A Client Side Measurement Scheme for Request Routing in Virtual Open Content Delivery Networks[1]

Rami Mukhtar, *Student Member, IEEE* and Zvi Rosberg[2]

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

Abstract - A Content Delivery Network (CDN) routes a client's request to an edge server that can best serve the content at the time. Effective request routing is heavily dependent on up-to-date information of server loads and the network condition. In this paper we propose a novel client side measurement scheme that measures both server load, and the network condition, in a single measurement. The scheme is distributed, does not require any modification to the clients, and is compatible with existing client and server software. By experimenting on the existing Internet, we demonstrate that this measurement metric is highly correlated with total page loading time. Combined with *Request-Routing*, our measurement method provides an efficient client request redirection system that is totally transparent to the content provider, and facilitates the deployment of virtual open CDNs.

Keywords: Internet, World Wide Web, Content Delivery Network, Caching Proxy, JavaScript, client-side probes, edge-server selection.

I. INTRODUCTION

The World Wide Web (web) is the most commonly used application to publish and retrieve multimedia content on the Internet. In recent years we have witnessed an evolution in the technologies that are used to deliver content from the content provider's server (*origin-server*) to clients. *Content Delivery Networks* (CDN)s, are beginning to replace traditional technologies, such as site mirroring, caching proxies and intelligent web switches.

A CDN is a network of geographically distributed *edge-servers*, which maintain and serve all or part of the origin-server's content from their locations. Edge-servers are strategically placed close to client aggregation centers; hence shortening their routes to the content. A typical CDN solution consists of network infrastructure that is distributed over OSI layers four through seven. Fundamentally, a CDN attempts to optimize all of the transactions required by layer seven protocols (e.g. HTTP [1] and RTSP [2]), which include the distribution of objects such as text, images, video and audio.

Two core functions of a CDN are 1) Measurement and 2) Request-Routing. Request-Routing is the dynamic routing of a client's request for content based on which edge-server can best serve that content at that time. It is facilitated by up-to-date measurements of the edge-servers' load, and the network condition between edge-servers and clients.

Existing schemes ascertain the current network condition by actively probing the network and/or passively monitoring the performance of active data transfer sessions. This leads to unnecessary network traffic or server loading. Information about each edge server's load is usually obtained by directly querying the edge-servers. Although this method is direct and efficient it has two major drawbacks: 1) it relies on a standardized querying protocol, which is often proprietary, and 2) it is difficult to obtain a metric that is abstracted from the platform on which the server is implemented. These two problems have hindered the development of open CDN architectures, in which third parties can offer edge-server services to potential CDN providers. Furthermore, the majority of existing solutions decouple server load from network condition measurements. Since the performance of layer seven protocols are intimately tied to both, it is far more efficient to combine these two measurements into a single measurement that directly reflects the users perception of performance.

In this paper, we introduce a novel client side measurement scheme. We argue that by initiating measurements from the client, it is possible in a single measurement to determine both the network condition, and the server load, in a distributed manner. In addition, our proposed architecture is completely transparent to the content provider, reducing set up costs. By experimentation on the Internet, we show that this measurement technique provides an effective mean of selecting the best edge server in a CDN. Furthermore, when coupled with a Request-Routing component collocated with the origin-server, it facilitates an open and distributed CDN architecture, where a virtual CDN is created by edge-servers subscribing to provide content distribution services to content providers on individual basis.

[1] This work was supported by the Australian Research Council.

[2] Department of Communication Systems Engineering Ben Gurion University, Beer-Sheva, 84105, Israel, Email: rosberg@bgumail.bgu.ac.il. This research has been conducted while Z. Rosberg was visiting the ARC Special Research Center for Ultra-Broadband Information Networks, University of Melbourne.

II. Related Work

A. Request Routing

Available request-routing techniques can be classified as either DNS Request-Routing, transport-layer Request-Routing, or application-layer Request-Routing.

