

User Specific Request Redirection in a Content Delivery Network

Sampath Rangarajan Sarit Mukherjee Pablo Rodriguez

Abstract— This paper discusses user specific request redirection for personalizing responses to user requests in Content Delivery Networks (CDN). User specific request redirection refers to the process of redirecting user requests to a server on the content delivery network based on user specific information carried in the user request. Current redirection schemes in CDNs are either based on the authoritative DNS model or the URL rewrite model. The authoritative DNS model is not flexible to support user specific redirection as user specific information is not available to the DNS server; the URL rewrite model cannot support user specific redirection in practice because of the cost of on-the-fly URL rewrites required on a per user request basis. We discuss a technique that allows for flexible user specific request redirection. The technique is simple enough to be implemented at wire-speed in a switch.

I. INTRODUCTION

Content Delivery Service Providers (CDSP) such as [1], [2], [3], [4], [5] distribute content from origin sites to cache servers on the edge of the network and deliver content to the users from these edge servers (referred to as Content Delivery Servers or CDS). The distribution mechanism could be based both on push technologies such as multicasting the data to all the edge servers through terrestrial or satellite links or pull technologies such as those used by proxies. The goal is to decrease the latency of user access to the objects by delivering the objects from an edge server closest to the user.

A CDN consists of a set of CDSs across the Internet as well as a DNS infrastructure which is used to route user requests to the nearest CDS. For this to work, the DNS requests sent from the user browser now needs to be directed to the DNS of the CDSP. There are two ways to accomplish this.

Authoritative DNS: One way is for the CDSP to “takeover” the DNS functionality of the origin site and become the authoritative DNS for the origin site [6]. This is an easy approach to implement but the problem with this approach is that all the objects from the domain that has been taken over needs to be served from the CDSs. This approach will not work for example, if it is required that all html pages be served from the origin site but all the images (e.g., gifs and jpgs) be served from the CDSs.

The authors are with Center for Networking Research, Bell Laboratories, Holmdel, NJ and Microsoft Research, Cambridge, UK. Email: {sampath, sarit}@bell-labs.com, pablo@microsoft.com

URL Rewrite: In this approach, the authoritative DNS functionality still stays with the origin site’s DNS. Any top level page requested by a user will be served from the origin server. But before the page is served, all the embedded links found in the top level page are rewritten to point to the CDSP DNS so that requests to embedded objects can be redirected by the CDSP DNS to the closest CDS. This mechanism has been popularized by Akamai [1].

In this paper, we consider User specific request redirection. This refers to the process of redirecting requests to the same set of embedded objects in a top level page, arriving from different users, to different CDSs, based on information contained in the user request. The information found in the user requests that could be used include the user IP address, cookies present in the user request, user browser information, etc. Other information that is not contained in the user request could be used to make redirection decisions as well. These include time of day, priority of the user, cost of accessing the CDSP network and performance of the CDSP network.

The next section discusses and motivates the need for user specific request redirection. It is further argued that the two redirection mechanisms discussed above, namely Authoritative DNS and URL rewriting, cannot, in practice support user specific redirection. In the subsequent section, we propose a technique based on modifying HTTP response headers. This can be used to perform user specific redirection very efficiently in practice.

II. USER SPECIFIC REQUEST REDIRECTION

The main objective of performing user specific request redirection is to provide service differentiation based on user specific information. In addition, by performing such redirection to specific server IP addresses, inaccuracies due to DNS based redirection can be eliminated and performance can be improved by bypassing DNS. These issues are discussed below. In the following discussion, URL rewriting is used only to illustrate user specific redirection. In a latter section it is argued that this is not a very practical approach.

A. Service differentiation based on user specific information

Service differentiation based on user specific information is very important for service providers and is very

much tied to a user and the price the user is willing to pay. There are several different ways to ensure service differentiation with user specific redirection.

Service differentiation using DNS hierarchy: With user specific redirection, it becomes possible for the content provider to instruct the CDSP to provide different service levels to different users. For example, if two users carrying cookies “priority = high” and “priority = low” access a top-level page, the CDSP could provide the page rewritten in two different ways so that the service level provided to these two users, when they access the embedded objects in the top-level page, is different. One way to provide different service levels is to redirect the embedded object URLs to two different CDSP DNS hierarchies where one DNS hierarchy resolves the DNS query to one of a set of fast servers close to the user and the other DNS hierarchy resolves the query to one of a set of slower servers or a server which is on a site with limited bandwidth. In addition, requests from two different users could be redirected to two different service provider networks; for example, a high priority user may be directed to `www.cdsp1.com` DNS hierarchy which provides fast service but costs more and the low priority user may be directed to `www.cdsp2.com` which provides slower service but is cheaper.

