DEW: DNS-Enhanced Web for Faster Content Delivery

Balachander Krishnamurthy
AT&T Labs–Research
Florham Park, NJ, USA
bala@research.att.com

Richard Liston
Georgia Tech
Atlanta, GA, USA
liston@cc.gatech.edu

Michael Rabinovich
AT&T Labs–Research
Florham Park, NJ, USA
misha@research.att.com

ABSTRACT

With a key component of latency on the Web being connection set up between clients and Web servers, several ways to avoid connections have been explored. While the work in recent years on Content Distribution Networks (CDNs) have moved some content 'closer' to users at the cost of increasing DNS traffic, they have not fully exploited the available unused potential of existing protocols. We explore ways by which a variety of Web responses can be piggybacked on DNS messages. While we evaluated our idea in the Web context, the approach is generic and not restricted to Web responses. We propose an architecture for HTTP piggybacking in DNS messages and carry out a detailed performance analysis based on a trace-driven simulation study. Our architecture requires minimal extensions to existing protocols, utilizing only the allowed optional fields for these extensions. It is fully compatible and can coexist with the current Web.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Network Architecture and Design

General Terms

Performance,Design

1. INTRODUCTION

Many researchers have explored ways to improve user-perceived latency and reduce load on origin servers and the network. A common theme among the various strands of work that have turned out to be beneficial is one that allows incremental deployment rather than new large scale changes to existing user's practices. Consequently, we seek ways to exploit inefficiencies in the current, largely stable and generally difficult to modify protocols involved in a Web download (HTTP/1.1 is *not yet* a standard six years after introduction!), while minimizing any protocol extensions and making sure these extensions can coexist with the current practice.

A typical Web transaction (for more details, see [6], Section 15.4.1) involves a few protocols (DNS, HTTP, UDP, TCP) with multiple short transactions among the associated entities (clients, proxies, servers).

Inefficiencies in the transport layer have been largely squeezed out by protocol changes at the HTTP layer (persistent connections, pipelining) as well as proposals to move some HTTP traffic from TCP to UDP [2, 13]. Latency reduction has been advanced by standard proxy caching techniques and in some cases via non-standard ones (such as DNS-based content distribution), or again via protocol extensions (such as compression, delta-encoding [11], and HTTP range requests). Piggybacking, which dates back to pre-TCP days, has been exploited for obviating unnecessary cache validations [7] and for sending hints [3]. Piggybacking techniques that exploit existing protocols, have low overhead, reduced deployment costs, are easier to test and hopefully be adopted.

In this paper, we explore ways to speed up Web delivery by piggybacking some or all of a HTTP resource in DNS responses. Most HTTP transactions are preceded by a DNS lookup of the Web site's server name which (unless cached) is satisfied by authoritative DNS servers. Authoritative DNS servers are often close, in a network sense, to the machines whose names they serve [4]. To access a Web object, clients often must communicate with a distant host to resolve the Web server name, only to go back to the same location again for the object itself. Our approach attempts to reduce the frequency of such repeated communication, and thus reduce the latency of Web accesses. It will therefore be especially beneficial for environments with long message latency, such as dial-up clients, clients connected by satellite links and clients accessing very remote Web sites.

We examine different scenarios where useful information can be piggybacked in the DNS message exchange that comes before regular responses. It is not necessary for the entire HTTP response to be piggybacked. With some simple adjustments, we can alter the piggybacked response (e.g., compress it), or send semantically significant portions (an initial portion of predefined size or differences between versions of a resource, if a previous version is cached).

A DNS trace study [5] done at MIT and the Korea Advanced Institute of Science and Technology examined the failure rates as well as overall performance and reported on usefulness of caching. A recent example of exploiting existing DNS servers (although for an entirely different application, that of measuring latency between arbitrary Internet hosts) and avoiding installation of new measurement infrastructure is the King system [4].

Although in this paper we focus on piggybacking HTTP on DNS, the idea is more general and could be used in any context where a transaction involves a sequence of protocols. For example, in some peer-to-peer systems, a file download is preceded by a directory query to find a relevant peer. Furthermore, while many P2P systems currently use hardwired IP addresses, short peer-to-peer query responses can be piggybacked in DNS responses, when a DNS query is required in that context.

2. BACKGROUND

Web interaction today involves the following mechanisms. Every HTTP client ('client') is configured to use a DNS server ('local DNS' or LDNS) to resolve URL hostnames into their IP addresses. Typically a group of clients (usually in the same client location) share the same LDNS. LDNSs in turn are configured to use well-known *root DNS servers*, which keep a database of the DNS servers that maintain hostname-to-IP-address mappings for given domains (*authoritative DNS servers* referred to here as remote DNS or RDNS). A Web interaction typically starts with the client sending a DNS query to LDNS, which resolves it from the appropriate RDNS (first obtaining the RDNS identity from a root server) and forwards the response to the client. The client then opens a TCP connection to the Web server and downloads the page using HTTP protocol. The DNS interactions occur over connectionless UDP and involve simple exchange of request/response datagrams.

DNS makes extensive use of caching at all levels. A client caches DNS responses to avoid the overhead of DNS queries when accessing multiple URLS from the same Web site. An LDNS caches responses to avoid going to RDNS when another client from the LDNS's group previously already resolved the same hostname. Clients sharing the same LDNS share the LDNS's cache of DNS responses. To keep cached responses from becoming stale, RDNS servers assign *time-to-live (TTL)* to the responses, indicating validity duration of cached responses.

3. DNS-ENHANCED WEB

Here we describe our proposal for DEW—DNS-Enhanced Web. Before a Web client can send the HTTP request, it often needs to resolve the host name of a Web server. The client's resolver is modified to piggyback the HTTP request on the unused portion of its DNS request to its DNS server. DNS servers (both local and authoritative) extract piggybacked HTTP requests and may choose to piggyback HTTP responses or parts of them on the DNS responses. We consider *strict piggybacking*, where the piggybacked information fits entirely into the unused portion of the DNS response, and *extended piggybacking* where the DNS response may include up to two additional datagrams. Limiting the size of extended piggybacking allows DEW to leave congestion control (essential in large data transfers) to TCP mechanisms.[1] We refer to the maximum amount of piggybacked HTTP data as *size threshold*. Our idea leaves open different design choices. We leave them for an extended version of this paper and concentrate here on the architecture shown in Figure 1.