DNS Request-Routing is based on an enhanced DNS server designated as the authoritative server for the content provider entire or sub domain. On receiving a name resolution request, one or more "*A*" records containing the IP addresses of the edge-servers that can best serve the request are returned. In another variation of this technique multiple enhanced DNS servers cooperate in the name resolution using "*NS*" or "*CNAME*" records to redirect the authority of the next level domain. DNS Request-Routing may also use the Anycast service [3] to simplify the best edge-server selection; and URL encoding to provide the object type or object hash into the DNS name. DNS Request-Routing has several severe limitations as summarized in [4], one of which has been further substantiated in the recent study [5]. It has been shown there that only *16%* of the clients and local DNS are in the same network. Therefore, DNS Request-Routing that redirects a request to an edge server based on the network proximity to the client's local DNS server is very coarse. Our measurement technique alleviates this problem as the measurements are directly between the edge servers and the requesting clients.

Transport-layer Request-Routing techniques intercept user request packets and select the best edge-server based on the first packet of the request. It then hands off the session to the selected edge-server using triangulation, where inbound packets flow through the Request-Routing host, and outbound packets flow directly to the client. The main two limitations of this technique is the packet parsing overhead, and IP spoofing by the edge-server of outbound packets, which will suffer from dropping at ingress filters. Moreover, in HTTP/1.1 with persistent connections, new request messages may arrive in any packet of a TCP session. Thus all packets arriving at port 80 must be inspected to track HTTP requests.

Application-layer Request-Routing techniques also intercept the user requests but examine its information at the HTTP or RTSP protocol layer. The Request-Router component could either be a server plug-in or an in-path reverse proxy. After request examination and edge server selection, the request is routed using a redirection response message (*"3xx"* status code), connection tunneling or packet diversion as in transport Request-Routing techniques. Our approach is to couple the application-layer Request-Routing technique with a novel client side measurement technique.

B. Measurement Techniques

Measurements that are commonly used for edge server selection are: inbound path hop-count (from target to probe), outbound path hop-count (from probe to target), message Round-Trip-Time (RTT) and lost packets. Other techniques examine TCP header fields (sequence numbers, acknowledgement numbers, receiving window size and explicit congestion notification bits) of active connections. By constantly observing these quantities, bandwidth and packet loss can be estimated. Some techniques also look into the TCP stack state variables to get its estimated packet RTT and sender congestion window.

Active probing is done by periodically sending ICMP, UDP or TCP "ping" messages and inspecting the response messages (if returned). For example, ICMP ping sends ICMP ECHO request message of an arbitrary size to a target IP address. When the corresponding ICMP ECHO_REPLY is returned, the Round-Trip Time (RTT) of the ICMP packet is computed, and the Time-To-Live (TTL) header field of the encapsulating IP packet is extracted. Since the initialized TTL values of most Operation Systems are widely known, and each router usually reduces TTL by one, the in-bound hop count can be estimated. The alternative UDP or TCP "ping" methods are useful when the route passes through "ICMP ping shaping" devices or firewalls.

Passive probing examines the behavior of existing TCP connections without explicitly initiating ping messages of any kind. All the statistics are evaluated from the TCP header fields and its stack state variables. Examination of TCP header fields and stack data structures is time consuming and produces statistical estimators that are based on the history of exchanged packets. Clearly, passive probing is feasible only at the edge-servers, in which case, both the processing time and estimation bias need careful consideration.

In current CDN deployments [6], edge-servers are an integral part of the architecture and therefore can effectively support active or passive probing. It is assumed that these probes measure actively or passively only clients that access them, since probing other clients is considered unethical.

Server-side probes have three main limitations. 1) Clients that are assigned to an edge-server cannot be effectively measured by other edge servers. 2) Measurements produced by statistical estimators that are computed from past measurements do not sufficiently track changes in the network congestion and edge server loads. This will often result in a non-optimal assignment of clients to edge servers. For example, it has been shown in [7], that selection algorithms based on statistical estimators are out-performed by others that use run-time (dynamic) estimators. 3) The measurements are conducted in the wrong path orientation. Internet paths are usually not symmetric in neither bandwidth nor propagation delay. For instance, passive techniques that examine the TTL field in incoming packets may not accurately reflect the outbound hop count. Accordingly, we argue that client-side probes are more effective than server-side probes.

