Class-based Delta-encoding: A Scalable Scheme for Caching Dynamic Web Content

Konstantinos Psounis
Department of Electrical Engineering
Stanford University, Stanford, CA 94305
kpsounis@leland.stanford.edu

Abstract—**Caching static HTTP traffic in proxy-caches has reduced bandwidth consumption and download latency. However, web-caching performance is hard to increase further due to the growing number of non-cachable dynamic web-documents. Delta-encoding is a promising technique that exploits temporal correlation among different snapshots of a dynamic document, and renders dynamic traffic cachable. It achieves this by combining a cachable, previous snapshot of a document, called base-file, with a small difference-file, called delta, to generate the current snapshot of the document. However, it has not yet been deployed due to the significant scalability concerns related to the storage requirements for base-files on the server-side.**

In this paper we introduce class-based delta-encoding, a scalable scheme to perform delta-encoding on dynamic web-traffic. The idea is to group documents into classes, and store one document per class on the server-side. Thus, the proposed scheme exploits both temporal correlation in a dynamically evolving document, and spatial correlation among different documents. Finally, we present an architecture to deploy the scheme, that is transparent to clients, proxy-caches, and web-servers. Experimental results report that class-based delta-encoding combined with compression reduces the bandwidth consumption by a factor of 30, and the latency perceived by most users by a factor of 10 on average, without suffering from enormous storage requirements on the server-side.

Keywords—**web-caching, dynamic document, delta-encoding, delta, base-file, grouping, privacy.**

I. Introduction

The use of web-caches for reducing bandwidth consumption and download latency has been quite successful in the past. However, traditional web-caching is applicable only to static documents, or to documents that change in large timescales. Since the proportion of dynamic versus static documents is increasing day by day, current caching solutions have reached a point where their performance cannot be significantly improved unless they incorporate a mechanism to "cache" dynamic documents.

In particular, no matter the replacement scheme, the cache size and the user population serviced by the cache, proxy-cache hit rates are usually around 40% [18]. However, if proxy-caches were equipped with mechanisms that exploit redundancy from all documents, static and dynamic, hit rates could have been up to 80% [18]. Another study [17] reports that after proxy-caching has been applied, an additional 40% of web-traffic is found to be redundant.

Recently, various methods to exploit the redundancy of dynamic traffic have been proposed. Delta-encoding was introduced by Banga et al. [1], and independently in a more restricted context in [8]. The idea is that both ends of a slow link store the same snapshot of a web-document, to be called *base-file* onwards, and upon a request for that document: (i) the end towards the server gets the current snapshot of the document from the web-server, (ii) it computes the difference, to be called *delta* onwards, between the current and the stored snapshot, (iii) it sends the delta over the slow link to the other end towards the client, and (iv) the end towards the client reconstructs the current snapshot by combining the delta and the stored snapshot, and (v) it sends the response to the client. Mogul et al. [13] use dynamic traces and show that delta-encoding can provide remarkable improvements in response size and response delay for an important subset of HTTP content types.

Douglis et al. [6] take an application-specific view, in which they separate the static and dynamic portions of a document. Static parts are cached as usual, while dynamic parts are obtained on each access from the server. They also provide an HTML extension to support this scheme. According to their simulations, the size of network transfers are typically 2 to 8 times smaller than the original sizes. This idea is simpler than delta-encoding, but it is less efficient. Clearly, delta-encoding exploits more redundancy than this scheme. Also, it is feasible to deploy delta-encoding without requiring HTML or other support from the existing infrastructure.

Another interesting work, by Cao et al., is Active Cache [4]. The Active Cache scheme allows servers to supply cache applets to be attached with documents, and requires proxies to invoke cache applets upon cache hits to furnish the necessary processing without contacting the server. Even though this approach is efficient for tasks like rotating advertisements, it is not efficient as a general mechanism that fully exploits temporal correlation in dynamic traffic. There are situations where one cannot avoid direct communication with the server that hosts the current snapshot of a document. Similar ideas with [4] are also used in [14] where caches dynamically generate content for clients by running code provided by publishers.

Finally, Smith at al. [16] propose a new protocol to allow individual content-generating applications exploit query semantics and specify how their results should be cached and delivered. Applications may declare a dynamic request to be identical, equivalent, or partially equivalent. In the first two cases cached results are up to date, while in the third case content can be immediately delivered as an approximate solution while actual content is generated and delivered. This is not a complete solution to the problem of exploiting temporal correlation in dynamic traffic; there exist dynamic documents that share many data, yet their redundancy cannot be exploited with this simple scheme.

In another direction, there has been a lot of research activity in the area of precomputing and handling dynamic traff

This work has been done while working part time at FineGround Networks, Inc.