[1]TCP starts a transfer of HTTP responses with congestion window of two segments. In this paper, we study cases where DEW's transfer is more conservative (strict piggybacking), equivalent (one extra datagram) and slightly more aggressive (two extra datagrams) than TCP's.

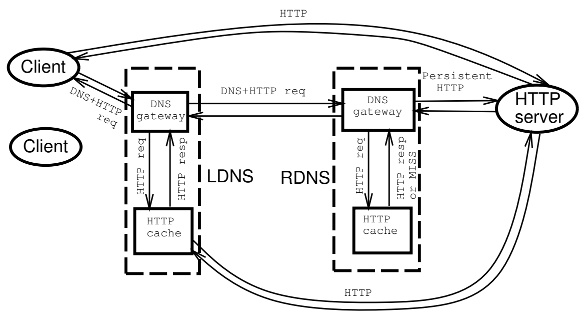


Figure 1: DEW architecture that combines DNS servers and HTTP caches.

3.1 The Architecture

A high-level architecture of DEW is shown in Figure 1 (root DNS servers are not shown, they are not modified in DEW). DEW-enabled DNS servers add an HTTP cache to their normal DNS cache. When the LDNS receives a query with piggybacked HTTP request, LDNS can respond locally if it has both DNS and HTTP responses in its respective caches. If neither response is cached, LDNS forwards the entire query to the authoritative DNS for the hostname provided the latter is DEW-capable. The remote DNS server contacts the origin server, with which it may maintain a persistent TCP connection. If the origin server decides that it is appropriate to do so (based on, e.g., whether the HTTP request is idempotent in case of potential loss of DNS response), it sends the desired object, or a part thereof, to the RDNS server. The RDNS server piggybacks this HTTP response in its DNS reply to the LDNS, which in turn forwards the reply to the HTTP client. The RDNS can also cache the HTTP response in its HTTP cache for future use. If the HTTP client receives a DNS response that includes the requested HTTP object, the client can immediately use it. If the DNS response contains just the IP address of the host (or only a portion of the HTTP response), the client will obtain the (remaining portion of the) object from the HTTP server using a HTTP Range request.

When the LDNS can satisfy the hostname query from its DNS cache but does not have the requested HTTP object in its HTTP cache, the LDNS can fetch the object from the Web server and piggyback it on its DNS response to the client. Its HTTP cache is therefore a standard client HTTP proxy, but it receives requests from LDNS with the hostnames in the requests already resolved. This cache must implement all Web proxy features such as cacheability and cache validation rules. Although piggybacking in this case is limited to the interaction between the client and LDNS, it is important especially for dial-up users because the communication between their HTTP clients and LDNSs occur over dial-up connections with typical latencies of around 200ms. Considering that a document download often involves HTTP interaction with multiple Web sites (e.g., the container HTML page from the origin site, the site's embedded images from a CDN, and banner ads from advertising sites), the latency overhead for communication with LDNS and the TCP set-up can be significant. If responses from various sites can be piggybacked, the benefit accumulates. However, parallel downloads will lower the probability of cumulative benefit.

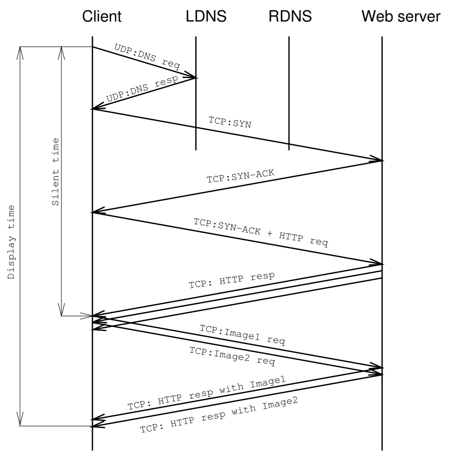


Figure 2: Message exchange in today's Web (DNS response cached by LDNS).

DEW could be especially useful in existing deployed Content Distribution Network architectures. CDNs, such as AT&T's ICDS or Akamai, already use DNS to locate CDN mirrors (presumed to be) closer to the browser client's LDNS. Since most CDNs still deliver small static images that have a typical size distribution in the few hundreds of bytes, they make excellent candidates for DEW. CDNs already run their own DNS hierarchy and have access/control over delivery of the static image resources. They typically return DNS responses with low TTLs and do not maintain persistent HTTP connections with the clients. CDNs are good candidates to accept piggybacked DNS queries and actually reduce latency for users in fetching these objects by obviating the HTTP connection.

3.2 Use Cases

DEW uses extensions allowed in the existing DNS protocol specification and DEW-enabled components (HTTP clients, LDNS and RDNS servers, and Web servers) can coexist with their current counterparts. Section 6.1 provides details on the DEW protocol and on fitting DEW into DNS. Here, we describe various scenarios in DEW and compare their timing properties with the current Web.

Consider a client that fetches a page with two embedded images, image1 and image2. In the time diagrams, we assume a dial-up client, with the RTT between the client and other hosts being three times the RTT between well-connected hosts on the Internet. Let T_l and T_r be RTTs between the client and its ISP and between the ISP and the Internet, respectively. The RTT between the client and the Internet is $T_l + T_r$. Let C and E be the size of the container page and embedded content, and $D = C + E$ be the size of the entire document. Let the time to download a file of size f over TCP be $t(f)$, which is the time interval between the first and the last byte of this content arriving at the client (that is, excluding connection establishment). Let $B(f)$ be time for a bandwidth-limited download of such file over UDP.

When the client has the IP address of the host in its local DNS cache, it fetches the document from the Web server over HTTP, both in today's WEB and in DEW. Such a request is termed *non-participant* as no piggybacking is done by DEW.

Now assume that host name needs to be resolved. We analyze the case when there is no previously established persistent connection between the client and the Web server. This is the most likely case because if the client must resolve the host name, then either this is its first access to this site, in which case the client cannot have an existing connection, or the previous DNS resolution timed out, in which case it is most likely that the persistent connection timed out also. The analysis of the cases where there is a persistent connection is very similar.