In the past, it has been suggested that client side probing may bear a substantial cost of thousands of specialized client-probe devices, and is unpractical in current CDN deployments. This assumption has indeed motivated the study in [8] where passive server-side probing has been used to cluster clients into equivalent sets with respect to edge server selection. It has been proposed in [8] that "*there is no way to remotely ask an*

arbitrary client to measure distance right then, and choose a content server based on its result". In the present paper we challenge this proposition. We present a novel client side measurement scheme that does not require any modification to the client. Furthermore, our measurement scheme is easily integrated into the Request-Routing mechanism. We shall demonstrate how our scheme is capable of transparently turning a client request to the origin-server for content, into a measurement without any noticeable overhead. Furthermore, this measurement provides a run-time estimator for network latency and server load that is highly correlated with the actual page loading time, and can immediately be utilized by the edge server selection algorithm.

Our results are aligned with the findings in [7] where it was empirically demonstrated that runtime estimators perform better than statistical or static estimators as defined in that paper. Static estimators are based upon hardware resources and configuration, such as number of hops, connection bandwidth and server hardware. Statistical estimators are computed from past performance data such as latencies and bandwidths. Dynamic or run-time estimators use probes to detect current network and/or server conditions during, or immediately before accessing the content. Thus, our main contribution with respect to previous work and commercial CDNs is a feasible measurement mechanism easily integrated into a Request-Routing system that combines the advantages of client-side probing, dynamic estimation, fast adaptation and negligible cost. We show that the measurements we use are best correlated with actual page loading times. The techniques presented in this paper are based on a patent pending CDN design [11].

In Section III we describe the Measurements subsystem and in Section IV we explain how it operates from the Request-Routing module. In Section V we present our experimental results that demonstrate that our client-side object loading time estimators are correlated with the actual page loading time better than other estimators.

III. The Measurement Subsystem

A major problem in any CDN deployment is to collect and maintain measurements that closely reflect which edge server can best serve each client request. Existing CDN solutions include special monitoring software or devices at the edge-servers and throughout the Internet, digesting their measurements in a central database. We present an implementation where clients arriving at each Request-Routing host are silently instructed to measure the loading times from several edge-server candidates and report the measurements back to their activating Request-Routing host. Each Request-Routing host then maintains its own local database for its clients and their corresponding edge-server candidates. Our proposed measurement scheme comprises a probe-program and two reporting methods.

A. The Probe-Program

The measurement device is a probe-program written in a scripting language interpretable by the client (e.g., JavaScript) or implemented in platform independent object code (e.g., Java), which can be embedded into a response message. Once received by the client, and executed by the browser, the probe program instructs the client to download selected objects from a select number of candidate edge-servers. The object loading times are measured and recorded by the client. In the preferred scenario the probe program attempts to store the measurement data onto the client hard drive and communicate it back to the request-redirecting host using a standard web-cookie. In addition or alternatively, the program sends the measurement data back to its activating request-routing host using the "*instant-cookie*" method described below.

Note that each client may have a different set of edge-server candidates (that may change in time), thus candidates must be inserted into the probe-program on-the-fly. To synchronize measurement times from different time zones and different client clocks, the Request-Routing host sets its local time in every probe-program to time stamp the corresponding measurements.

Fig. 1 illustrates a probe-program implemented in JavaScript, [12], that uses a measurement technique devised in [13]. The program measures image loading times from two different URI locations (defined by 1.5, 1.6 in the figure) when the user passes the mouse over the link defined by 1.4 in the figure. On mousing the link, the start time is set to `var start` (defined by 1.1 in the figure) and when the image loading completes, the event handler (defined by 1.3 in the figure) calls the function `noteTime()` (defined by 1.2 in the figure) that computes the loading time.

Measurements made by the probe-programs are reported to their active Request-Routing hosts via two methods: (i) standard web-cookies and (ii) "*instant-cookies*" - a new mechanism that avoids the limitations of cookies.

