INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2005; **18**:119–142
Published online 13 December 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/dac.692

Estimating and eliminating redundant data transfers over the web: a fragment based approach

Christos Bouras[1,2,*,†,‡] and Agisilaos Konidaris[1,2,§]

[1] *Computer Engineering and Informatics Department, University of Patras, GR-26500, Patras, Greece*
[2] *Computer Technology Institute-CTI, Riga Feraiou 61, GR-26221 Patras, Greece*

SUMMARY

Redundant data transfers over the Web, can be mainly attributed to the repeated transfers of unchanged data. Web caches and Web proxies are some of the solutions that have been proposed, to deal with the issue of redundant data transfers. In this paper we focus on the efficient estimation and reduction of redundant data transfers over the Web. We first prove that a vast amount of redundant data is transferred in Web pages that are considered to carry fresh data. We show this by following an approach based on Web page fragmentation and manipulation. Web pages are broken down to fragments, based on specific criteria. We then deal with these fragments as independent constructors of the Web page and study their change patterns independently and in the context of the whole Web page. After the fragmentation process, we propose solutions for dealing with redundant data transfers. This paper has been based on our previous work on 'Web Components' but also on related work by other researchers. It utilises a proxy based, client/server architecture, and imposes changes to the algorithms executed on the Proxy server and on clients. We show that our proposed solution can considerably reduce the amount of redundant data transferred on the Web. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: web components; web cash and proxy; redundant data transfers; performance; web fragment

1. INTRODUCTION

The Web is a constantly evolving set of resources called Web resources. Many studies [1–3] have shown that the evolution of the Web is continuous and rapid. Web resources can be identified, in a broad interpretation of the term, as 'any file that can be requested by a Web client'. This includes all files contained on a Web server that can be requested with the use of the HTTP protocol. HTML files, image files, pdf files and doc files are only some of the file types that can be considered Web resources.

*Correspondence to: Christos Bouras, Department of Computer Engineering and Informatics, University of Patras, GR-26500, Patras, Greece.
† E-mail: bouras@cti.gr
‡ Associate Professor.
§ E-mail: konidari@cti.gr

Received August 2003
Revised July 2004
Accepted September 2004

The issue of redundant data transfers over the Web has concerned the Web community for quite a long time, and is still considered a major issue today. Redundant data transfers have mostly been related to the resource change estimation problem. Many technologies and techniques have been adopted, in order to reduce redundant data transfers. Client Caching, proxy servers and specific HTTP headers are only some of them. Redundant data transfers are mainly attributed to re-transmissions of unchanged Web pages. Thus, it is clear why the redundant data transfer problem has been related to the resource change estimation problem. Only when a proxy or a cache can efficiently estimate the change frequency of a Web page, can it correctly decide whether to request a new copy of a page or use a local copy.

The problem of Web page change estimation can be diversely approached. Let us focus on a specific Web page. The question is: 'How often does the Web page change and how can the change be identified?'. This question can be answered differently, from the point of view of a Web server, a Web crawler or a Web proxy. A Web server 'knows' when a Web page changes, because the Web page is stored on its hard disk as a file whose updates can be easily monitored. A Web proxy can identify a Web page change by comparing the page's HTML to a previous version, or just by requesting the if-modified-since HTTP header (when available). In terms of change estimation it is difficult to be accurate from a Web proxy point of view since the only changes that the proxy can be aware of are those that occur in versions of the page that are available to it. Since in most cases a Web proxy follows a passive request approach (relying on users' requests) the versions of a Web page between consecutive requests are never visible to the proxy, and that is why change estimation is a somewhat difficult task. The case of the Web proxy covers that of a Web crawler in most points.

In this paper, we look at the Web page change estimation problem in relation to Web data transfers. Let us assume that a Web client cache is able to accurately compute the change frequency of a Web page. This means that the client cache knows when to actually request a 'fresh' copy of the page from a Web server and when to use a locally stored copy. We argue that even in this case, the client causes the transfer of redundant (or unchanged) data over the Web. This argument is based on the observation, that even when Web pages actually change, they do not change completely. In fact, as we will show in our study, most of the data in a page do not change, even when the page can be identified as 'changed'. This has to do with the Web page structure. The structure of a Web page changes very rarely as we will also show.

This work does not aim at introducing a new Web page change estimation methodology as in Reference [4], for example. Our main concern is not how to estimate the change frequency of pages. All of our observations, results and proposals aim at the reduction of transferred data over the Web through a novel approach on how to evaluate identified changes in Web pages. Some change estimation techniques may be more efficient than others in the context of our methodology. Our proposal may utilise all of these techniques. In order to make things as clear as possible in this paper, we use the simplest technique for change estimation. Changes are predicted with the use of a statistical model, based on change probabilities that are in turn derived from experimental observations.