Service differentiation using URL prefix: If URL rewriting is used to redirect user requests, when URLs are rewritten, different prefixes can be added to the URL path to provide indication to the CDS about the service level that should be provided to the user. For example, if embedded objects are prefixed with `level1` when the top level page is served to one user and with `level2` when it is served to another user, this could indicate to the CDS that if both requests arrive at the CDS, the request with a prefix of `level1` should be provided better service than the one with the prefix `level2`.

When redirection is performed, IP addresses could be used instead of host names. User specific redirection with IP addresses has advantages that are discussed in the next two sections.

B. IP address based redirection

Performing IP address resolution using the DNS of the CDSP has certain drawbacks as described below.

Source IP address inaccuracy: When a DNS request is received, the CDSP DNS checks the source IP address on the request, and based on this, returns the IP address of the CDS “closest” to the source IP address. This decision is made based on the assumption that the source IP address on the request is either the IP address of the user or one “close” to the user. But this may not be the case in practice. The source address on the DNS request is the IP address of the entity that sends the DNS request to the CDSP DNS.

Normally, this is the local DNS server on the user site. Depending on how DNS requests are forwarded, it could also be a DNS server further along the hierarchy from the local DNS. Therefore, the selected CDS is “closest” to the entity that sends the DNS request but not necessarily “closest” to the user. Server selection based on local DNS server IP addresses can result in a non-optimal server selection since users are frequently distant from their local DNS servers [7].

Inaccuracy due to DNS caching: When the CDSP DNS returns the IP address of the “closest” CDS, this IP address is cached by the browser and subsequently used to resolve domain names to IP addresses locally. This means that subsequent DNS queries to the same domain name will not even be sent to the CDSP DNS until the cached information is flushed. A non-optimal CDS may be used for this period of time if the network conditions change. Similarly, the local DNS or one of the DNS servers upstream towards the CDSP DNS could also cache DNS information. This type of DNS caching may lead to inaccurate server selection. One way to address this issue is to specify a DNS timeout (TTL) which is very small. There are, however, two problems with this approach. The first problem is that the DNS caches do not need to obey the timeouts. The second problem is that it is difficult to select this timeout. The timeout needs to be small enough so that dynamic server selection is possible; on the other hand, a DNS timeout value that is too small will lead to very frequent DNS lookups at the CDSP DNS server.

In addition to the drawbacks discussed above, DNS requests themselves add to the response time when content is retrieved from a CDN. The work in [8], [7] shows that DNS requests for rewritten URLs account for a significant overhead and clearly reduce the benefits of having content replicated at the network edge. It appears that the user response time to access a top-level page (where a user enters the URL in a browser or clicks on a link) needs to be within 3 to 4 seconds; otherwise the user may stop the download of the top-level page [9].

The aforementioned drawbacks can be eliminated by performing user specific redirection using IP addresses instead of host names. For example, with dynamic URL rewrite, when a URL `www.xyz.com/fashion/style.gif` is rewritten in a top level page, instead of pointing the URL to the CDSP DNS, the URL could be rewritten as `192.1.1.87/www.xyz.com/fashion/style.gif` where `192.1.1.87` is the IP address of the server “closest” to the user to whom the top level page is being served. The server IP address can be chosen accurately based on the source IP address of the client itself (as this is now visible to the URL rewriter). In addition, rewriting links with IP addresses will eliminate the need for a DNS lookup and hence make the download of the page faster. Using IP addresses for URL rewrite is especially beneficial to wireless users. DNS lookups across wireless links is a performance bot-

tleneck. This is more of a problem when the TTL for the DNS responses are kept very small. By using IP addresses thereby eliminating DNS lookups, the latency for wireless users will be significantly decreased.

III. IMPLEMENTING USER SPECIFIC REQUEST REDIRECTION

We now turn to the issue of implementation. Of the two redirection mechanisms discussed in an earlier section, the authoritative DNS is not flexible enough to provide user specific request redirection because DNS requests do not carry any user specific information. URL rewrite is a possible option for this purpose. We discuss URL rewrite in more detail below and argue that it is not a viable option for user specific redirection in practice.