B. Measurement Reporting Methods

Measurement data must be returned by the client sufficiently fast in order to be useful for dynamic selection of the best edge server for any subsequent object request. The web-cookie, [14], that is a piece of information exchanged between a browser and a web server, is an almost ideal mechanism to report such measurements. A cookie contains a directory path to the origin-server telling the browser where to send it, and information written in a series of "NAME=VALUE" pairs. However, 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 of time after which they expire.

The request-routing host uses cookies (in the headers and in embedded JavaScript programs) as a mechanism by which the client returns the following redirection data:

- Edge-server candidates for the particular request.
- Measured loading times from every candidate to the requesting client.
- Measurements time stamp using the Request-Routing host clock.

- Cookie type distinguishing between local measurement values and estimated values.

Fig. 2 illustrates a JavaScript sample code (along with explanatory comments) that silently saves a standard web-cookie in the client browser hard drive.

```
<HTML> <HEAD>
<SCRIPT language=JavaScript>
<!--
// 1.1
var start;
var stop;
// 1.2
function noteTime() {
stop = new Date()
diff = stop.getTime() - start.getTime();
alert(diff+' milliseconds');
}
// -->
</SCRIPT> </HEAD>
<BODY>
<P><B>Clear memory & disk cache and mouse over
the links below to measure its loading
time.</B><HR> <P> <CENTER> <!— 1.3 -->
<IMG OnLoad='noteTime();' src=""
alt="loading_image" name="image_1" width="150"
height="100"><BR> </CENTER>
<HR>
<!— 1.4 -->
<A onmouseover="start = new Date();
<!— 1.5-->
image_1.src='http://edge.domain1.com/test.gif';"
href="'http://edge1.domain1.com/test.gif'">Edge
Server 1</A><BR>
<HR>
<A onmouseover="start = new Date();
<!— 1.6 -->
image_2.src='http://edge.domain2.com/test.gif';"
href="'http://edge.domain2.com/test.gif'"> Edge
Server 2</A><BR>
<HR>
</BODY>
</HTML>
```

Fig. 1: A JavaScript Probe-Program code example

Using web-cookies as measurement couriers has two benefits. The current HTTP specification requires that intermediary proxy servers propagate them to the client or server. Thus, exchanging messages containing cookies guarantee that new measurements (rather cached ones) are always delivered to the Request-Routing host. Furthermore, the redirection cookies as described above, contain the most recent and most relevant measurements required for edge-server selection. Thus, a Request-Routing host responds immediately and accurately without further searching a measurement database.

However, standard web-cookies have two limitations: (i) they can be disabled by the clients; (ii) they may be returned after some delay. To address these two limitations we propose the following "*instant-cookie*" method that is implemented as part of the JavaScript probe-program.

Due to the limitations of JavaScript, the only available technique for sending information to the server is to embed the information in requests for image objects. For illustration, consider the pre-prepared URI string: `"http://n.n.n.n/notify/"`, where `"n.n.n.n"` is the IP address of the activating Request-Routing host, and `"/notify/"` indicates measurement data. Then by setting `"http://n.n.n.1/notify/<value>.gif"` to a "src" property of an Image object in the client-probe-program, `<value>` is sent to the Request-Routing host. Obviously, `<value>` does not reference any image file in the origin-server. The Request-Routing host that intercepts the request just decodes the URI string without fetching the file.

```
// Server time set by the Request-Routing
// Date(yr, mo, day, hr, min, sec)
var server_time = new Date(01,12,09,13,27,45);
// Edge-server candidates set by Request-Routing
var urls = new
Array("http://cache.domain1.com/test.gif",
"http://cache.domain2.com/test.gif",
"http://cach.domain3.com/test.gif");
//Edge-server preference order init with zeros.
//Set by the Client-Probe-Program timing
function.
//E.g., ("3","1","2") indicates that edge2 is
best //edge3 is second best and edge1 is the
worst.
var url_order = new Array("0","0","0");
//Loading times (ms). Init 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-Routing.
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 with "NAME=VALUE"
pairs
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.ge
tTime()+ 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; }
```

Fig. 2: A JavaScript to set a cookie in the client hard disk.