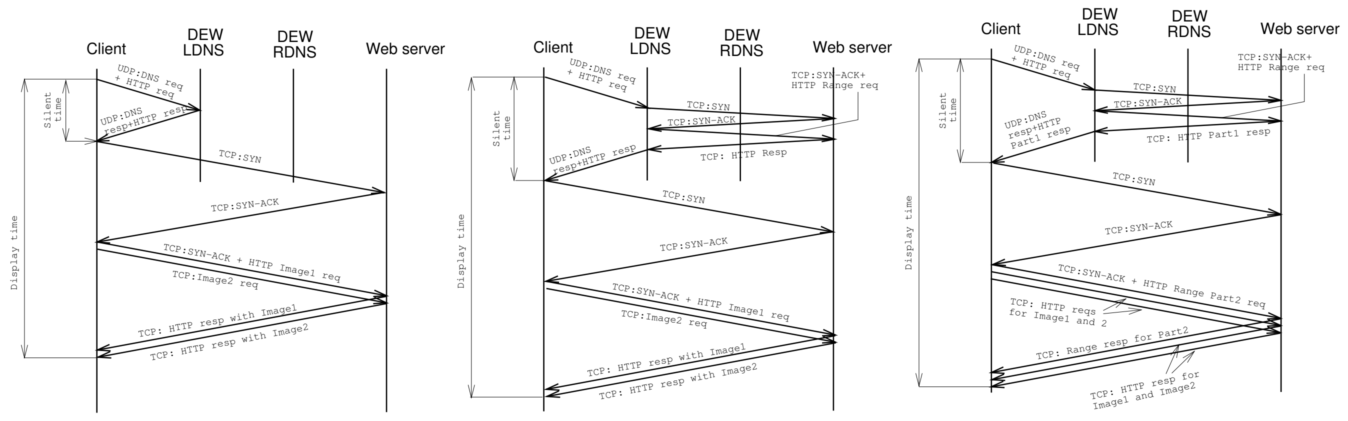
Figure 2 shows the time diagram of the messages involved in today's Web interaction when the local DNS has the requested host name in its cache. The silent time for this interaction is $S_1 = T_l + 2(T_l + T_r)$ and the display time is between $S_1 + t(D)$ and $S_1 + (T_l + T_r) + t(D)$ depending on the degree of the overlap between the delivery of the container page and the requests for embedded content. The case when LDNS does not cache the requested host name adds T_r to all expressions.

With the architecture of Figure 1 and assumption that a client sent a DNS request with HTTP request to its LDNS raises following cases:

- The LDNS has the host name in its DNS cache and the container page in its HTTP cache, and the page is below size threshold. The time diagram for this case is shown in Figure 3a. Compared to Figure 2, we observe the reduction in both the *display time*, which is the total delay before the user sees the entire document with all the images, and especially the *silent time*, which is the delay before the browser starts showing some information to the user [1]. This request is called a *full cache hit*. The silent time here is T_l and display time is either $T_l + 2(T_l + T_r) + B(C) + t(E)$ if there is embedded content or $T_l + B(C) = T_l + B(D)$ otherwise.[2] For a more vivid comparison with current Web, we can further simplify the display time expression for the case with embedded image by noting that $B(C) + t(E) < t(D)$. Then, the conservative estimate for display time in this case becomes $T_l + 2(T_l + T_r) + t(D)$.

- The LDNS has the host name in its DNS cache but the container page is not in its HTTP cache. The latter now issues the HTTP Range request to the Web server (Figure 3b). The Range request is needed to limit the transfer to a initial portion of the page that is below the size threshold. Still assuming for now that the page is below the threshold, the server responds with the entire object, which the LDNS caches in its HTTP cache and forwards to the client in the DEW response. The effect on response time here depends on the environment. For a U.S. dial-up client accessing a domestic Web site, there will be a reduction in silent time because DEW replaces TCP handshake over slow dial-up link (RTT in the order of 200ms) with TCP handshake over faster Internet link (cross-country RTT that are roughly a third of dial-up times). If there were no embedded images this would translate into similar sav-

[2]This and similar expressions later in this section are a slightly conservative estimate since the display time could be less due to a possible partial overlap of $B(C)$ and $t(E)$. Any difference would be small however because C is small in this case.


Client
DEW LDNS
DEW RDNS
Web server
Silent time
Display time
UDP:DNS req + HTTP req
UDP:DNS resp+HTTP resp
TCP:SYN
TCP:SYN-ACK
TCP:SYN-ACK + HTTP Image1 req
TCP:Image2 req
TCP: HTTP resp with Image1
TCP: HTTP resp with Image2
TCP:SYN-ACK+ HTTP Range req
TCP: HTTP Resp
UDP:DNS resp+HTTP Part1 resp
TCP: HTTP Part1 resp
TCP:SYN-ACK + HTTP Range Part2 req
TCP: HTTP reqs for Image1 and 2
TCP: Range resp for Part2
TCP: HTTP resp for Image1 and Image2

(a) DNS and HTTP response cached by LDNS (*full cache hit*).

(b) DNS but not HTTP response cached by LDNS (*full DEW request*).

(c) Partial HTTP piggybacking (*partial HTTP request*).

Figure 3: Message exchanges in DEW.

Category	Explanation
Non-DEW-participant	Client knows Web site's IP address, no DNS request sent
DEW participant	Client does not know Web site's IP address
Full DEW participant	HTTP response to client is fully piggybacked on DNS response
Partial DEW participant	Initial portion of HTTP response is piggybacked followed by a Range HTTP request.
Full cache hit	Full participant satisfied from LDNS's cache.
Partial cache hit	Partial participant satisfied from LDNS's cache.
Full HTTP request	Full participant such that LDNS's cache contains DNS response but not HTTP response.
Partial HTTP request	Partial participant such that LDNS's cache contains DNS response but not HTTP response.
Full DEW request	Full participant such that LDNS's cache does not contain DNS response
Partial DEW request	Partial participant such that LDNS's cache does not contain DNS response

Table 1: Summary of request categories.

ings in display time. With embedded images, depending on the environment, DEW might *increase* the display time in this case because it trades one RTT between client and the Internet for two RTTs between the ISP and the Internet.[3] This request, called a *full HTTP request*, has a silent time of $S_2 = T_l + 2T_r$; display time of $S_2 + 2(T_l + T_r) + B(C) + t(E)$ (or, conservatively, $S_2 + 2(T_l + T_r) + t(D)$) if there is embedded content or $S_2 + B(C) = S_2 + B(D)$ otherwise. We assume that the link to the client is the bottleneck thus determining the container page's download time.