The URL rewrite process involves rewriting the embedded links inside a html page to point to the CDSP DNS. For the most part, embedded objects that are automatically fetched by the browser are images that are large and will benefit from being served from the edge. During the content delivery process, the request to the top level page is sent to the content provider’s origin server. DNS requests to resolve the rewritten URLs inside the top level page will then be sent to the CDSP’s DNS. The CDSP DNS server returns the IP address for the CDS which is “closest” to the user. The request for these objects are then sent to this CDS and retrieved from that CDS.

Normally, the process of URL rewrite is performed statically using some software tool. Pages that need to be rewritten are parsed and the embedded URLs rewritten using this tool. This technique does not provide the flexibility to rewrite the same page in multiple different ways and provide different versions of the rewritten page to different users in order to personalize downloads of embedded objects based on information found in user requests (which is what is needed for user specific redirection). Thus, static URL rewriting is not a valid option for user specific redirection.

CDSP companies have also partnered with caching and switching vendors to provide what is referred to as *dynamic URL rewrite* [1], [10], [11], [12], [13]. In this technique, a reverse proxy cache [11], [10] or a load balancing switch [12] placed in front of the content provider’s servers performs URL rewrite on the objects. The transformations need to be performed are downloaded into the device. When a user request is received at the device for the first time, the request is sent to a server and the object is fetched. This object is then (URL-)rewritten at the proxy/switch and then sent to the user.

There are two obvious ways in which Dynamic URL rewrite can be used to implement User specific redirection: **a)** Every time a user request is received, the html page is parsed and the embedded URLs are transformed appropriately based on the requesting user. For example, if the transformation is performed based on IP addresses, each

URL will be transformed to point to the IP address of the best server for that user, and **b)** All the embedded URLs in the page are rewritten a priori in all possible ways in which it needs to be delivered to the users and all these copies are cached. When a user request is received, the appropriate page is delivered.

The first technique may not be practical as the html page needs to be parsed every time a user request is received. The second technique may be possible if the number of ways in which URL rewrite needs to be performed on a page is small. But if the transformation is performed using IP addresses, the number of ways in which URL rewrite can be performed on a page equals the number of server IP addresses. This will become impractical even for a reasonably large number of servers.

In the next section, we present a technique based on HTTP response header modification that enables user specific redirection to be implemented very efficiently. We refer to this technique as Request Deflection. The attractiveness of the technique is that it could be implemented in a switch at the packet level at wire-speed. A switch level implementation of our technique is not described in detail in this paper, but a brief discussion is included that shows the viability of such an implementation.

IV. REQUEST DEFLECTION

The Request Deflection technique enables user specific redirection without the need for parsing the html page every time a user request is received or performing a URL rewrite on the page a priori in all possible ways and caching these page copies. There are two steps to the Request Deflection process as discussed below.

A. Step 1: A Priori Transformation of Embedded URLs

In the first step, the page on which URL rewrite is to be performed is parsed and the embedded URLs are transformed as follows. This transformation is performed only once and the transformed page is cached.

1) *Handling absolute URLs:* All the embedded absolute URLs that are to be redirected to CDSs are converted into relative URLs. This process is shown in Figure 1. Note

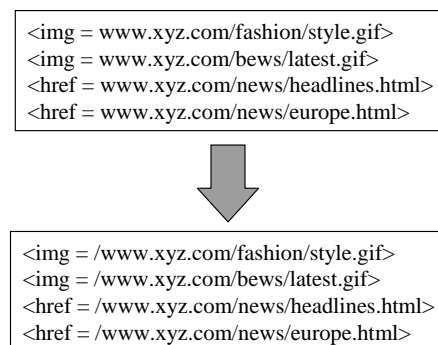


Fig. 1. Transformation of Absolute URLs

that the domain name of the CDSP is not prefixed to the

embedded URLs but the URLs have only been changed to relative URLs by prefixing a /.

2) *Handling relative URLs*: Note that if a relative URL exists in the page (see Figure 2), there must either be (a) a `<BASE=>` command present at the top of the html document that indicates the server from which this object needs to be retrieved, or (b) a `<BASE=>` command is not present at the top of the html page but the expectation is that the server will add a `Content-Base` token to the HTTP header which points to the server name (or IP address) of the main page that contains these embedded objects.