Other encoding techniques (not discussed here) may provide trade offs 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-Routing host through which they could send the measurements using HTTP `GET`, `POST` or `PUT` methods (see [1]). The reason

for using HTTP rather another protocol is to penetrate firewalls.

Our measurement scheme also takes into consideration other aspects that may affect the performance of a CDN that other techniques may ignore. One is the overhead active probes impose on the edge servers. If each client probes every edge server, the server would quickly become overwhelmed with too many messages. We avoid this by clustering clients into equivalent classes based on BGP/IGP routing table information obtained from querying BGP/IGP routers. This clustering method is similar to the approach taken by Cisco's Distributed Director, [9], and F5 Network 3-DNS, [10]. Using this clustering scheme and the hop-count between BGP/IGP routers, we select the best edge server candidates to serve each client cluster. Only a small number of candidates in each cluster are randomly selected for probing. We suggest that the clustering configuration is updated on monthly intervals.

C. Request Clustering

Any practical CDN solution should also address scenarios where a client request does not contain a redirection web-cookie. This can be a common occurrence since there are potentially a large number of possible clients. To address this, we use the concept of "*request-clusters*". Request-clusters are a refinement of client-clusters that are loosely defined as sets of client IP addresses that are close to each other with respect to some network proximity metric. For manageability reasons it would be wise to consider a metric that remains stable over a long time period (e.g., several calendar weeks or months). We also require that client clusters be disjoint.

A simple client-cluster is a class C of IP addresses (addresses that differ by their last 8 bits) since they are most likely allocated to the same local or metropolitan sub-network. However, there are 2^{24} C classes, which is still too many. Clients that access the Internet through small or medium size ISPs are connected to routers that are geographically (and path-wise) close to each other, and are linked by high bandwidth backbone channels. The CIDR (Classless Inter Domain Routing) blocks allocated to each of those ISPs form a sensible client-cluster. We propose that the CIDR blocks allocated to large and medium size ISPs, should be split into several client-clusters based on geographical proximity. This is especially important to deal with ISPs that re-allocate part of their CIDR blocks to other organizations. (It is not rare to find CIDR blocks that have initially been allocated to UUNET and MIT, and are currently being used by organizations located in South America or Asia).

"Whois" servers such as [15] maintain the allocated CIDR blocks of each registered ISP and Autonomous Systems (AS) but their databases are not kept up-to-date. More reliable sources are the BGP (Boundary Gateway Protocol) routers. (A BGP router is a router used to route the TCP packets from one AS network to another.) By their protocol convention, BGP routers exchange routing records that contain the feasible path to each CIDR block addresses. Since each hop of that path consists of the gateway address and the AS identification, the CIDR blocks of the last AS hop can be extracted from these records. Thus, by adding a special code to a BGP router, one can collect that data. There are also some web sites that make updated BGP routing tables readily available.

In the absence of any further AS administrative information, the set of CIDR blocks that are allocated to any given AS defines a single client-cluster. If more information is available from network administrators, then each AS can further be segregated based on geographical locations. Client-clusters are based only on client IP addresses and network proximity. However, edge-sever assignment to a particular client cluster must also depend on what protocol it supports. Hence, each client cluster must is split into several "request-clusters", one per each access protocol.

IV. Request-Routing

To successfully implement the scheme described in the previous section, the Request-Routing subsystem must be collocated with origin-servers. Each client request must be intercepted and an appropriate action taken. In particular we require that the origin-server has the ability to embed the JavaScript probe-program in selected response messages.

For that purpose we propose a Request-Routing software module that is collocated with the origin-server. Additional to request routing, the module embeds probe-programs and maintains local measurement records for each request-cluster. The module can be implemented in several ways, (e.g. software plug-in or server extension module) depending on the origin-server configuration. It is also possible to run the Request-Routing module as a separate reverse proxy process (in the same or another machine) that intercepts all client requests.

A. The Request-Routing Module

The Request-Routing module intercepts client requests arriving at the origin-server and decides how to process them based on their requested content, source IP address, and other information they may carry. Selected requests are redirected to one out of multiple edge-server candidates distributed across the Internet. To implement efficient and optimal redirections at any time, the Request-Routing module manages a local database detailing network proximities between request-cluster and edge-server candidates.