A lot of work has been done in the fields of Web page change estimation and the reduction of web data transfers. The issue of statistical change estimation has been a popular problem [5]. In this paper we do not study the issue from a statistical point of view. We only use very simple estimation theories as an aiding tool for our fragment approach. The fragment approach that we present here is based on our previous work in Reference [6]. The rate and frequency of change of

Web resources has also been a popular problem with many interesting and significant studies such as [2–4, 7].

Some very important work on fragments and their significance to Web performance has been presented in References [7, 8]. There are also some Web services available that follow a very similar approach to ours [9].

The work in Reference [4] analyses the change frequency estimation problem very efficiently and has been a very helpful basis for our approach. We must also refer to the view and dynamic Web page materialization problem as a related field to our own [10–13], since our approach can have applications to dynamic page materialization policies. We have actually referred to this problem in the context of Web components (called fragment in this paper) in our previous work.

The work in Reference [14] refers to Web proxy cache replacement policies and is related to our work since the experimental study in this paper is carried out under a client/proxy/Web server scheme. Finally the work in References [15–17] analytically presents the Web latency problem that we also attempt to efficiently address in this paper.

Our methodology in this work is quite straightforward. In Section 2 we provide an overview of the Web page change estimation problem and describe how we relate it to the reduction of Web data transfers. In Sections 3, 4 and 5 we analytically present our experimental and computation methodology and approach, along with our initial results on redundant Web data transfers. In Section 6 we present two techniques that would prevent redundant Web data transfers. In Section 7 we present our future work and in Section 8 our conclusions.

2. THE WEB PAGE CHANGE ESTIMATION PROBLEM

Several studies have focused on the 'frequency of change' problem [1–4]. Most of these studies have estimated the change frequency of selected Web pages, after repeatedly requesting them over a period of time. There are two basic parameters (or sub-problems) involved in the process of estimating Web page changes.

The first sub-problem is the definition of the source data. All the studies, until today, have focused on web pages as HTML files (e.g. from the $<$HTML$>$ tag to $<$/HTML$>$ tag). The studies follow the straightforward process of requesting specific Web pages and comparing the resulting HTML. The result of the comparison is whether a Web page has changed or not during the time elapsed from a previous request. This process, of comparing HTML files, can lead to insufficient or inaccurate results. These results, in most cases, may be attributed to factors such as adds, page counters, etc. In this paper we estimate Web page change probabilities after an initial Web page manipulation procedure. This manipulation procedure is able to isolate change-causing parameters, such as adds and page counters. We explain the HTML manipulation algorithm that we used to achieve this in the following paragraphs.

The second sub-problem, related to Web page change estimation, is the methodology used to estimate the actual occurrences of change in specific source data. In most cases Web page changes are computed through change frequency estimation, by using the so-called 'naive estimator'. If N are the total number of changes observed during a request period equal to T, the 'naive estimator' computes the Web page change frequency as $F = N/T$. Of course, there are more efficient techniques to estimate the frequency of change in a Web page [4, 18].

The ultimate goal of this work is to improve the performance of Web servers, Web crawlers and Web proxies by efficiently reducing redundant Web data transfers. Intuitively we expect

that with the use of our methodology, Web servers will be able to improve their dynamic Web page materialization policies and Web server caching algorithms [11, 12, 19]. We expect that Web crawlers will be able to index 'fresh' copies of Web pages, thus improving their services and performance. We also expect that Web proxies will be able to cache 'fresh' copies of Web resources, and in this way speed up their users' browsing. All the above can be considered as the goals of our work.

The substantial contribution to previous work that we will make in this paper, is related to the identification of the source data used to estimate Web page changes. In previous work the source data used to estimate Web page change was the whole HTML code that was the result of a request to a specific page. In this paper we will consider different source data. We describe our methodology analytically in the following sections, but in short we use an HTML manipulation algorithm before extracting results from a requested page. This algorithm receives the whole HTML code of a Web page as an input, and produces fragments of the same Web page. The fragments will help us determine if changes have actually occurred in a Web page and where exactly they have occurred.

3. METHODOLOGY AND APPROACH

In this paper, we base our results on HTML files collected from popular Greek and International Web sites. The sites that we selected were portals and news sites. The reason for this selection was that we intended to monitor sites with frequent changes on their pages. Our sample data, even though not very extensive, is very representative of our target group of sites. We aim at reducing data transfers for sites that experience extensive requests that result in extensive network traffic. These sites were selected for very specific reasons. They are very professional and tidy in terms of HTML. They are very popular and their pages change frequently. These are the Websites that need to be relieved of redundant data transfers.

After collecting HTML files, we measured several change parameters in order to define the following:

- if a change had occurred in the Web page,
- where (topologically) the change occurred,
- how changes were related in a Web page.