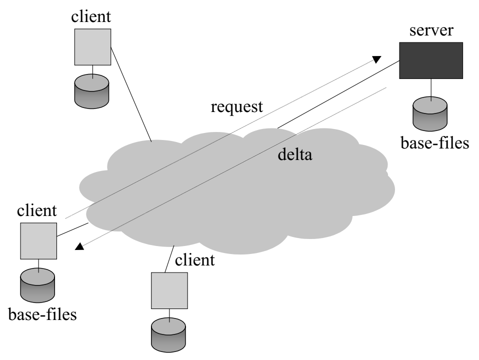

Fig. 1. Using delta-encoding for web-documents.

server-side [5], [7], [10], [19]. The idea is to offload servers from computing dynamic documents on the fly. These schemes reduce user latency only when the bottleneck is server's CPU, and do not reduce bandwidth consumption. However, they can be deployed in conjunction to delta-encoding mechanisms, since they are orthogonal to the previous schemes.

Delta-encoding could provide an efficient solution to the problem of exploiting temporal correlation in dynamic traffic. However, the basic delta-encoding scheme suffers from scalability problems at the server-side. In particular, the storage requirements at the server-side grow enormously due to the increasing number of dynamic documents. This problem becomes even more intense due to personalized web-documents, since the scheme requires to store many personalized versions of each dynamic document.

In this paper we introduce class-based delta-encoding, a scalable scheme to perform delta-encoding on dynamic web-traffic. The idea is to group documents into classes, and store one document per class on the server-side. In essence, in addition to exploiting temporal correlation in a dynamically evolving document, we also exploit spatial correlation among different documents.

The outline of the paper is as follows: in the next section we describe the class-based delta-encoding scheme. In Section III we elaborate on how to group requests into classes. In Section IV we propose an online algorithm to choose a base-file for each class. In Section V we address the security issues raised by associating with every class a shared base-file. Finally, in Section VI we present simulation results to evaluate the efficiency of class-based delta-encoding, and comment on implementation issues.

II. Class-Based Delta-Encoding

Delta-encoding is the process of generating a difference file, called *delta*, between two files with the following two properties: (i) the combination of the delta and one of the files, called *base-file*, suffices to reproduce the other file, and (ii) the size of the delta is as small as possible. The delta-generation and the file-reconstruction phases should also require as less computation time as possible to execute. For a thorough discussion on delta-encoding algorithms, see [9].

In the context of HTTP, delta-encoding can be used to exploit *temporal* correlations between consecutive snapshots of a dynamic web-document. As shown in Figure 1, the client and the server, for example an end-user and a content-provider respectively, share a common base-file which is a snapshot of the dynamic document at some point in time. Whenever the client requests the document from the server, the server computes the delta between the current snapshot of the document and the base-file, and sends the delta to the client. Upon receipt of the delta, the client computes the current snapshot of the document by combining the delta and the locally stored base-file.

In section VI we show that this scheme is very efficient in reducing bandwidth consumption and user latency. We also show that the computation requirements to calculate the deltas on the server side are reasonable. However, this simple approach is not scalable due to the enormous storage requirements on the server-side, since some web-servers have too many dynamic documents. The problem becomes even more intense with personalized web-documents. When documents are personalized (ex: my.yahoo.com) the server must store for each document many base-files corresponding to each user.

The solution to the scalability problem is to exploit *spatial* correlation, i.e. correlation between different documents, in addition to *temporal* correlation, i.e. correlation between different snapshots of the same document. This leads us to introduce *class-based* delta-encoding. Under class-based delta-encoding, dynamic documents are grouped into classes, and a single base-file is stored at the server per class.

There are three issues related to class-based delta-encoding that we address in this paper:

- How to design an automated mechanism that groups documents into classes, based on the proximity of their content.
- How to design an efficient online algorithm that identifies a good base-file for each class.
- How to design a mechanism that removes private information from the base-file of each class. This is important since the base-file of a class is stored locally by many clients.

III. Grouping Documents into Classes

In this section we describe a mechanism to group documents into classes. The mechanism aims to quickly identify a good class for each document. A class is good if the size, in bytes, of the delta between the base-file of the class and the document is small.

Call a *delta-server*, an engine that implements class-based delta-encoding and services the contents of some web-servers. Since every document corresponds to a URL-request, we group documents by grouping their corresponding URL-requests.

All requests are processed by the delta-server before they are forwarded to the web-servers. Initially, there are no classes formed in the delta-server. Whenever an ungrouped URL-request arrives at the delta-server, the scheme groups it into an existing class, or creates a new class. As the number of classes grows, it is not practicable to perform an exhaustive search among all existing classes to identify a good class. We shall use as a search-hint the observation that a similarity between two URLs is an indication of a similarity between their corresponding contents, and perform a search over a small and appropriate subset of the existing classes.