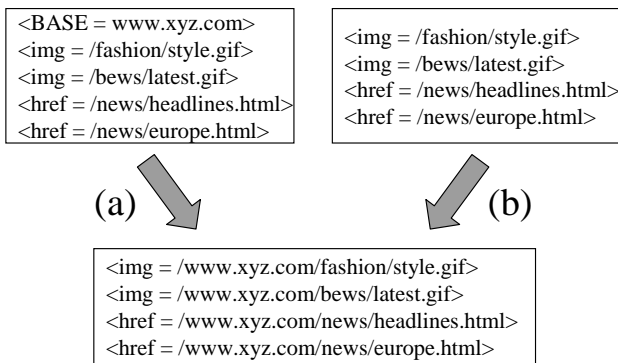


Fig. 2. Transformation of Relative URLs

For all relative URLs that are to be redirected to CDSs, in case of (a), the `<BASE=>` command should be removed and the server name found in the base command should be prefixed to the relative URL (after prefixing this relative URL, the embedded URL should still be relative; i.e., a / should be prefixed). In case of (b), the server name from where the main page was fetched should be prefixed to the relative URL (again leaving the URL still relative). In Figure 2, assume that the server name where the main page is present is `www.xyz.com`. Figure 2 (a) shows a page with the `<BASE=>` command. The server name found in the command is prefixed to the embedded relative URLs and the URLs are left in relative form with a / in front. Figure 2 (b) shows a page with no `<BASE=>` command but with the expectation that the server will send a `Content-Base` token as part of the HTTP header which will point to `www.xyz.com`, which is the server name where the main page is located. This page is similarly transformed by prefixing the server name of the main page leaving it intact as a relative URL with a / in the front.

B. Step 2: On-the-fly HTTP Response Header Modification

The second step is performed every time a user request is received. The page transformed using Step 1 is sent to the user, but the server includes a `Content-Base` token to the HTTP response header that contains the server name (or server name + URL prefix path) or the IP address (or IP address + URL prefix path) from which the embedded objects in the page need to be retrieved. This is a really fast operation in that the server does not have to parse the html page every time a user request is received, but can rewrite

the page differently for each user with respect to the server name or server IP address from which the object should be fetched.

As an example, consider requests from four different users C1, C2, C3 and C4. Assume that to provide service differentiation, users C1 and C2 need to be redirected to `www.cdsp1.com`, user C3 needs to be redirected to `www.cdsp2.com` and user C4 needs to be redirected to `www.cdsp3.com`. For each of these requests, the server will serve a transformed object (using Step 1) as shown in Figure 2, but will include a `Content-Base` token as part of the HTTP header which will be `Content-Base: http://www.cdsp1.com` for C1 and C2, `Content-Base: http://www.cdsp2.com` for C3 and `Content-Base: http://www.cdsp3.com` for C4. The user browsers at C1 and C2 will construct the embedded URLs as `http://www.cdsp1.com /www.xyz.com/fashion/style.gif`, etc, and fetch these objects from `www.cdsp1.com`. Similarly, C3 and C4 will fetch the objects from `www.cdsp2.com` and `www.cdsp3.com` respectively. If IP addresses are used for redirection and assuming that the “closest” servers to C1,C2,C3 and C4 are IP1,IP2,IP3 and IP4, the content-bases token used will be `Content-Base: IP1` for C1, `Content-Base: IP2` for C2, `Content-Base: IP3` for C3, and `Content-Base: IP4` for C4.

It is easy to see that Step 2 required for each user request requires very little overhead and can be implemented very easily in a reverse proxy. A more important observation is that adding a `Content-Base` token could be performed at wire-speed in a switch. We briefly discuss this issue in the next section.

C. Wire-speed Request Deflection

As described in the previous section, Step 1 of the request deflection process can be performed a priori and the html page can be cached. Assume that the server stores html pages after performing Step 1. The goal then is to perform Step 2, which is performed on a per user request basis, at wire-speed, using a switch in front of the server.

Let us consider performing Step 2 using IP addresses. When a main page is sent from the server to the user, if a `Content-Base` token is found in the HTTP response header, the value of the `Content-Base` token should be changed to `http://IP` where IP is the IP address of the server from which the embedded objects should be retrieved. If a `Content-Base` token does not exist, the string `Content-Base: http://IP` should be introduced into the HTTP response header. Let us consider the case where the `Content-Base` token does not exist and should be introduced into the HTTP response header. The case where a `Content-Base` token already exists can be considered as a situation where the existing token is removed and a new token that contains the IP address is introduced.

For this discussion, assume that the Content-Base token will be included into IP packets after re-assembly of IP fragments. That is, IP packets that are received at the switch are re-assembled to remove IP fragmentation and only then the Content-Base token will be added. At the switch, we cannot introduce new TCP segments into the flow. This means, when the Content-Base token is introduced into an IP packet, we need to make sure that the resulting IP packet does not go beyond a TCP segment.