- The LDNS does not have the host name in its DNS cache. It forwards the DEW request to RDNS, which responds with the message containing both the requested IP address and the HTTP page.[4] Similar to the cache hit case, this case (named a *full DEW request*) results in the reduction of silent and display time, the latter due to better pipelining. The expressions for the silent and display times are the same as in the case of the full cache hits with added T_r for RDNS query in both expressions.

- The page is above the size threshold. When the LDNS has the host name in its DNS cache but the container page is not in its HTTP cache, LDNS does not know the page size. So, its HTTP cache issues the Range request for P initial bytes, which allows LDNS to limit the response size without knowing the page size. When the Web site returns the initial portion of the container page, it is cached for the future use and returned to the client piggybacked on the DNS response. This allows the browser to immediately start displaying the received portion of the data, reducing the silent time.

  The browser also issues HTTP Range request for the remaining part of the container page. Unlike in the current Web, if embedded objects happen to be specified in the initial portion of the container page already received, the browser can pipeline the requests for these objects right

[3]Because this case assumes a small container page, there would be little overlap between the download of the container page and requests for embedded content, so the Web interaction would involve at least three RTTs between the client and the Internet as in Figure 2.

[4]RDNS would obtain the HTTP page either from its HTTP cache or from the origin HTTP server. In the latter case, we assume that RDNS and the HTTP server are connected with a high-bandwidth low-latency link because they usually belong to the same enterprise. Thus, we do not distinguish these two cases in our analysis.

	Web	DEW			
		full cache hit	full HTTP req	partial HTTP req	partial cache hit
Silent	$3T_l + 2T_r$	T_l	$T_l + 2T_r$	$T_l + 2T_r$	T_l
Display	$3T_l + 2T_r + t(D)$ to $4T_l + 3T_r + t(D)$	$3T_l + 2T_r + t(D)$ or $T_l + B(D)$	$3T_l + 4T_r + t(D)$ or $T_l + 2T_r + B(D)$	$3T_l + 4T_r + t(D)$ to $4T_l + 5T_r + t(D)$	$3T_l + 2T_r + t(D)$ to $4T_l + 3T_r + t(D)$

Table 2: Download times of Web and DEW with LDNS caching the DNS response

	Web	DEW	
		full DEW req	partial DEW req
Silent	$3T_l + 3T_r$	$T_l + T_r$	$T_l + T_r$
Display	$3T_l + 3T_r + t(D)$ to $4T_l + 4T_r + t(D)$	$3T_l + 3T_r + t(D)$ or $T_l + T_r + B(D)$	$3T_l + 3T_r + t(D)$ to $4T_l + 4T_r + t(D)$

Table 3: Download times of Web and DEW when LDNS does not have the DNS response (the estimates for DEW are conservative).

after the HTTP Range request for the remainder of the container page (see Figure 3c). The effect of DEW on the display time in this case is unclear because improved pipelining is offset by an extra TCP handshake over the fast link. We call client requests processed according to this scenario as *partial-page HTTP requests*. The silent time here is $S3 = T_l + 2T_r$. The display time is between $S3 + 2(T_l + T_r) + B(P) + t(C - P + E)$ and $S3 + 3(T_l + T_r) + B(P) + t(C - P + E)$ depending on whether the embedded content is referenced in the beginning or the end of the container page. The opportunity for pipelining HTTP requests is higher here than in the current Web. For example, Figure 3c shows a scenario when all HTTP requests from the client to the Web server are pipelined, while today the request for the container page inherently precedes the pipelining. We can conservatively replace the last two terms in both formulas with $t(D)$.

- The other two cases, *partial-page cache hits* and *partial-page DEW requests*, are analogous to their full-page counterparts. The initial portion of the container page in the partial-page cache hit case is satisfied from LDNS, resulting in significant reduction in silent times. Similar to partial-page HTTP request, partial-page cache hit improves pipelining of the remaining portion of the page and embedded images, but this time without extra TCP handshake between LDNS and the Web server, reducing the display time. The silent time here is $S4 = T_l$. Recalling that $B(P) + t(C - P + E) <= t(D)$, the conservative estimate for the display time is between $S4 + 2(T_l + T_r) + t(D)$ and $S4 + 3(T_l + T_r) + t(D)$ depending on whether the embedded content is referenced in the beginning or the end of the container page.

  The partial-page DEW request occurs when LDNS piggybacks the HTTP request over its DNS request to RDNS, and the RDNS returns only the initial portion of the page because the page exceeds the size threshold. Similar to full-page DEW requests, this scenario reduces silent time, and depending on where the embedded images are specified, the display time due to better pipelining. The RDNS query adds T_r to the expressions for the silent and display times for the full-page DEW request case.

We summarize request categories in DEW in Table 1 and the expressions for silent and display times for different categories in Tables 2 and 3. We study how often these scenarios occur in real traces in Section 5. In many of the above scenarios, DEW reduces the total number of packets exchanged by eliminating some TCP handshake and acknowledgment messages. Quantifying the effective reduction in network loads is beyond the scope of this paper.

4. EXPERIMENT METHODOLOGY

The DEW architecture will have different effects from the vantage points of the client, the proxy (if any), and the content provider. We used trace-driven simulations to evaluate the effect of the system from these three perspectives. Given the time analysis of individual scenarios from Section 3.2, we focus on understanding how often these scenarios occur.

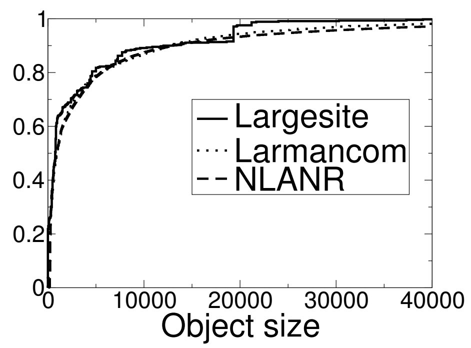


Figure 4: CDF of object sizes in the logs.