URL	hint-part	rest
www.foo.com/laptops?id=100	laptops	id=100
www.foo.com/?dept=laptops&id=100	dept=laptops	id=100
www.foo.com/laptops/100	laptops	100

TABLE I
URL-PARTS FOR DIFFERENTLY ORGANIZED WEB-SITES.

Partition the URLs in three parts, the server-part, the hint-part, and the rest. The server-part is the string from the beginning of the URL till the first slash, as usual. The portion of the URL that is used as the hint-part differs among web-servers and depends on how the web-server organizes its content. For example, let www.foo.com be a web site that sells computers, laptops and desktops. Assume that documents corresponding to laptops are similar, while they differ from documents corresponding to desktops. Table I shows the hint-part for three different cases. Depending on the web-site, the administrator describes to the grouping mechanism how to partition URLs into parts using regular expressions. Then, the mechanism uses these parts to expedite the grouping process.

We now describe the grouping mechanism. A matching is said to occur if the delta between the requested document and the base-file of the class is smaller than a threshold. Since it is very unlikely that two documents originating from different servers will be close enough to join the same class, a new class is created in case there are no classes with members whose server-part is the same with the request's server-part. Else, some heuristics are used to tradeoff between search-time and matching-quality:

- If some classes have members whose hint-parts are the same with the request's hint-part, the mechanism only considers those as potential classes to group the request.
- The mechanism never considers more than N existing classes as potential classes to group a request. If no matching is found after N tries, a new class is created.
- The mechanism first attempts to group the request into classes with many members, and then into less popular classes. In particular, the first $a \cdot N$ tries consist of the most popular classes, and the last $(1-a) \cdot N$ consist of random selections among the rest of the eligible classes [1].
- Since for grouping purposes it is not required to generate a precise delta between the requested document and the base-file of a candidate class, but rather to estimate how close they are, a light version of the delta algorithm[2] is used to reduce computation cost.

We described an automated mechanism to group documents into classes. Sometimes, this mechanism is not efficient, for example, for web-sites that are organized in a completely ad-hoc manner. In this case, the administrator has the option to manually group URLs into classes.

IV. Choosing a Base-file

Once classes are formed, it is required to identify a good base-file for each one of them. The simplest scheme would set as a base-file the document corresponding to the request creating the class. From a performance point of view, the best scheme would construct an artificial base-file consisting of all those document-parts that are popular among the members of each class. However, this is too expensive. A reasonable compromise is to only consider entire documents as potential base-file candidates, and choose the best among those. Despite its relative simplicity, this choice performs well in practice.

Ideally, if all future requests $r_1, r_2, \ldots, r_n$ where known in advance, an offline algorithm would choose as a base-file the document that minimizes the sum of deltas between itself and every other document. The exhaustive online algorithm that considers all documents seen so far as base-file candidates is not practicable due to memory and computational constraints. Instead, we propose the following scheme:

1. Sample each request with probability p i.e. consider the corresponding document a base-file candidate and store it in the memory.
2. Use as a base-file the best of the stored documents i.e. the document that minimizes the sum of deltas between itself and all the other stored documents.
3. Store up to K documents. Thus, after acquiring K samples, whenever a new sample is drawn evict the document that maximizes the sum of deltas [3].

By design, the algorithm stores good base-file candidates and uses the best out of those as a base-file. Call a rebase the process of changing a base-file. After a rebase, the new base-file should be distributed to all clients before they can benefit from delta-encoding. To control the number of rebases, a rebase takes place if both a better base-file candidate exists, and a rebase-timeout, since the previous rebase, has expired.

Rebases caused by this algorithm are called group-rebases to distinguish them from basic-rebases that are triggered when the generated deltas are relatively large. When a basic-rebase takes place, all K stored documents are flushed. Group-rebase and basic-rebase are orthogonal operations.

The computational requirements of this algorithm are reasonable. Indeed, whenever a new sample is acquired, the algorithm calculates only the deltas between the corresponding document and the rest of the stored documents, and this calculation can be done *offline*. The memory requirements are also low since values of K around 10 are enough.

In Section VI we evaluate the performance of the algorithm in practice. Here, we perform some simple analytic calculations.

[1] Here it is implied that the procedure stops as soon as there is one matching. Another option is to always consider N classes and choose the best matching. The former is preferable since in practice it is more critical to reduce search-time. Typical N values are less than 10.

[2] A specific implementation of a delta-encoding algorithm is *Vdelta*[9]. *Vdelta* uses a hash table approach with enough indexes into the base-file for fast string matching. Each index is a position which is keyed by the four bytes starting at that position. Thus, the file is partitioned in four-byte-chunks. Further, in order to identify the maximally long matching prefix, the algorithm traverses the file both forwards and backwards. We use a light version of this algorithm that uses larger byte-chunks and only traverses the file in the forward direction.