The maximum size of the string that will be added to the IP packets will be $\text{strlen}(\text{Content-Base:}) + \text{strlen}(\text{http://}) + \text{strlen}(\text{maximum number of characters in an IP address}) + \text{an EOL character}$. The maximum number of characters in an IP address is 15. Thus, the maximum number of characters added will be 36. In order to accommodate these 36 bytes in a TCP segment without overflowing the segment, we need to make the server send TCP segments whose maximum length is 36 bytes less than the maximum segment size (MSS) that has been announced by the user. This way, the extra 36 bytes can be added to a TCP segment without exceeding the client requirement.

When the SYN packet for the TCP connection from the client (which will download the main page) is received at the switch, the value of the MSS TCP option is decreased by 36. For example, if the original MSS sent by the user is 1460, it is changed to 1424. This way, the server is forced to send TCP segments whose size does not exceed 1424. When the HTTP response is received from the server, the Content-Base token is added to the HTTP response header. Given that different tokens of the HTTP header can appear anywhere in the header, it is possible to add this header to the first response segment that is received from the server. Even after this addition, the TCP segment size will not exceed 1460 and so is acceptable to the client. Of course, to accommodate the change in the size of the TCP segment, the sequence numbers on all packets that follow this modified packet from the server to the client and the ack packet for this modified packet as well as subsequent packets from the client to the server need to be adjusted. Assume that after the addition of the Content-Base token, the length of the segment is increased by x , where $x \leq 36$. The ack sequence number on all packets from the server to the client that follow the modified packet should be decreased by x and the sequence number on the ack for the modified packet as well as subsequent packets from the client to the server should be increased by x . The length field of the modified packet should be adjusted as well.

The drawback of decreasing the MSS indicated to the server is that the server is forced to send TCP segments smaller than what the user can accept. This means, the server may end up sending more TCP segments. The justification for this is that main pages are mostly of small size and should fit in a few TCP segments. Because of this, the number of extra TCP segments sent should be very minimal.

A Request Deflection device (either a reverse proxy or a switch) needs a mechanism to decide the mapping between a user request and the IP address of the server that should be chosen to serve the embedded objects. The next section discusses two possible approaches for this.

D. Resolving server IP addresses

Downloading the mapping table into the Request Deflection device: Currently the CDSP DNS server serves two main functions: a) It collects information about the network and the status of the CDSs and runs a proprietary algorithm based on this information to construct a table of user IP address to CDS IP address mapping, and b) It replies to DNS requests from the users with the IP address of the appropriate CDS.

With Request Deflection using IP addresses, the CDSP still will compute a table of user IP address to CDS IP address mapping, but instead of serving as a DNS server, it will now download the table into Request Deflection device. There are two problems with this solution. Firstly, the CDSP DNS (which is now only an IP address mapper) needs to download this information into all the Request Deflection devices that may exist in the CDSP network. This problem is compounded by the fact that normally one CDSP DNS server does not compute all user to CDS mapping. This computation is performed by multiple DNS servers in the CDSP DNS hierarchy. This means that information needs to be downloaded into all the Request Deflection devices from all the DNS servers that compute the mapping. Secondly, the table size could be large; storing this table at all the Request Deflection devices could be difficult. This approach may be feasible only with small CDSP networks like the ones that will be found within an enterprise (enterprise level CDSP networks) [14].

Performing address lookup from the Request Deflection device: Instead of downloading the mapping table, the Request Deflection device could perform an operation similar to a DNS lookup at the CDSP DNS whenever a user to CDS mapping entry is needed. The device could send the IP address of the user to the CDSP DNS which will return the IP address of the optimal CDS for that user.

When such an approach is followed, there is no need to maintain and update a table at the device and the scalability of the CDSP network is not a problem anymore. Still, this solution requires that the CDSP DNS be changed so that it understands queries sent by the Request Deflection device. The CDSP DNS could let the devices cache this information by providing a timeout value. It is expected that the devices will obey this value. Also note that unlike the DNS lookups performed in current CDSP implementations, the user IP address that is sent to the CDSP DNS is the IP address of the real user. It is also possible to send more detailed information to the CDSP DNS about the user. For example, information such as the cookies contained in the

request and the user browser information could be sent in addition to the user IP address. This way, the CDSP DNS could return an IP address for a CDS whose identity depends not only on the “closeness” to the user but also on other user characteristics. In fact, multiple users making requests from the same user machine could be identified and redirected to different CDSs, if needed. Also note that the value returned by the CDSP DNS does not just have to be the IP address. It could return a URL prefix that not only contains the IP address, but also information about the customer making the request for billing purposes (for example, the customer code as used by Akamai [1]), serial numbers and other such information that will enable the CDSs to understand this request in a proprietary way.