The processing steps are schematically depicted in Figure 3. 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:

- Type 1: a request from an edge-server;
- Type 2: a request for content that should be appended with a probe-program;
- Type 3: a measurement notification request;
- Type 4: a request for content that should be redirected to one of the edge-servers;
- Type 5: other requests.

Requests of type 1-3 are used for CDN management and

data collection, whereas requests of types 4 and 5 are ordinary ones.

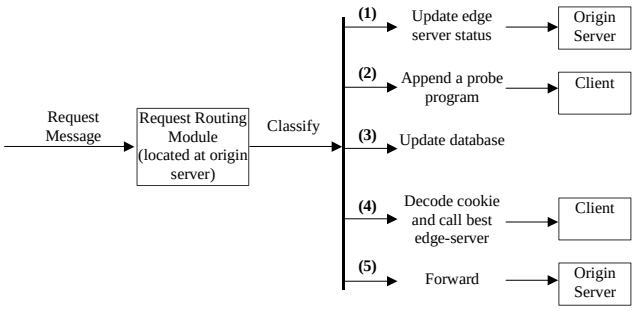

Fig. 3: Request-Routing processing

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 ("3xx" 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. Fig. 4 illustrates the transactions required for a typical client request.

B. Virtual CDNs

The Request-Routing module encapsulates all the functionalities of a CDN except for the edge-servers. However, ISPs, Access and Enterprise Networks have client side caching proxies located at Points of Presence (POPs) and other client aggregation points. These locations are as good as a content provider may hope for. The problem with those caching proxies is that their locations, capacities and content consistency are controlled by the ISPs. Some content providers cannot afford to depend on a completely independent third party to administer the delivery of their content. By enhancing the caching proxies to operate as edge-servers, as shown in [11], and exercising a service level agreement between the ISPs and the content publishers, Virtual CDNs can easily be setup.

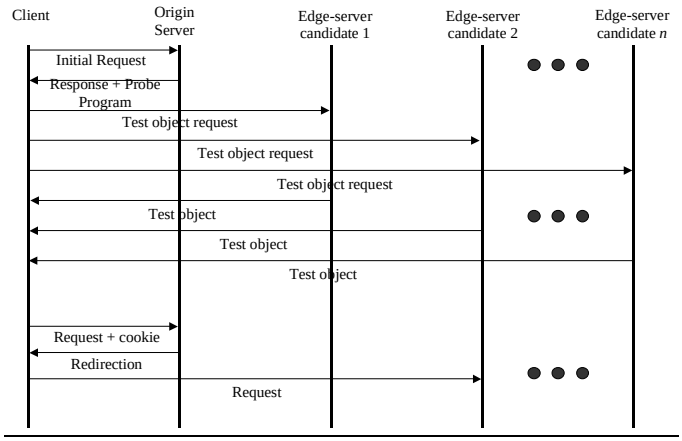

Fig. 4: Typical client transaction

A Virtual CDN is implicitly created by ISPs or other edge service provider subscribing to provide content distribution services to a content publisher on an individual basis. To facilitate very large scale Virtual CDNs, an independent third party may be needed to provide standard transactions between origin and edge servers, and to enforce security and compensation agreements. We refer to such an organization as an Authentication, Accounting and Authorization (AAA) broker, which could serve as a trigger for Virtual CDNs. An AAA broker would be economically justified by charging a commission for each transaction completed, making the business a lucrative one.

V. Experimental Results

To verify our client side measurement technique, we conducted experiments that involved downloading web pages from pre-existing servers 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 the page loading time are network latency (or RTT) and server load. The objective of our experiment is to demonstrate that the image loading time is the best predictor for page loading time and therefore the best metric for edge-server selection.

A. Experimental Setup

Measurements were obtained using a *perl* script that in turn called *curl* [16] to perform DNS name lookup and HTTP transfer, as well as to provide detailed timing of the process. Given a URL, the script calls *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 measurement noise due to delays incurred from sending HTTP requests to other servers such as advertisement servers or page hit counters.