[3] To avoid storing K documents that are very close to each other but not close to most of the members of the class, at periodic intervals we evict a random sample among the stored documents excluding the current base-file, instead of evicting the document that maximizes the sum of deltas (worst document). Another option is to maintain two sets of K documents: one holding the base-file candidates and one holding K random samples against which deltas are calculated for each base-file candidate. At eviction times, the worst document is evicted from the candidates' set, and a random sample is evicted from the other set. Both options work well in practice.

Assume there is a sequence of requests for distinct documents that are members of a class. We shall calculate the probability that the proposed scheme will discard the best base-file candidate. Let R be the total number of requests. Then, the expected number of base-file candidates is $N = R \cdot p$. Let S_N denote the set of all base-file candidates, and S_K denote the set of all K stored documents at some point in time. Finally, let d_{ij} denote the delta between documents i and j.

Associate with every base-file candidate i a global utility value, equal to $\sum_{j \in S_N} d_{ij}$. Index the documents in increasing order of their global utility value, i.e. document 1 minimizes $\sum d_{ij}$. Associate with every stored document i a local utility value, equal to $\sum_{j \in S_K} d_{ij}$.

Let i_1, i_2 be two documents such that $i_1 < i_2$, i.e. $\sum_{j \in S_N} d_{i_1 j} < \sum_{j \in S_N} d_{i_2 j}$. Suppose that the probability of the event $\{\sum_{j \in S_K} d_{i_1 j} > \sum_{j \in S_K} d_{i_2 j}\}$ equals $c/|i_1 - i_2|$ where c is a normalizing constant such that $c \cdot \sum_{i=1}^{N-1} \frac{1}{i} = 1$. This event corresponds to the situation where document i_1 is a better base-file candidate than document i_2, but the algorithm believes the opposite [4]. The best candidate, document 1, can only be evicted if $\sum_{j \in S_K} d_{1j} > \sum_{j \in S_K} d_{ij}$ for all other stored documents $i \in S_K$. In this case the algorithm will make an error.

The probability of error at an eviction time is at most $c^{K-1}/(K-1)!$. This probability corresponds to the event where the K best base-file candidates are stored and $\sum_{j \in S_K} d_{1j} > \sum_{j \in S_K} d_{ij}$ for $i = 2 \ldots K$. Thus, the probability of keeping the best base-file candidate throughout the process is at least $\left(1 - \frac{c^{K-1}}{(K-1)!}\right)^{N-K} \approx 1 - (N-K)\frac{c^{K-1}}{(K-1)!}$, and since $c \approx 1/\ln(N-1) \approx 1/\ln N$, the probability of error is bounded above by

$$P_{error} \leq \frac{(N-K)}{(\ln N)^{K-1}(K-1)!},$$

which is very small for all practical cases. For example, for $R = 10^5$, $p = 10^{-2}$, and $K = 10$, $N = 1000$ and $P_{error} \leq 8 \cdot 10^{-11}$.

V. Privacy Concerns

Class-based delta-encoding raises privacy concerns since many clients share the same base-files. In particular, each base-file is stored locally at more than one client. It is therefore critical to remove private information, for example credit card numbers, from shared base-files.

Before we present a mechanism that removes private information from base-files, it is useful to elaborate on a specific implementation of delta-encoding called *Vdelta* [9]. *Vdelta* uses a hash table approach with enough indices into the base-file for fast string matching. Each index is a position which is keyed by the four bytes starting at that position. The algorithm compares four-byte-chunks from the base-file and the requested document and finds maximally long matchings between the two documents. Some byte-chunks are common, while others are not present in both documents.

A simple but efficient mechanism to identify and remove private information from base-files is the following: (i) execute delta-encoding between the base-file and N other documents that are members of the class, (ii) record for each four-byte chunk of the base-file how many times it was common between the base-file and another document, and (iii) remove from the base-file byte-chunks that were not common between the base-file and any of the N other documents. We call this the *anonymization* process. The rationale behind this scheme is that private information is unique among users. Thus, by making sure that during the anonymization process the base-file is compared against documents corresponding to different users, any private information will be removed. Also, by doing the comparison with more than one document, the process does not remove potentially useful information from the base-file.

Note that until the base-file is properly anonymized, it should not be distributed to clients. Thus, delta-encoding should not be used until the anonymization process terminates. However, if there is already an anonymized base-file and a rebase is triggered, the previous base-file can be used until the new one is properly anonymized. Therefore, the performance penalty is not significant. Finally, the anonymization process can take place concurrently with the mechanism for finding a base-file, since both require the computation of deltas between some documents that are members of the same class.

A point of concern is that the same user may be falsely regarded by the server as a different user. Indeed, the standard way to distinguish users is by distributing to them user identifications through cookies. However, for example, Netscape and Internet Explorer do not share cookies and thus user identifications. As a result, if a user opens one Netscape and one Internet Explorer session at the same time and uses his/her credit card to make two transactions, the system will interpret these transactions as originating from different users.