Table 4 shows the traces used in our study. Larmancom is a trace merged from three proxy logs at three sites of a large manufacturing company recorded over the course of a week in July 2001. This trace contains distinct client IP addresses,

Name	Date	Total Requests	Mean/Median Object Size (bytes)	Simulation (section)
Larmancom	July 8-14, 2001	216,251,781	7610/965	client (5.1), proxy (5.2)
NLANR	October 21-27, 2002	11,815,289	12050/902	proxy (5.2)
Largesite	August 1-7, 2001	51,747,434	3680/792	content provider (5.3)

Table 4: Logs used for simulations.

anonymized such that class locality is preserved, allowing us to use it for studying both client and proxy perspectives. The NLANR trace is taken from a proxy in the NLANR cache hierarchy [14]. The client IP addresses in this trace are sanitized; hence we used this trace only for the proxy perspective evaluation. For the content provider evaluation we used a trace from a large commercial Web site.

One important aspect determining the request category is the size of the response. Table 4 lists mean and median responses sizes in our traces, and Figure 4 shows the CDFs of the response sizes. The size distributions all have a similar shape and confirm previous studies showing a prevalence of small objects. With a median size of less than 1K, a majority of the objects are candidates for strict piggybacking.

The effectiveness of DEW depends on the following key factors: size of objects that can be returned in a DEW response, TTLs of objects (both DNS responses and HTTP objects) in the DEW cache, and grouping of clients into clusters that share the same LDNS server. In our simulations we have direct control over the first two factors. We ran the simulations using different combinations of values for these factors. To reduce the number of combinations, we used the same value for TTLs of both DNS and HTTP responses. We also simulated the effect of compressing objects. We ran our simulations assuming that textual objects are compressed by a specified ratio (since compression increases the effectiveness of DEW, we assumed conservatively that only textual objects are compressible). We inferred textual objects by either MIME type (when available) or by the file name extension from the object's URL (e.g., URLs ending with ".html" or ".asp" were assumed to be textual objects). We chose a compression ratio by measuring it on homepages of top 148 Web sites as discussed in the next section.

To evaluate the deployment of DEW at the proxy we simulated a DEW-capable LDNS server that is used solely by the proxy. In the client- and content provider-based evaluations we simulated deployment of DEW-capable LDNS servers at each client location. The client evaluation assumes no proxy participation. The content provider evaluation is agnostic to proxy participation since clients in the logs may represent end-users as well as proxies. In the last two evaluations, we needed to group individual clients into clusters sharing the same LDNS server.

For the client-based evaluation we do not have specific information about the sharing of LDNS servers by the clients in the logs, and we assume that clients are grouped by the most significant three octets of their IP address. This imprecise grouping can affect the breakdown of DEW participants into categories[5] but does not change the overall percentage of DEW participants.

For the content-provider simulation, our trace is from the same site and the same time period that was used in a previous study to assign Web clients to their LDNS servers [8]. Thus, we have precise client-LDNS maps for content-provider evaluation. We used these maps to precisely simulate sharing of LDNS servers by clients and to maintain the state of the LDNS servers.

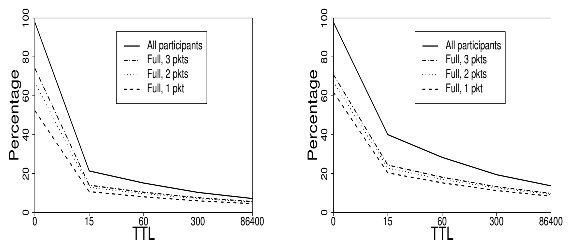
During the simulation, each request in the trace is examined according to the choice of values for the current simulation run and the state of the DEW-capable LDNS cache. The request is then categorized according to the use scenarios defined in Section 3.2. The output of each simulation is a count of the number of requests in each category. We do not model packet loss. As explained later in Section 6, DEW's behavior in high-loss environments should not be worse than the current Web.

5. RESULTS

We now present the results of evaluating the impact of the system from the perspectives of clients without a proxy, a client proxy, and a content provider, which is agnostic as to whether or not its clients represent proxies or end-users. Our main findings are:

- There is a reasonable opportunity for piggybacking of HTTP on DNS messages. While this opportunity drops for higher TTLs, it remained significant (20%) in the content provider case even for TTL of one day.
- Most of the DEW benefits come from piggybacking HTTP on DNS responses that come all the way from RDNS to LDNS to clients. Contributions from HTTP caching and downloading functionality of DEW-enabled LDNS was low.
- DEW Participation expectedly goes down in the presence of proxies due to reuse of cached DNS resolutions but there is still roughly 20% participation in the Larmancom trace and 40% in the NLANR trace for DNS TTL of a minute.
- Compression benefits were inconclusive: it increased a fraction of full DEW participants in the content provider and NLANR traces, but did not have much effect in the Larmancom trace, for both client and proxy perspectives.

[5]Because the imprecision is likely on the side of overestimation of LDNS sharing, it may increase the percentage of participants categorized in Section 3.2 as cache hits and HTTP requests at the expense of DEW requests.



(a) Raw requests.

(b) Container pages.

Figure 5: Percentage of DEW participants (Larmancom logs).

5.1 Client

We used the Larmancom logs to to evaluate the system from the client's viewpoint. We ran the simulations grouping clients as described in Section 4 using the common TTLs of 15, 60, 300 and 86400 seconds (1 day), as well as 0 seconds as a boundary case. Some clients do not obey small TTLs in DNS responses and cache them anyway for some period of time (up to a few minutes). For these clients, DEW participation would freeze at the level corresponding to this "imposed TTL" for any smaller TTL returned by the RDNS. We considered size thresholds of 1180, 2660, 4140, which approximate responses taking one, two, and three UDP packets. We assume a 1480-byte datagram and deduct 300 bytes from the total size for HTTP headers and the DNS response itself. Although the DNS specification limits DNS responses to 512 bytes [10], this limitation was specified before MTU discovery has made 1480-byte packets ubiquitous and consequently led to the Proposed Standard (RFC 2671 [16]) that specifies a method of increasing the DNS response size. A DEW-capable server can just assume that, since it is receiving a DEW request, the LDNS can handle responses of a size greater than 512 bytes.

Figure 5a shows the total percentage of DEW participants and the percentage of full DEW participants (those completely fitting into the size threshold) for each combination of values. Participants include requests categorized as DEW Requests, HTTP Requests and Cache Hits.

