A Closer Look at a Content Delivery Network Implementation

B. Molina, V. Ruiz, I. Alonso C. E. Palau, J.C. Guerri, M. Esteve
Communication Department, Polytechnic University of Valencia (UPV)
C/ Camino de Vera s/n 46022 Valencia, Spain
Phone: +34 963877301 Fax: +34 963877309

Abstract -- **Success in Internet applications involves user interactions whose quality is mainly affected by application response time. Content Delivey Networks (CDNs) have shortly appeared as a distributed solution to serve content faster than contacting a centralized server. Their effectiveness has been showed by larger companies such as Akamai and Speedera. However, there is currently a certain gap about implementations issues of this technology, and only arquitectural designs and performance reports are published. This article tries to describe a CDN from a different point of view, paying much attention on the implementation process of a CDN.**

I. INTRODUCTION

With the explosive growth of the World Wide Web, popular web sites receive an enormous share of Internet traffic. These sites have a competitive motivation to offer better service to their clients at lower cost. For this reason, there has been currently an increasing trend towards placing content (globally or partially) in content delivery networks or using P2P schemas.

A Content Delivery Network (CDN) is an overlay network on top of the Internet which pushes content closer to end users. It is achieved by strategically placing servers, called surrogates, next to these users and serving them the desired content. The surrogates act typically as intelligent and transparent proxy caches that retrieve content previously from the origin server before responding. As the origin server is less accesed, backbone traffic is reduced and network bandwith is efficiently used. Besides, load can be balanced among the servers.

Existing work on CDNs has primarily focused on techniques for efficiently redirecting user requests to appropriate surrogates to reduce request latency and balance load, and placement strategies to place server replicas in order to achieve better performance. Little attention has been paid to implementation issues of a CDN, although many CDN service providers like Akamai [1] offer some 'overview' whitepapers, they hide the real implementation as a private secret and fundamental key of their business success. This paper tries to provide some implementation hints to establish a CDN basis.

The rest of the paper is structured as follows. Section 2 introduces the motivation and previous work. In section 3 we present our CDN model architecture with the description of the main building blocks from an implementional point of view. The paper finishes with the conclusions and future work.

II. MOTIVATION AND RELATED WORK

A CDN is a global scale-out approach attempting to reduce netwok latency by avoidance of congestion paths. Leading CDN companies have placed from hundreds up to thousands of servers throughout the world, being able to serve content from a nearby surrogate. How to correctly deploy and manage such huge content networks is a case study in this article.

Previous research has focussed on the performance of CDNs, which is largely determined by its ability to direct client requests to the most appropriate server [2]. Some studies address the DNS effectiveness, paying attention on the incurred overhead in the process [3]. There are other studies that evaluate the accuracy of the server selection algorithm at choosing the optimal server [4]. Recently, some analytical models have been proposed to test the behaviour of a CDN and its performance [5].

However, very little has been done at implementing a real (and free for study) CDN. PRISM testbed architecture of AT&T Labs describes basic functionality of a streaming oriented CDN [6]. Globule is probably the most current reference for an open CDN nowadays, and includes various interesting and descriptive published articles [7]. Our approach is less specific in a certain sense, as it tries to put some clearness before coming into details. We firmly think that this design step between description and implementation, though often not considered, is really important.

III. CONTENT DISTRIBUTION ARCHITECTURE

A. General architecture

G. Peng [8] proposes the architecture shown in Fig. 1, which comprises six basic elements. The relationships between blocks are as follows: the origin server delegates its URI namespace to the request routing system (1), and publishes content (2) to be distributed to the remote surrogates (3) by the distribution system. A client requests a content from what he perceives to be the origin server, but his request is treated by the request routing system (4) which redirects him to the optimum surrogate server (5). The surrogate servers periodically send

0-7803-8271-4/04/$20.00 ©2004 IEEE

information to the accounting system (6), which summarizes it in detail statistics and sends it as feedback to the origin server (7) and the request routing system (8).