Another point of concern is that sometimes private information is shared among a small set of different users, for example corporate credit card numbers. If two employees use a shared corporate credit card to make a transaction at the same website, while base-file anonymization is in progress for the class in which the corresponding requests belong to, the credit card number is at risk.

Even though the odds of these situations occurring are small, we wish to protect private data from such cases. To solve this problem, we introduce a second parameter M, and only include in the anonymized based-file byte-chunks that are common between the base-file and at least M documents. $M = 0$ corresponds to no privacy, $M = 1$ corresponds to the basic anonymization scheme, and larger values of $M \leq N$ increase privacy. Values of M close to N significantly reduce the size of the base-file and thus compromise performance since deltas tend to be large. A rule of thumb is that N should be at least twice as large as M.

The steps of the algorithm are as follows:

1. Choose a base-file.
2. Associate with each byte-chunk of the base-file one counter to record the number of times that each byte-chunk is common between the base-file and another document.
3. For the next N requests that belong to the specific class and originate from distinct users[5], increment the counters of the byte-

[4]The probability expression $c/|i_1 - i_2|$ is used as an example. Its rationale is that if $i_1 < i_2$ ($i_1 > i_2$), the smaller the distance between the indices of the two documents, the more probable for the algorithm to believe $i_1 > i_2$ ($i_1 < i_2$).

[5]These users should also be different from the user that corresponds to the base-file.

chunks that are common between the base-file and the requested document.
4. Remove all byte-chunks with counter values less than M.

We perform some elementary calculations to estimate how probable it is to accidentally include in an anonymized based-file private information. The exercise consists of starting with a non-anonymized base-file, drawing N documents out of which X may share private information with the base-file, and calculating the probability of the event $\{X \geq M\}$. Let X_i, $i = 1 \ldots N$, be a random variable equal to 1 if the i^{th} sample shares private information with the base-file, and zero otherwise. Then, $X = \sum_{i=1}^{N} X_i$. Assume the X_i's are i.i.d random variables equal to one with probability p and zero otherwise. Then, X is binomially distributed with parameters N and p, and the probability of error, $P(X \geq M)$, equals $\sum_{i=M}^{N} \frac{N!}{i!(N-i)!} p^i (1-p)^{N-i}$. Assuming $P(X \geq M) \approx P(X = M)$ [6] and $(1-p)^{N-M} \approx 1$, the probability of error is bounded above by

$$P_{error} \leq (Ne/M)^M p^M.$$

For $p = 0.01$, $N = 10$ and $M = 5$, the upper bound equals $4.7 \cdot 10^{-7}$ while the exact probability of error equals $2.4 \cdot 10^{-8}$.

In the preceding analysis we assumed that the X_i's are i.i.d. However, it is more likely that $P(X_2 = 1|X_1 = 1) < P(X_2 = 1|X_1 = 0)$. In particular, if p_j is the probability of the j^{th} occurrence of a document sharing private information with the base-file, p_j is expected to be a decreasing function of j. As an example of a decreasing function of j, suppose $p_j = p^j$. Recall that for an error to occur, there should be at least M out of N documents sharing private data with the base-file. Since the probability of getting the $(M+1)^{th}$ such document is very low, we can ignore the cases where $X > M$ and only compute the probability of the event $\{X = M\}$. Since $P(X = M) \leq \frac{N!}{M!(N-M)!} \cdot p \cdot p^2 \cdots p^M \cdot (1-p^M)^{N-M}$, and $(1-p^M)^{N-M} \approx 1$, the probability of error is bounded above by

$$P_{error} \leq (Ne/M)^M p^{M(M+1)/2}.$$

VI. Simulation Results and Deployment Issues

In this section we present simulation results to evaluate the performance of class-based delta-encoding. First, we report bandwidth savings achieved by delta-encoding using real traces, and convert bandwidth into latency savings. Second, we evaluate the performance of the class-related operations, namely grouping, choosing base-files, and anonymizing base-files. Finally, we propose an architecture to deploy class-based delta-encoding, and report experimental results from a specific implementation of that architecture.

A. Delta-Encoding Performance

Delta-encoding is very efficient in reducing bandwidth consumption. Using traces from commercial web-sites, we calculate the total outbound traffic when delta-encoding and compression of the generated deltas is used, and compare it to the total direct outbound traffic. Notice that the access-logs of web-sites represent HTTP requests after any proxy-caches, and thus correspond to traditionally uncachable traffic. Table II shows the results obtained by three commercial web-sites[7]. From this table we can see that delta-encoding combined with compression[8], reduces bandwidth consumption by at least a factor of 20, and sometimes by a factor of 30 or more. In these numbers a factor of 2 on average is thanks to compression. These result are in accordance with the previous studies [13] and [15]. Notice that the experimental results reported here correspond to actual savings when using an implementation of the architecture presented in Section VI-C. In contrast, the previous studies assume the existence of a delta-encoding mechanism and report projected savings by applying delta-encoding on the server-documents.