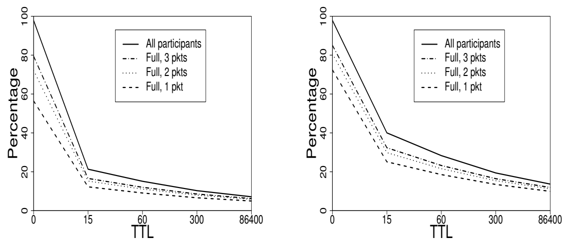
The boundary case of zero TTL indicates the maximum possible percentage of requests that are candidates for DEW participation. A request is not a DEW candidate if its HTTP request method is other than GET or HEAD, or if it does not involve a DNS lookup (hostname portion of the URL is a raw IP address). Approximately 97% of the requests are potential DEW participants. 2% of the requests have IP addresses in the URLs and 1% employ non-GET or HEAD request methods.

With a higher TTL, clients use cached DNS resolutions for requests that revisit the same Web server; as explained in Section 3.2, these requests become non-DEW participants. The higher the TTL the longer a client can use cached DNS responses, and the fewer requests are potential DEW requests. Each of the participant curves decreases as TTL increases. The percentages of total DEW participants are 97%, 21%, 15%, 10% and 7%, respectively. This quick dropoff is due to HTTP traffic pattern: loading a Web page is typically followed by a burst of requests to the same server (or a limited number of servers) for embedded objects (images, scripts, etc.). These subsequent requests are "masked" because the client already has the IP address of the destination server and will not contact the LDNS for the subsequent requests until the DNS TTL expires. Low DNS TTLs are common for objects delivered by CDNs. Clients are therefore expected to experience a greater performance enhancement from DEW for CDN-delivered objects.

The initial requests for container pages (i.e., those that embed images and other objects) are of particular importance because their silent time determines the silent time for the entire Web document. As our time analysis of Section 3.2 showed, DEW participation of a container page also improves display time for the entire document in most cases, due to better pipelining. We thus examined the percentage of DEW participants among requests for container pages only. Figure 5b shows the proportion of DEW participants and full DEW participants among container pages. Comparing Figures 5a and 5b we see that higher percentage of container pages benefit from the DEW architecture. For example, for TTL of 300 seconds, approximately 20% of requests to container pages are full or partial DEW participants, and 11% are full participants with strict piggybacking (i.e., they can be piggybacked in a single DNS response packet).

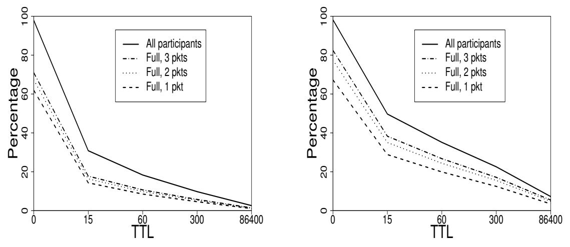
To gain insight into the behavior of Figure 5b, we inspected the breakdown of the categories of DEW participants from container pages. We normalized each of the points of Figure 5b to 100% and examined the contribution of the category to the normalized total. For example, in Figure 5b, for a TTL of 15 seconds, the "all participants" curve shows approximately 40% of requests being either partial or full participants. We found that nearly 100% of these requests are DEW requests. This was true for all TTLs up to 300 seconds. Similarly, nearly all full participants are DEW requests for any size threshold. Overall, we found that the DEW Requests category far outweighs the HTTP requests and Cache Hits. Even with a TTL of one day, the proportion of cache hits is only around 10%. This suggests that requests for container pages in conjunction with DNS requests exhibit only insignificant sharing among clients belonging to the same client site. Thus, most benefits come from the piggybacking capability and not from the caching and HTTP functionality of the LDNS server.

Most DEW benefits occur when a request is a full participant. Figure 5 shows that full participants constitute at least half, and often a majority, of participants for all size thresholds. We investigated the possibility of further increasing the proportion of full participants by combining DEW with the practice of compressing objects at the content provider. We downloaded the home page of 148 popular Web pages [9] and compressed them via `gzip`. The mean and median compression ratios were 4.6:1 and 4.4:1, respectively. Using a conservative compression ratio of 4:1, we simulated the compression of all objects deemed compressible as per the criterion mentioned in Section 4. These results are shown in Figure 6a. There is very little change between Figure 5a and Figure 6a. For example, with a TTL of 15 seconds, 4:1 compression raises the percentages of full participants from 21%, 14%, 13% and 11% to 21%, 17%, 15% and 12%, respectively. This result is not surprising given the small gap between full and



(a) Raw requests (b) Container pages.

Figure 6: Percentage of DEW participants, 4:1 compression (Larmancom logs).



(a) Larmancom trace, no compression (b) NLANR trace, 4:1 compression

Figure 7: Percentage of DEW participants for container pages at a proxy.

total participants on Figure 5.

Although compression provided little benefit to DEW when considering all requests—container pages and embedded objects—compression proved more beneficial when considering only container pages. As shown in Figure 6b, a noticeably higher percentage become full participants, particularly for lower TTLs. This suggests that container pages for domain names that generally have lower TTLs, such as those delivered by CDNs, will receive an added benefit by compressing container pages.

5.2 Proxy

To evaluate the effects of DEW from a client proxy perspective, we simulated an environment where many clients share a single proxy. We assume an *explicit* proxy deployment (see [12], Chapter 8) where all clients send all HTTP requests to the proxy with unresolved host names, and only the proxy interacts with the LDNS. We do not consider *transparent* proxy deployment because clients in this case perform their own DNS lookups and hence the DEW effects would be similar to those of Section 5.1.

We use the Larmancom and NLANR traces for this experiment, assuming that all requests from the trace are processed by a single proxy, simulate the state of a DEW-capable LDNS used by the proxy, and categorize requests accordingly. Merging three proxy traces into one makes our results on DEW participation conservative because this increases sharing of cached DNS responses among a greater number of clients. We use the same set of TTLs and thresholds as we did for the client evaluation as the simulation parameters.