Curl provides detailed timing information to differentiate between the times for DNS name lookup, establish the TCP connection and transfer each object. To emulate the HTTP 1.1 protocol 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.

For 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 as follows. 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 measured from the successful `GET` request until the last byte of the object has been received. It does not include the time to resolve the host name or to establish the TCP connection.

B. 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 that is impractical to perform from the client browser.

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 [18]. The geographic diversity of the sites ensured that the RTT to each individual site was significantly distinct. As is clearly indicated by the monotonic pattern in Fig. 5 and Fig. 6, both the image loading time and the RTT metric, are highly correlated with the total page loading time. However, packet loss rate (Fig. 7) and hop count (Fig. 8), are not at all correlated with page loading time. This evidence suggests that the image loading time is the best metric to determine an edge-server for a particular client request, since it also captures the server load (as demonstrated by our next experiment).

C. Server Load

Server load is the other dominating factor that determines the web page loading time. The focus of this experiment was to verify that our image loading metric provides information on server load. This was done by repeatedly running our script at 30-second intervals over a period of over 50 hours, on a single web site serving the FIFA world cup soccer results [17], at a period that coincided with the Korea-Japan FIFA world cup tournament. Fig. 10 and Fig. 11 compare how the mean RTT and image loading metric capture the load of the server. As can be seen by comparing Fig. 9 and Fig. 10, the mean RTT metric has very little correlation with the server load over the 50 hours period. In fact, the mean RTT is almost constant. Fig. 9 and Fig. 10 show that image loading times nicely follows the peaks of the total page loading time along the 50 hours period. Since those peaks could be explained only by server overload, we suggest that the image loading time is a good reflection of this server's load. This demonstrates that our image loading time metric is capable of capturing server load.

Note the relatively small number of noisy samples in Fig. 11 that correspond to image loading times of approximately 0.23 seconds. We think 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.

VI. Conclusion

We have presented a novel client side measurement scheme that facilitates the selection of the best edge server to serve content to a particular client. Server load and RTT both have a significant impact on the performance of an edge server. We have shown that our scheme is effective in measuring both with a single measurement. Based on a JavaScript implementation, our scheme does not require any modification to clients, and is compatible with existing server software. The scheme is distributed, and is well suited to an open CDN architecture deployment.

References

[1] 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.

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

[3] Partridge, C., Mendez, T. and Milliken W., "Host Anycasting Service", RFC 1546, November 1993. Available at: http://www.ietf.org/rfc/rfc1546.

[4] Barbir, A., Cain, B., Douglis, F., Green, M., Hofmann, M., Nair, R., Potter, D. and Spatscheck, O., Known CN Request-Routing Mechanisms, Internet-Draft Nov. 8, 2002. Available at: http://search.ietf.org/internet-drafts/draft-ietf-cdi-known-request-routing-01.txt.

[5] Mao, Z. M., Cranor, C. D., Douglis, F., Rabinovich, M, Spatscheck, O. and Wang, J., A Precise and Efficient Evaluation of the Proximity between Web Clients and their Local DNS Servers, *Proceeding of USENIX, June 2002, Monterey, CA.*

[6] 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.

[7] Dykes, S. G., Robbins, K. A. and Clinton Jeffery, C. L., "An Empirical Evaluation of Client-side Server Selection Algorithms," *Proceedings of INFOCOM'00*, March 2000, pp. 1361-1370.

[8] Andrews, M., Shepherd, B., Srinivasan, A., Winkler, P. and Zane F., Clustering and Server Selection using Passive Monitoring. *Proceedings of INFOCOM'02*, 2000.

[9] Cisco Distributed Director. Available at: http://www.cisco.com/univercd/cc/td/doc/product/iaabu/distrdir/index.htm.

[10] F5 Networks 3-DNS. Available at: http://www.f5.com/univercd/3dns.