Web sites	Total requests	Direct KB	Delta KB	Savings
1	16407	736495	38308	94.8%
2	1476	49536	2474	95.0%
3	7460	230840	6640	97.1%

TABLE II
BANDWIDTH SAVINGS USING ACCESS-LOGS FROM THREE COMMERCIAL WEB-SITES.

Bandwidth savings are important because they offload the network, save money, and decrease the delay perceived by users. However, since delay is what matters most to end-users, it is interesting to investigate the relationship between bandwidth and latency savings.

Web-documents that benefit significantly from delta-encoding have an average size around 30 to 50KB, as observed from access-logs-driven simulations. Gzipped deltas tend to be 1 to 3KB. Without loss of generality, suppose that two documents of size $S_1 = 30KB$ and $S_2 = 1KB$ are sent from a server to a client using TCP. Let L_1 and L_2 be the user-latency corresponding to document 1 and 2 respectively. Given that $S_1/S_2 = 30$, we wish to estimate L_1/L_2.

Assume the transmission takes place over a high-bandwidth connection. Due to TCP slow-start the number of round-trip times (RTTs) required to send document 1 is roughly $\log S_1/S_2$ times the number of RTTs to send document 2. This can be easily verified by taking into account the additive increase of the TCP-window when counting RTTs in both cases. Thus, L_1/L_2 is roughly equal to 5. The latency gain from delta-encoding is further increased when the client is connected through a low-bandwidth connection. Without loss of generality assume the connection is through a 56Kb/s modem with 100ms RTT. The transmission time of a single packet is roughly equal to twice RTT, thus the advantage of larger TCP-windows is reduced since transmission time is now the bottleneck. In this case L_1/L_2 is a linear function of S_1/S_2. Due to the cost associated with setting up a TCP connection, the queueing delay, the timeouts and the retransmissions caused by packet losses, L_1/L_2 is estimated to be around 10. More information on how to derive such estimations can be found in [2]. Using measurement tools [22], we measured L_1 and L_2 and verified the estimated values for L_1/L_2 in case of high- and low-bandwidth connections.

[6] p^i decreases a lot faster than $\frac{N!}{i!(N-i)!}$ increases as a function of $i < N/2$, for typical values of p and N.

[7] Due to privacy concerns, we are unable to provide the URLs of the three sites.

[8] Deltas are compressed using gzip.

Permutations	First Response	Randomized	Online Optimal
1	1704	1559	1406
2	1774	1636	1540
3	1785	1599	1515
4	1876	1626	1542
5	2025	1679	1575

TABLE III
AVERAGE DELTA SIZES, IN BYTES, RESULTING FROM VARIOUS ALGORITHMS THAT IDENTIFY BASE-FILES FOR CLASSES.

M	N	Base (plain)	Base (anon)	Delta (plain)	Delta (anon)
2	5	84213	73434	5224	6520
4	12	84213	72714	5224	6097
4	8	84213	71090	5224	6505

TABLE IV
BASE-FILE AND DELTA SIZES, IN BYTES, FOR VARIOUS ANONYMIZATION (ANON) LEVELS. DELTAS REPRESENT AVERAGE VALUES AMONG A LARGE POOL OF DOCUMENTS.

Delta-encoding is a CPU-intensive process since it involves compressing one file using the contents of another file. However, there are quite efficient algorithms, like Vdelta [9], to perform this task. In Section VI-C we report experimental results which show that the CPU overhead associated with delta-encoding is reasonable.

B. Performance of Class-Based Operations

We have implemented the grouping, choosing-a-base-file, and anonymization functionalities, and integrated them with the delta-encoding routines to evaluate their performance.

The grouping mechanism aims to find a good class for all documents as fast as possible. The automated mechanism introduced in Section III, when used against a well-structured website, groups requests in classes after a couple of tries, provided that proper regular expressions are given to the scheme to partition the URLs into classes. Further, for the traces used in Table II, the number of produced groups are between 10 and 100 times less than the number of dynamic documents. No noticeable reduction on the bandwidth and latency savings is observed.

To evaluate the performance of the proposed algorithm for choosing base-files, we use various algorithms to choose a base-file, and calculate the average delta-size obtained over the same sequence of requests for each of the algorithms. In particular, we compare the scheme that uses the first response as a base-file, to the randomized online algorithm proposed in Section IV, and to the online optimal algorithm that uses as a base-file the one that minimizes the average delta so far. In Table III we present these results for five different random permutations of our sequence. The randomized algorithm uses a total of 8 samples, and a probability of choosing a sample as a base-file candidate equal to 0.2. It performs close to the optimal, and both of them are better than the scheme that uses the first response as a base-file. In general, depending on the web-site and the request sequence, the performance of the scheme that uses the first response as a base-file can be very bad, which is never the case for the randomized algorithm, since the later adapts to the characteristics of the request sequence.

