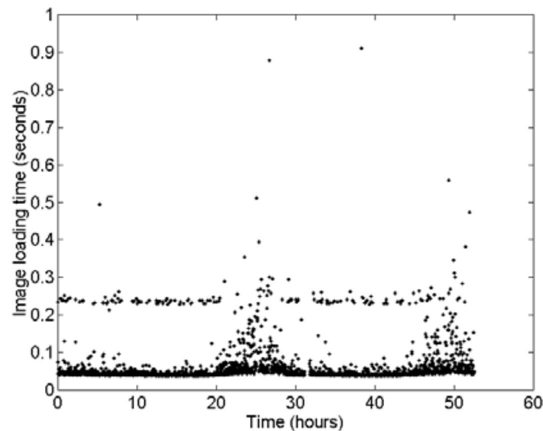


Fig. 7: Loading time of largest image

7. Conclusion

We introduced the major elements that publish, consume, serves, transport and manage content in the web. We further explained the main obstacles faced by content delivery solutions, and presented the existing ones. Then we described the architecture of the Request-Redirector - a new patent-pending component that facilitates an open CDN. The technical aspects of the proposed architecture have been elaborated, specifically the redirection mechanism, the network proximity measurement subsystem, and the algorithms to select best edge-servers. The measurement mechanisms have been demonstrated by a JavaScript program that measures object loading times from multiple edge-server candidates and report them to the activating Request-Redirector. We also explained the advantages of our architecture compared with existing solutions.

References

[AKAM] Leighton, F. Thomson and Lewin, Daniel M., Global hosting system, *US patent* no. 6,108,703, May 19, 1999. Available at http://patft.uspto.gov/netahtml/srchnum.htm.

[ARIN] American Registry for Internet Numbers (ARIN). Available at: http://www.arin.net/.

[CGI] The Common Gateway Interface. Available at: http://www.w3.org/CGI/.

[COO] Netscape Cookie Specification. Available at: http://home.netscape.com/newsref/std/cookie_spec.html.

[DGIS] Farber, David A., Greer, Richard E., Swart, Andrew D. and Balter, James A., Optimized network resource location, *US Patent* no. 6,185,598, February 10, 1998. Available at: http://patft.uspto.gov/netahtml/srchnum.htm

[HTML] The HyperText Markup Language. Available at: http://www.w3.org/MarkUp/.

[HTTP] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and Berners-Lee, T, Hyper Text Transfer Protocol 1.1, RFC 2068, June 1999. Available at: http://www.ietf.org/rfc/rfc2616.

[IRR] Overview of Internet Routing Registry (IRR). Available at: http://www.irr.net/docs/overview.html.

[JS] Flanagan., D., *JavaScript: The Definitive Guide*, O'Reilly & Associates Inc., 4th Edition Nov. 2001.

[KWZ] Krishnamurthy, B., Wills, C. and Zhang, Y., On the Use and Performance of Content Distribution Networks, *ACM SIGCOM Internet Performance Measurement Workshop* 2001. Available at: http://www.icir.org/vern/imw-2001/imw2001-papers/10.pdf.

[RaEl] Rajamony, R. and Elnozahy, M., Measuring Client-Perceived Response Time on the WWW. *USENIX Symposium on Internet Technology and Systems*, San Francisco, California, USA , March 2001.

[REGEX] Regular Expressions, *The Single UNIX Specification, Version 2, The Open Group*, 1997. Available at: http://www.opengroup.org/onlinepubs/7908799/xbd/re.html.

[RIPE] Reseaux IP Europeans (RIPE). *RIPE Network Coordination Centre*. Available at: http://www.ripe.net/.

[ROS] Rosberg, Z., Mechanisms and Methods To Redirect Client Requests for Content Based on Loading Time Measurements Made from Requesting Clients, USPTO 60/347895, 1/15/2002 (Patent-Pending).

[RTSP] Schulzrinne, H., Rao, A., Lanphier, R., The Real Time Streaming Protocol, RFC 2326, April 1998. Available at: http://www.ietf.org/rfc/rfc2326.