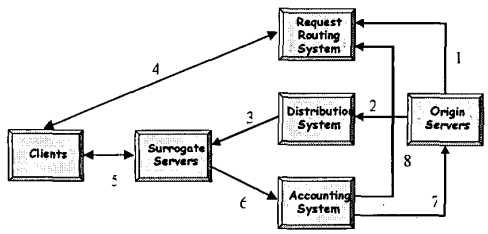

Figure 1. General architecture of a CDN

B. Integrating the tasks inside real processes

The above architecture suffers from its abstract overview at implementation time. One has to convert the blocks into real objects to be programmed. Furthermore, it is essential to define all the tasks on each functional module, as well as their relationship and temporalization. Let's start from the following sequence of actions that takes place in a content transaction.

- The client will connect to a portal, e.g. *www.portal.com*, through a web browser. A portal consists of a set of surrogates that build together a CDN.
- The request is processed by the authoritative DNS server, which is responsible to map the name *www.portal.com* into at least one IP address. This is the best point to introduce the Request Routing System, and is mostly used by current CDN companies. In fact, the DNS server is nothing but an interface: another process, call it *Redirector*, is the one in charge for determining the optimal surrogate.
- The process Redirector is mainly composed by an algorithm that accepts input parameters and produces a response, tipically a list of IP addresses. The choice of an appropiate server depends on client proximity, server overhead and network congestion.
- Server overhead and network congestion implies some type of continuous monitorization, for example, through SNMP. This is addressed by another process, say *SNMP Monitor*, responsible for capturing periodical information of the servers and the network.
- The client will retrieve a list of IP addresses decrementally ordered by optimal performance estimated by the *Redirector* process. Once the client enters the portal from one of the surrogates, it has to select a content. This content is typically in a multimedia format and is delivered streamlined by a media server. So we need both a web server and a media server.
- Once the desired content is selected by a user, a new resolution phase is needed, as this selected content supposes a new input parameter. It is also important to note that target web surrogates could be different from target media surrogates. The resolution phase takes place at HTTP level, acting the first contacted surrogate as interface.
- In order to distribute the content in a streamlined multimedia format, some kind of plug-in is required inside the browser, such as RealPlayer, QuickTime or, in an open way, a simple Java applet. This plug-in connects to the media server in order to retrieve the content.

C. A closer look at the components

Once the processes have been basically described, it is time to study feasible ways of implementation. Though there are possibly many interesting tools in the business market, the open approach of the CDN suggests the search for open protocols and solutions.

Following the previous order and obviating the client browser, let's start with the *DNS server*. The function of our DNS is to simply map CDN name servers into CDN identifiers. Once a client request for a certain website arrives at the *DNS server*, it filters it depending on the content: if the site is associated to a certain CDN, then the *DNS server* obtains the corresponding CDN identifier and resends the request to the *Redirector* module. Otherwise, the request can be forwarded to a local DNS server following the hierarchical DNS operation.

The *Redirector* is a key process of the whole system, as is the one in charge of deciding an adequate surrogate for each client request. There are two different functional modes, though similar, related with the number of input parameters that the included algorithm supports. In the first mode, which takes place at DNS resolution phase, the *Redirector* module retrieves the CDN identifier and a client IP address. The latter parameter (IP address) is at this stage unnecessary if only scalability is targeted. The second mode takes place after the client has selected the content. This time the surrogate that is serving him has to interact in background with the *Redirector* module to retrieve an optimal surrogate for serving this content, which is a key parameter in the selection strategy.

Now it is also important to serve content from a nearby surrogate in order to obtain a low response time; therefore, client proximity is estimated and taken into account. If the CDN environment remains local (iCDN) and the number of surrogates is not considerable, a simple way of calculating proximity consists of sending pings from each surrogate to the client. The algorithm used to balance load among the surrogates is based on [9] but takes as input parameter a linear combination of available server resources, such as CPU utilization, memory usage and number of established connections. After that, an assignation probability weigth is targeted to all surrogates of the portal. If all servers are equally loaded within a time interval, each of them will serve a client request with the same probability ($1/N_s$). On the contrary, there is also an admission control routine: if a surrogate is overloaded above an established limit, it will not be considered in the algorithm.

The *SNMP Monitor* captures status information from the surrogates. This information is of two types: on the one hand, the monitor stores data about available resources in each portal or surrogate (memory, CPU utilization and number of connections); on the other hand, the monitor tracks information about network status between clients and portals. Whereas the first type of data is periodically readed, the second type is asynchronously requested from the *Redirector* module each time a client issues a request to the CDN.