Anonymization is required to provide privacy. The downside is that base-files tend to decrease in size and produce larger deltas. In Table IV we present the size of base-files before and after anonymization for various anonymization levels. Average sizes of the corresponding deltas with and without anonymazation are also presented. Even when high anonymization levels are used, deltas tend to increase by a very small amount. Thus, anonymization is achieved at a minimal cost.

Even though anonymization increases delta sizes, class-based delta-encoding with anonymization may improve bandwidth savings in comparison to classless delta-encoding. Indeed, anonymized based files are cachable, and thus many different users will download the same base-files from a proxy-cache. Since base-files are a lot larger than deltas, the gain from cachable base-files is expected to be larger than the loss from slightly larger deltas.

C. Architecture and Deployment

We have established that class-based delta-encoding is a scalable scheme that significantly reduces bandwidth consumption and user latency. However, in order to be widely deployed, it should require no changes to the current web-architecture. Web-servers, proxy-caches, and clients' browsers should be able to interoperate with the class-based delta-encoding functionality without any changes. We present here one architecture that satisfies this requirement. More information about this architecture can be found at [20].

On the server side, there should be a delta-server responsible to generate the deltas and store all base-files. The delta-server should be as close to the web-server as possible, to minimize delays due to the transmission of current document snapshots from the web-server to the delta-server. The obvious choice is to place the delta-server next to the web-server. Recall that class-based delta-encoding groups similar documents and personalized versions of the same document in one class, and uses a single base-file per class. Thus, the storage requirements on the delta-server are reasonable.

On the client-side, one may use the browser's cache to store base-files, and rely on Java-scripts, enabled at the browser, to combine deltas and locally stored base-files in order to compute the current document snapshot. Another option is to use specialized plug-ins. Notice that base-file anonymization guarantees that shared base-files do not contain private information and cannot be associated with any particular user.

Base-files can be marked cachable. Thus, proxy-caches can cache them as usual, resulting in the known benefits of proxy-caching.

Therefore, class-based delta-encoding can be deployed transparently to clients, proxy-caches, and web-servers, with the only necessary addition being a delta-server on the server-side, as shown in Figure 2.

A specific realization of the architecture shown in Figure 2 has been implemented and tested. We use this implementation to comment on the CPU overhead of such a mechanism. The numbers reported correspond to a PC configured with a Pentium III at 866MHz and 512MB of RAM, where the delta-se

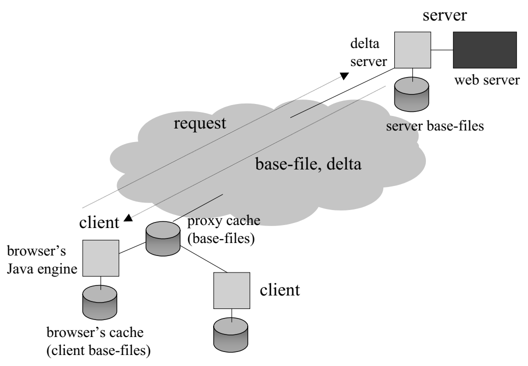

Fig. 2. Transparent deployment of class-based delta-encoding.

been integrated to an Apache 1.3.17 [21] web-server, running under Linux 2.2.19.

A plain Apache web-server is measured to have a capacity of 175 to 180 requests per second, with a maximum number of concurrent connections equal to 255. We will compare these numbers to the respective numbers achieved by the system of the delta- and web-server.

For a base-file of around 50 to 60KB, and a generated delta of around 8KB (uncompressed) or 3KB (compressed), the delta generation process requires 6 to 8ms. This latency is insignificant for a client[9]. However, the delta generation is a CPU-intensive process. As a result, the capacity of the system of the delta-server together with the web-server is around 130 requests per second. Even though the system of the delta- and web-server has lower capacity than a stand-alone web-server, it can sustain a higher number of concurrent connections. In particular, thanks to an offloading effect on the web-server from the delta-server, the system is possible to sustain 500 or more concurrent connections.

From the above we conclude that the CPU overhead of the delta-encoding mechanism is reasonable. In addition, CPU is cheap in comparison to the cost of access links. Thus, in practice it is very common that the bottleneck resource at a web-server is the access link out of the web-site and not the CPU. This further reduces the significance of the CPU overhead.

VII. Conclusions and Future Work

In this paper we introduced class-based delta-encoding, a practicable scheme for rendering dynamic traffic cachable. By combining the basic delta-encoding technique with the idea to group documents into classes, our scheme benefits from both temporal and spatial correlations among dynamic traffic. As a result, class-based delta-encoding significantly reduces bandwidth consumption and latency perceived by users, without suffering from impracticable storage requirements on the server-side. Further, we presented how to implement the proposed scheme in a secure and transparent way.