[11] 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).
[12] Flanagan., D., *JavaScript: The Definitive Guide*, O'Reilly & Asso. Inc., 4th Edition Nov. 2001.
[13] Rajamony, R. and Elnozahy, M., "Measuring Client-Perceived Response Time on the WWW," *USENIX Symposium on Internet Technology and Systems*, San Francisco, Cal., USA , March 2001.
[14] Netscape Cookie Specification. Available at: http://home.netscape.com/newsref/std/cookie_spec.html.
[15] American Registry for Internet Numbers (ARIN). Available at: http://www.arin.net/.
[16] cURL. Available at: http://curl.haxx.se/.
[17] http://fifaworldcup.yahoo.com/es
[18] http://www.tucows.com/

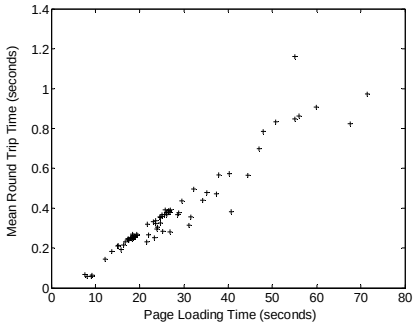

Fig. 5: Mean RTT vs. page loading time.

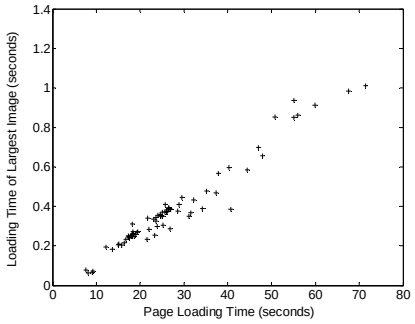

Fig. 6: Image loading time vs. page loading time.

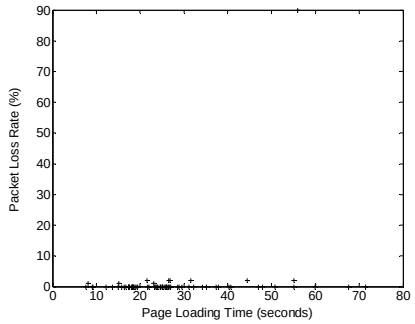

Fig. 7: Packet loss rate vs. page loading time

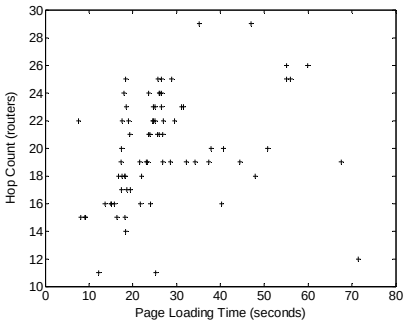

Fig. 8: Hop Count vs. page loading time

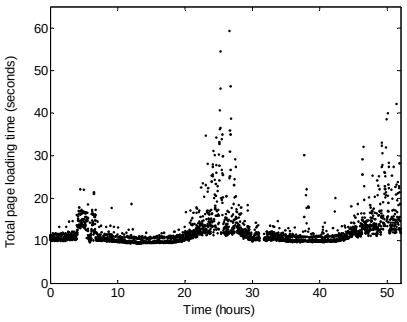

Fig. 9: Total page loading time

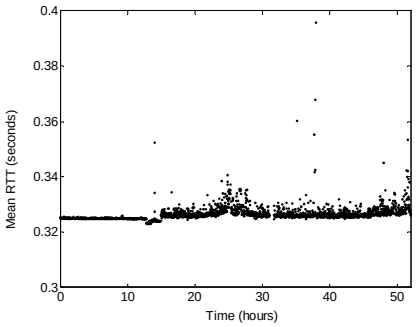

Fig. 10: Mean RTT

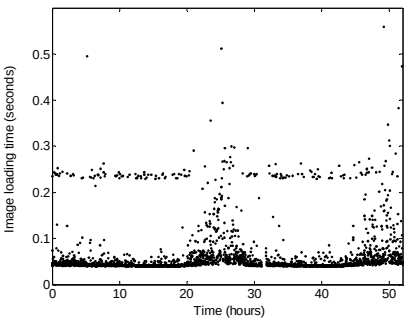

Fig. 11: Loading time of largest image