Since the proxy is the only client of its LDNS in the simulation, the DNS cache at the proxy and at the LDNS are exactly synchronized. The resolution for a domain name expires at the same time for both. In this configuration, since we assume that both the DNS response and the HTTP object have the same TTL, cache hit or HTTP request categories of DEW participants can never occur. Indeed, an HTTP request occurs when requested host name is in LDNS's DNS cache but the object is not in its HTTP cache. However, proxy in this case will also have this host name in its DNS cache and will send the request directly to the Web server by HTTP. A cache hit occurs when LDNS caches both the requested host name and the object. The proxy in this case will also have a valid host name in its DNS cache and will interact with the Web server directly. Thus, in the proxy simulations, the only possible participants are DEW requests.

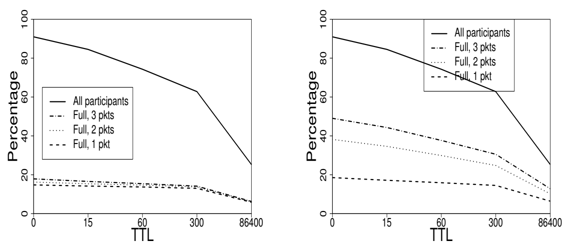
In the previous section we argued that reducing the delay for delivery of container pages provides the greatest benefit for clients. Therefore, for brevity, in this section and the next we examine the effect of the proxy-DEW configuration only for container pages, and also present results only without compression for Larmancom and with compression for NLANR. Figure 7a shows the fraction of DEW participants among requests for container pages in the Larmancom trace. Comparing to Figure 5b, we see fewer DEW participants, which is understandable because the proxy reuses locally cached DNS resolutions across all requests while clients can only reuse their own DNS resolutions. What is surprising is that for moderate TTLs the reduction is quite modest: for TTL of 60s, 18% of requests for container pages are still DEW participants, and 9% enjoy strict piggybacking. However, DEW participants all but disappear for TTL of one day.

Proxy results for the NLANR logs using 4:1 compression are shown in Figure 7b. These results are similar to those for the Larmancom logs, but show much higher percentages of DEW participants. For TTL of 60s, the participants approach 40% of all container page requests, with 20% fitting into a single packet for strict piggybacking. The numbers remain significant (around 20%) for TTL of 300s. While we could not use NLANR traces for studying the no-proxy environment because of scrambled client IP addresses, the number of participants in that case would be higher yet, as explained before.

5.3 Content Provider

Finally, we consider DEW performance from the point of view of the content provider. One of the major goals of the content provider is to provide the best possible performance to its clients. The simulation is similar to that of Section 5.1 except we use the Largesite logs and group clients around a shared LDNS server using precise client-LDNS maps as described in Section 4.

Again, in this simulation we focus solely on results for container pages, which were responsible for approximately 4%



(a) No compression (b) 4:1 compression

Figure 8: Percentage of DEW participants for container pages at server (Largesite logs).

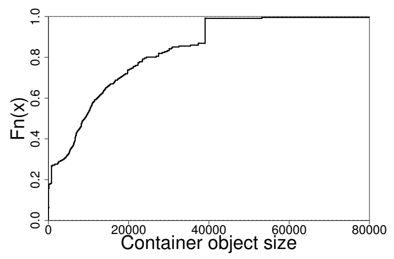


Figure 9: CDF of container page sizes (Largesite logs).

of the requests in the Largesite logs. The results, shown in Figure 8a, are very different from those of the previous experiments. First, the total number of DEW participants is significantly higher than in the previous experiments. Even with TTL of one day, when the greatest percentage of requests are masked from DEW at the LDNS server, 25% percent of container page requests participate. For TTL of 300s, a value more likely with a CDN-delivered Web site, the percentage of participants increases to over 60%.

Second, full participants represent a significantly smaller portion of all the participants. Furthermore, the number of full participants for different size thresholds are clustered together. To understand this result, we examined the CDF of sizes of requests for container pages in the Largesite logs, shown in Figure 9. This figure shows that relatively few – less than a third – of the container pages served by this particular server are below even the largest threshold we consider. Moreover, most of these pages are actually below the lowest threshold of one packet: the percentage fitting under the threshold grows only from 30 to 33% as the threshold increases from one to three packets.

However, considerably higher percentages of container pages—around 19%, 38%, and 50%—fall within these thresholds when a 4:1 compression ratio can be employed. The results of our simulations using 4:1 compression for the container pages are shown in Figure 8b. We find that for the Largesite logs, compression greatly increases the number of full DEW participants. Since full participants achieve the greatest benefits from DEW, we conclude that compression would be highly beneficial for this site.

6. ISSUES

In this section we discuss issues related to the deployment of a DEW system.

6.1 Implementation

Our goal is to use the existing protocols with minimal changes in such a way that DEW can be incrementally deployed. We achieve this in the following manner:

- The HTTP request can be embedded into the DNS request by using the OPCODE=Query in the DNS request header. The client would set the QDCOUNT to reflect the number entries in the query section, one of which is a new type of query, QTYPE=HTTP. The complete HTTP request header would then be embedded in the QNAME field. DNS servers that do not understand the HTTP QTYPE ignore this portion of the query and respond to the portion of the query that is understood. The client receives a normal response to the DEW query and proceed in a non-DEW fashion. This is used by the client when resolving a host name from its LDNS, and by the LDNS when sending requests to the remote DNS.

- When an LDNS server issues an HTTP request, it must indicate to the Web server that the response size should be below the size threshold and identify itself via the User-Agent request header that it is DEW-enabled. The LDNS server uses the Range request mechanism but is not aware of the precise size of the response header (although it is typically less than 200 bytes). The DEW-aware Web server would subtract the response header size from the Range limit specified and send the response that fits precisely into the piggybacked limit. Responses from DEW-unaware Web servers might slightly exceed this limit, causing the LDNS to discard the extra bytes.

- The client must know how many bytes are required to make up a received object. This is accomplished at the client by examining the `Content-Length` field of the response header. Note that if the `Content-Length` field is not present, the client can use the N- option in Range requests (e.g., `Range:1000-`) which will request all but the first N bytes.

- When a DNS response contains NS records referring the requester to other authoritative nameservers, the additional info section of the response carries DEW resource records (RRs). These records indicate which nameservers are DEW-capable. This is primarily intended to avoid unnecessary piggybacking of HTTP requests to DEW-incapable RDNS servers.