We intend to perform more simulations using real data from various web-sites, in order to understand better the robustness and performance of the class-related operations.

[9]Insignificant is also the latency associated with the document reconstruction on the client side.

VIII. Acknowledgments

I would like to thank Nat Kausik for exposing me to the transparent delta-encoding architecture presented in Section VI-C, and Jay Jawahar for fruitful discussions about class-based delta-encoding.

References

[1] Gaurav Banga, Fred Douglis and Michael Rabinovich, "Optimistic Deltas for WWW Latency Reduction", In proceedings of *USENIX Technical Conference*, '97.

[2] Paul Barford and Mark Crovella, "Critical Path Analysis of TCP Transactions", In proceedings of *ACM SIGCOMM*, '00.

[3] Lee Breslau, Pei Cao, Li Fan, Graham Philips and Scott Shenker, "Web Caching and Zipf-like Distributions: Evidence and Implications", In proceedings of *IEEE INFOCOM*, '99.

[4] Pei Cao, Jin Zhang and Kevin Beach, "Active Cache: Caching Dynamic Contents on the Web", In proceedings of the *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Middleware*, September '98.

[5] Jim Challenger, Arun Iyengar and Paul Dantzig, "A Scalable System for Consistently Caching Dynamic Web Data", In proceedings of the *18th Annual Joint Conference of the IEEE Computer and Communications Societies*, '99.

[6] Fred Douglis, Antonio Haro and Michael Rabinovich, "HPP: HTML Macro-Preprocessing to Support Dynamic Document Caching", In proceedings of the *1st USENIX symposium on Internet Technologies & Systems, USITS*, December '97.

[7] Vegard Holmdahl, Ben Smith and Tao Yang, "Cooperative Caching of Dynamic Content on a Distributed Web Server", In proceedings of the the *7th IEEE International Symposium on High Performance Distributed Computing*, '98.

[8] Barron Housel and David Lindquist, "WebExpress: A System for Optimizing Web Browsing in a Wireless Environment", In proceedings of the *ACM/IEEE 2nd annual international conference on Mobile computing and networking, MOBICOM*, November '96.

[9] James Hunt, Kiem-Phong Vo and Walter Tichy, "Delta Algorithms: An Empirical Analysis", *ACM Transactions on Software Engineering and Methodology*, Vol. 7, No. 2, April '98.

[10] Arun Iyengar and Jim Challenger, "Improving Web Server Performance by Caching Dynamic Data", In proceedings of the *1st USENIX symposium on Internet Technologies & Systems, USITS*, December '97.

[11] Shudong Jin and Azer Bestavros, "Greedy dual web caching algorithm: exploiting the two sources of temporal locality in the web request streams", In proceedings of the *5th Int. Web Caching and Content Delivery Workshop*, Lisbon, Portugal, May '00.

[12] David Korn and Kiem-Phong Vo, "The VCDIFF Generic Differencing and Compression Data Format", Internet-draft, http://www.ietf.org/internet-drafts.

[13] Jeffrey Mogul, Fred Douglis, Anja Feldmann and Balachander Krishnamurthy, "Potential benefit of delta encoding and data compression for HTTP", In proceedings of *ACM SIGCOMM*, '97.

[14] Andy Myers, John Chuang, Urs Hengartner, Yinglian Xie, Weiqiang Zhuang and Hui Zhang, "A Secure, Publisher-Centric Web Caching Infrastructure", In proceedings of *IEEE INFOCOM*, '01.

[15] Venkata Padmanabhan and Lili Qiu, "The Content and Access Dynamics of a Busy Web Site: Findings and Implications", In proceedings of *ACM SIGCOMM*, '00.

[16] Ben Smith, Anurag Acharya, Tao Yang and Huican Zhu, "Exploiting Result Equivalence in Caching Dynamic Web Content", In proceedings of the *2nd USENIX symposium on Internet Technologies & Systems, USITS*, October '99.

[17] Neil Spring and David Wetherall, "A Protocol-Independent Technique for Eliminating Redundant Network Traffic", In proceedings of *ACM SIGCOMM*, '00.

[18] Alec Wolman, Geoffrey Voelker, Nitin Sharma, Neal Cardwell, Anna Karlin and Henry Levy, "On the scale and performance of cooperative Web proxy caching", In proceedings of *ACM SOSP*, '99.

[19] Huican Zhu and Tao Yang, "Class-based Cache Management for Dynamic Web Content", In proceedings of *IEEE INFOCOM*, '01.

[20] "Breaking New Ground In Content Acceleration", FineGround Networks Inc Whitepaper, http://www.fineground.com/prod/whitepaper.shtml.

[21] Apache HTTP Server Project, http://httpd.apache.org/.

[22] MyVitalAgent, http://www.qip.lucent.com/