The surrogates or portals act as CDN entry points for the clients and are in charge of serving them the desired content. The portals store static content (web pages) and generate dynamic content. Once a portal receives a client request for a streaming media content, it firstly interacts with the *Redirector* module to obtain an optimum surrogate IP address. After that, the portal generates an applet that contains a media player and sends it to the client, including the IP address of the optimum server. The client then initiates the applet and reproduces the multimedia content.

The CDN Manager is responsible for initializing all CDN parameter values, as well as managing how and where to store content according to a certain policy. That includes cache time control, content transfers between portals, content inclusion, content deletion, etc.

D. Database design

Any system that stores and bases its behaviour on stored data (at least partially) must include an effective design of its database structure. The database design is highly dependent of the desired content to be published. For this article, we will assume that content is oriented to e-learning applications. For other applications, design may vary significantly. In the case of our CDN, there are various important databases associated to the different modules of the architecture. There is a global content database that includes three data tables:

- *table_lessons*: it includes some important information for reference (the title of the lesson, the correspondent subject, faculty and teacher).
- *lessons_CDN*: it associates a lesson with a portal.
- *copies_lessons*: it indicates which surrogate has an available copy of a certain lesson.

The first two data tables of the content database are remotely replicated on each surrogate, so that each surrogate has local knowledge of the available content in the CDN. The *SNMP Monitor* has its own database to store all the information obtained by the SNMP agents etiher periodically –CPU usage, used memory and connections – or asynchronously – pings and network hops. Note that ping mechanisms may suppose a problem if a client incorporates a firewall that rejects ICMP messages The redirection algorithm, as part of the *Redirector* Module, also has its own database to store values of server load and server proximity.

E. Data exchange between the components

A well performance of a CDN significantly depends on the correct communication of each process of the system. This communication takes place in form of messages, whose exchange is illustrated in Fig. 2.

Two different routes can be distinguished: a DNS resolution that redirects a client to a portal using a load balancing algorithm (4 steps), and a portal resolution phase when the client has already entered a portal and is going to select a streamlined multimedia content from a list of available ones (7 steps). If no server is available, an empty list is sent and an error message is forwarded to the client. Besides these messages that occur in a content transaction, there are additional ones related to management tasks, such as content transfers, cache control, etc.

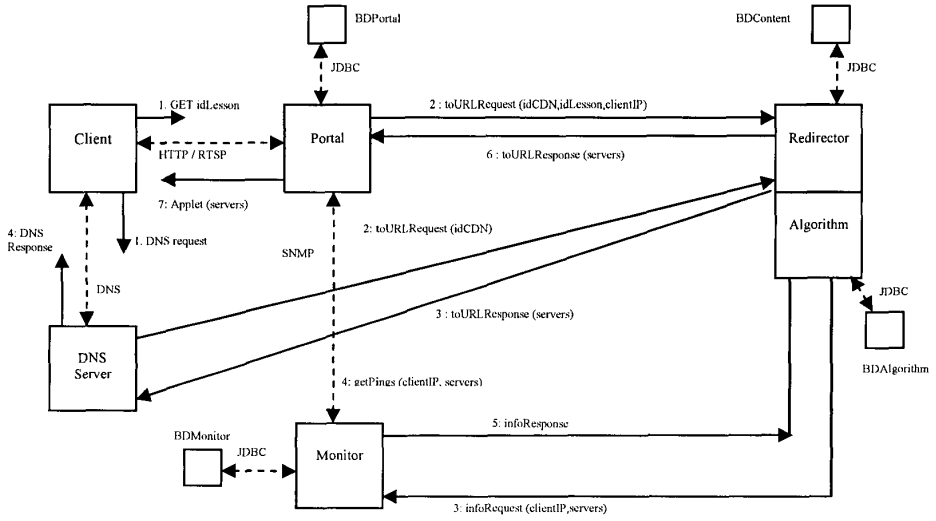

Figure 2. Data exchange among architectural modules

E. Some intermediate results

Though our CDN is still in deployment stage, some theoretical results as well as practical ones can be presented in this article. Table I shows mean values for two types of operations where a small web page is requested. The first operation implies a request without going through our CDN architecture, whereas the second one traverses our *Redirector* module. Note that the latter is significantly slower, as the 'redirector' delay supposes about 80% of the total delay experienced by a user. However, this is due to the extra time used in sending HTTP-SOAP messages (similar as in web services), not in the algorithm itself.