6.2 Packet Loss

Since DEW responses are piggybacked on DNS responses they are subject to packet loss. When the complete response can be contained in a single packet, loss of this packet results in no worse performance than it would without DEW; receipt of the DNS response is required before the data transfer can take place. When the piggybacked response contains more than one packet, the DNS response in the first packet is still sufficient for obtaining the rest of the content via a normal HTTP Range request to the Web server. Consequently, the LDNS handles a loss of a subsequent DEW response packet from RDNS by sending to the client a DEW response containing just the contiguous portion of HTTP data that was successfully received. The client handles packet loss by simply issuing the appropriate HTTP Range request to the Web server, using the N- option discussed above if necessary.

The remaining issue is how to decide when a loss occurred. Since there is no acknowledgment of data receipt in DEW, timeouts are the only option for loss detection. The timeout, however, can be quite short since the response packets are expected to be sent in an immediate burst, and should arrive within a very small time window at the client after the initial response packet. (Note that this timeout is different from the timeout in the DNS protocol that concerns retransmissions of the DNS requests.) Furthermore, a host (either an HTTP client communicating with its LDNS or the LDNS communicating with the RDNS) can avoid frequent timeouts during high-loss periods by specifying the size threshold of one packet in its DEW requests.

7. CONCLUSION

We proposed and evaluated a method to improve efficiency of Web browsing by piggybacking HTTP interactions on DNS responses. Such piggybacking promises improved responsiveness of Web browsing (our focus in this paper) as well as reduced network traffic due to fewer TCP control messages. We designed our scheme, called DEW for DNS-enhanced Web, conservatively with the goals of (a) limiting piggybacking only to existing DNS messages generated by the current Web anyway, (b) using only allowed optional fields in DNS and HTTP protocols for our protocol extensions, (c) allowing coexistence with current Web including compatibility with Network Address Translation (NAT [15]) and hence incremental deployment of DEW, (d) being non-intrusive in terms of Internet congestion control, and (e) coping with the best-effort nature of DNS messages.

We evaluated the frequency with which the opportunity for piggybacking arises in several environments: when clients access the Web without a proxy, when a proxy aggregates accesses from a large number of clients, and from the point of view of a large Web content provider. We found that, overall, DNS messages offer reasonable opportunity for HTTP piggybacking. Not surprisingly, this opportunity decreases with increased time-to-live of cached DNS responses, so DEW should be especially useful for content delivered through content delivery networks because they use very short TTLs. But even with TTL of a full day, when our proxy traces exhibited small DEW benefits, our content provider trace showed that 25% of full-document (container page plus embedded objects) downloads would still benefit from DEW. Furthermore, even the proxy evaluation, which we expected to show very small opportunity for piggybacking due to extensive DNS caching at the proxy, showed respectable piggybacking frequency for NLANR trace (over 20% of full-document downloads with TTL of 5 minutes).

Another interesting observation is that most of the DEW benefits come from piggybacking HTTP all the way from authoritative DNS servers to client DNS servers to HTTP clients, rather than from delivering piggybacked HTTP content cached by client DNS or obtained by client DNS from the Web over HTTP. Thus, most DEW benefits can be had with a simplified DEW architecture where client DNS servers have no added HTTP caching and retrieval functionality.

Our future work on the DEW project includes investigation of DEW benefits on CDN traces and on traces from more content providers. Studying CDN traces would be especially interesting because we expect DEW to be most useful for CDN-delivered content. We would also like to quantify DEW's effect on network traffic reduction and to prototype DEW to be able to directly evaluate its improvements in Web browsing responsiveness.

Acknowledgements

We would like to thank Fred Douglis and Ramón Cáceres for invigorating discussions on DEW and Chuck Kalmanek, Walter Sturm, and anonymous referees for useful comments on the preliminary version of the paper.

8. REFERENCES

[1] H. Chen and P. Mohapatra. Catp: A context-aware transportation protocol for http. Technical Report CSE-2002-19, Department of Computer Science, UC Davis, 2002.

[2] Israel Cidon, Raphael Rom, Amit Gupta, and Christoph Schuba. Hybrid TCP-UDP transport for Web traffic. In *IEEE Int'l Performance, Computing and Communications Conference*, 1999.

[3] Edith Cohen, Balachander Krishnamurthy, and Jennifer Rexford. Improving End-to-End Performance of the Web Using Server Volumes and Proxy Filters. In *Proc. ACM SIGCOMM*, September 1998. `http://www.acm.org/sigcomm/sigcomm98/tp/abs_20.html`.

[4] Krishna Gummadi, Stefan Saroiu, and Steven D. Gribble. King: Estimating latencies between arbitrary internet end hosts. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[5] Jaeyeon Jung, Emil Sit, Hari Balakrishnan and Robert Morris. DNS Performance and the Effectiveness of Caching. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop 2001*, November 2001.

[6] Balachander Krishnamurthy and Jennifer Rexford. *Web Protocols and Practice: HTTP/1.1, Networking Protocols, Caching, and Traffic Measurement*. Addison-Wesley, May 2001. ISBN 0-201-710889-0.

[7] Balachander Krishnamurthy and Craig Wills. Study of Piggyback Cache Validation for Proxy Caches in the World Wide Web. In *Proc. USENIX Symposium on Internet Technologies and Systems*, pages 1–12, December 1997. `http://www.research.att.com/~bala/papers/pcv-usits97.ps.gz`.

[8] Zhuoqing Morley Mao, Charles D. Cranor, Fred Douglis, Michael Rabinovich, Oliver Spatscheck, and Jia Wang. A precise and efficient evaluation of the

proximity between web clients and their local DNS servers. In *Proceedings of USENIX 2002*, June 2002.

[9] Media metrix. `http://www.mediametrix.com`.

[10] P. Mockapetris. RFC 1035: Domain names - implementation and specification, November 1987.

[11] Jeffrey Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta-encoding and data compression for HTTP. In *Proceedings of ACM SIGCOMM'97 Conference*, pages 181–194, September 1997.

[12] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2001.

[13] Michael Rabinovich and Hua Wang. DHTTP: An efficient and Cache-Friendly transfer protocol for web traffic. In *INFOCOM'01*, pages 1597–1606, April 22–26 2001.

[14] Squid internet object cache. `http://www.squid-cache.org`.

[15] P. Srisuresh and M. Holdrege. IP Network Address Translator (NAT) Terminology and Considerations, August 1999.

[16] P. Vixie. RFC 2671: Extension Mechanisms for DNS (EDNS0), August 1999.