V. CONCLUSIONS

User specific redirection is a useful technique for response personalization. This gives the ability for a content provider to differentiate users based on their IP addresses as well as other user identifiers such as cookies and browser type and provide service differentiation to these users although the objects are served to these users from a CDSP network. We have proposed a technique which makes user specific redirection practical. The advantage of the technique is that it could be implemented at wire-speed in a switch.

There is a limitation to the proposed technique. With Request Deflection, all objects referred to by the embedded URLs in a given page must be retrieved from the same CDS, thus, not permitting for different objects to be delivered from different CDSs. The reason for this is that Content-Base token can only specify one single server name or IP address to be used for all relative embedded objects in a given page.

Fetching all embedded objects from the same CDS is normally recommended to optimize end-user performance due to the following considerations: **a)** When different embedded objects are fetched from different CDSs, client browsers generate new DNS lookups and new TCP connection setups for each CDS. These new DNS requests create a significant overhead, thus, reducing the benefits of replicating content at the network edge and **b)** With the advent of HTTP/1.1 that uses persistent connections to download objects, fetching embedded objects from the same CDS enables the embedded objects to be fetched over a single (or a few) TCP connections. This means, the TCP connection setup overhead is minimized.

However, despite the fact that fetching all objects from the same CDS improves end-user experience, there are cases where different objects must be retrieved from different CDSs. This is the case when a single CDS cannot handle all the different content types embedded in a single page. For instance, different CDSs may be required to handle streaming content and images. One solution to overcome this limitation is to host a set of CDSs needed for all

content types (e.g. streaming, WAP content, images) under the same virtual IP address. These CDSs are then front-ended by a Layer 4 or Layer 7 switch. The switch takes care of redirecting different requests for different content types towards the right CDS based on the destination port number or on application specific information.

In some cases, e.g. due to network topology or network engineering reasons, CDSPs may not be able to host a set of CDSs needed for different content types under the same virtual address. When this is the case, the request deflection technique can only provide user specific redirection for the set of objects that can share the same CDS. The rest of objects that cannot be delivered from the same CDS do not benefit from such redirection. Instead, the URLs that refer to these objects are a priori rewritten using regular URL rewriting techniques and the page is cached. When the page is retrieved from the cache, the request deflection technique is only applied to those URLs that have not been rewritten a priori. Though this is a limitation of the request deflection technique we believe that in most scenarios a single CDS will be able to handle all object types, or a switch should be able to redirect requests for different object types into the right specialized CDS.

REFERENCES

- [1] Akamai Technologies, “http://www.akamai.com/en/html/services/content_delivery.html”.
- [2] Speedera Network, “http://www.speedera.com/technology/technology_implement.html”.
- [3] Mirror Image Internet, “http://www.mirror-image.com/services/instacontent_datasheet.html”.
- [4] Cable and Wireless, “<http://www.exodus.net/solutions/cdns/index.html>”.
- [5] Web-Caching, “<http://www.web-caching.com/cdns.html>”.
- [6] Unitech Networks, “<http://www.sw.unitechnetworks.com/en/products/idns/>”.
- [7] Anees Shaikh, Renu Tewari, and Mukesh Agrawal, “On the effectiveness of DNS-based server selection” in *Proceedings of the IEEE Infocom conference*, Apr. 2001.
- [8] Jussi Kangasharju, Keith W. Ross, , and James W. Roberts, “Performance evaluation of redirection schemes in content distribution networks” in *5th Web Caching Workshop, Lisbon*, 2000.
- [9] Netli, “<http://www.netli.com>”.
- [10] Novell, “<http://www.novell.com/news/press/archive/2000/10/pr00103.html>”.
- [11] CacheFlow, “http://www.cacheflow.com/files/whitepapers/the_cache_flow_akamaizer_white_paper_final.pdf”.
- [12] F5 Networks, “<http://secure.f5.com/news/articles/article052300b.html>”.
- [13] Nortel Networks, “http://www142.nortelnetworks.com/bvdoc/alteon/whitepapers/accel_delivery.pdf”.
- [14] Cisco, “http://www.cisco.com/univercd/cc/td/doc/product/webscale/content/cdnent/encdn302/m_en302.htm”.