Table II shows a simulation result that tests the balancing capability of the redirection algorithm. Five initially equal-loaded surrogates representing a CDN have to deal with a set of arriving client requests (from 50 up to 500). The table shows for two surrogates the mean number of connections that they have received as well as the mean standard deviation. As can be seen, the *Redirector* module balances between servers. If one surrogate would be initially more loaded, it would have received less connections compared to the others (supposing that all of them have the same power).

TABLE I.
MEAN TIME MEASUREMENTS FOR A CONTENT TRANSACTION WITH AND WITHOUT CDN

Transference	19,67524
DNS	13,0641
Redirector	136,6631
DNS+Redirector	149,7272
Total without CDN	32,7394
Total with CDN	169,4025

TABLE II.
MEAN NUMBER OF CONNECTIONS RECEIVED BY TWO SURROGATES AND THEIR STANDARD DEVIATION.

Connec.	S_1	Std(S1)	S_2	Std(S_2)
50	9,8	3,59	10,08	4,31
100	19,72	5,54	20,08	7,15
150	30,28	5,77	29,92	7,86
200	39,92	10,25	40,4	7,18
250	50,4	9,67	49,8	8,03
300	60,16	9,22	59,96	10,84
350	70,12	8,40	69,48	12,62
400	79,56	12,1	81,16	9,23
450	89,96	9,72	90,12	13,58
500	99,88	11,2	100,28	16,67

IV. CONCLUSIONS

The excitement that surrounds modern technologies sometimes overshadows associated implementation and management issues. Current Content Distribution Networks are one of those technologies that commonly remain in private property for their business potential. Therefore, some stages of their design and implementation phases stay unknown. This article has attempted to put some clarity in a practical way to build a CDN, identifying the main building blocks and the means they interact exchanging messages..

Appart from clients, surrogates that serve content as well as a CDN Manager are required. The surrogates behave as web or media servers, and must include logging activity to be managed by the acounting system of the CDN. The CDN Manager owns a redirector module which uses an algorithm to choose an optimum surrogate for each client. This algorithm suposses a trade-off among server proximity, network congestion and content availability. Besides redirection capacity, the CDN Manager is also in charge of managing the whole system. Most of the monitorization is addresed by an independent SNMP process that periodically scanns portal parameters. Network distances between clients and surrogates are measured in terms of latency (ping) and network hops (TTL) within an adjustable balanced equation. As our working environment corresponds to a campus intranet, we have control of clients and network equipment so that firewalling does not become a drawback. In a real scenario with filtering activity, however, further considerations have to be considered, such as BGP routing information mostly used by current CDNs. This is an open issue that will be treated in future work.

V. REFERENCES

[1] Akamai Technologies, http://www.akamai.com

[2] J. Kangasharju, K. W. Ross and J.W Roberts. Performance Evaluation of Redirection Schemes in Content Distribution Networks. 5th International Workshop on Web Caching and Content Distribution, Lisbon (Portugal), June 2000.

[3] A. Shaikh, R. Tewari and M. Agrawal. On the effectiveness of DNS-based server selection. IEEE Infocom'01, Anchorage (USA), April 2001.

[4] K. L Johnson, J.F. Carr, M.S. Day and M.F. Kaashoek. The measured performance of content distribution networks. 5th International Workshop on Web Caching and Content Distribution, Lisbon (Portugal), June 2000.

[5] B. Krishnamurthy, C. Wills and Y. Zhang. On the use and performance of content delivery networks. ACM Sigcomm Internet Measurements Workshop 2001, San Diego (USA), August 2001.

[6] C. Cranor et al. Enhanced Streaming Services in a content distribution network. IEEE Internet Computing July-Agost 2001. pp 66-75.

[7] G. Pierre, M. van Steen. Globule: a platform for self replicating Web documents. In Proceedings of the 6th International Conference on Protocols for Multimedia Systems, LNCS 2213, pages 1-11, Oct. 2001.

[8] G. Peng. "*CDN: Content Distribution Network*". January 2003. Experimental Computer Systems Lab. Technical Reports (TR-125)

[9] M. Castro, M. Dwyer and M. Rumsewicz. " Load balancing and control for distributed World Wide Web servers ", *Proceeding sof the 1999 IEEE Intemational Conference on Control Applications, Kohala Coast-Island of Hawaii,USA, August22-27, 1999*