The basic goal of research was to identify the pattern of change for every page. When one thinks of the Web page change problem, the intuitive expectation is that not all Web pages change with the same rate. A Web user can have an idea of the change rate of Web pages that he/she visits frequently. Even so, a user rarely scans the whole Web page to identify a possible change. The user in most cases is interested in a certain portion of the page and immediately looks for changes there. A Web page may have changed and a user may not spot the change since it has occurred in a portion of a page that does not interest him. Especially in the case of pages that require a lot of scrolling to view, changes can easily escape the users' eye. Our goal in this paper is to analyse many occurrences of a Web page (after repeated requests) in order to define if, when and where it has changed. We expect to identify the parts of the Web page that frequently change and how changes on the same page are related to one another. This will help us 'break-down' a Web page into what we call 'change zones' and create 'change relations'. The basic idea

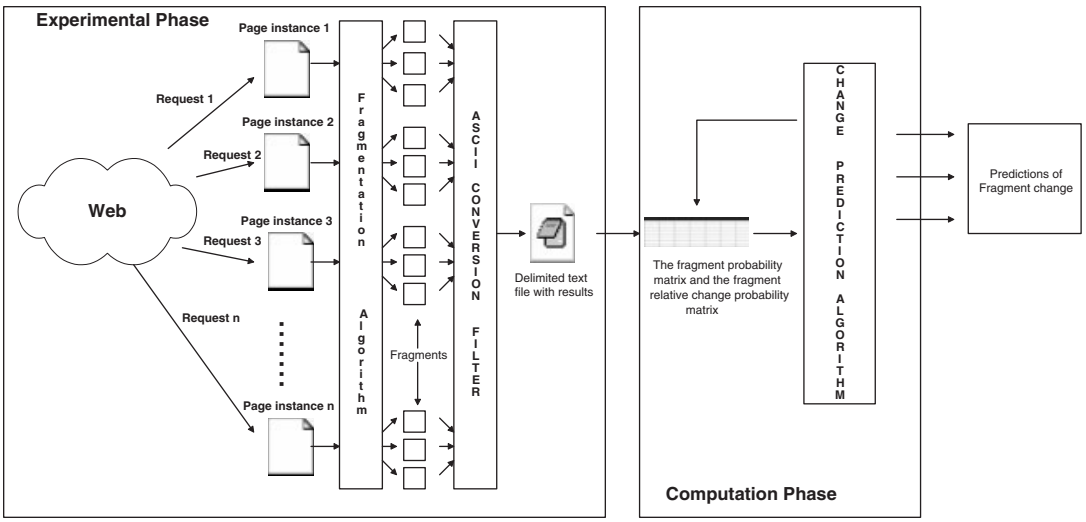

Figure 1. A schematic representation of the methodology used.

was to be able to reduce the amount of data moved around on the WWW. Why should unchanged data travel through the WWW, just because they are parts of changed Web pages?

In this paper we assume that the only material that we have at our disposal in order to identify and 'predict' changes in a Web page is the HTML code of the whole page. Our methodology has two steps:

- an experimental phase that is actually a content-gathering and manipulation step and
- a computation phase where content processing is carried out (Figure 1).

In the following sections we analytically explain these two phases of our methodology.

4. EXPERIMENTAL PHASE

During the first step of our study we collected data. Through a Java Web client that we implemented, we issued repeated requests for the home pages of 10 popular portal and news sites around the world. The experiment lasted for about 24 h. The requests to the home pages of the sites were executed at a rate of 1 request per 3 min. At every request, our client requested the home page of the site under review and saved the returned HTML to a text file. After collecting all the HTML files we passed over to the second step of this phase. The second step included HTML analysis and manipulation. The specific analysis that we performed relied on an HTML fragmentation algorithm.

The fragmentation algorithm is the basis for all other experimental evaluation steps. This algorithm was mainly based on our previous work [6, 13]. The basic goal was to fragmentize all the Web pages that we had saved. The procedure of fragmenting Web pages, as presented in this paper, is very simple and straightforward. Our goal is to use our technique for all pages in Web sites and not only specific Web pages. This is very important since the same fragments of data are included in multiple Web pages of a Web site. This accounts for Headers, Menus, footers,

etc. These fragments are usually redundant since they seldom change. The process was influenced by four basic parameters:

- Web service similar to the one found in Reference [9]. These services can fragmentize different Web pages and then pull together different fragments into one browsable Web page.
- Our previous work on Web components. In Reference [6], we propose a conceptual fragmentation technique that can help Web developers, analysts and users to improve Web performance and personalization. We have observed that almost all Web pages are constructed with the use of independent conceptual fragments. These fragments are usually autonomous and can meaningfully stand alone.
- Previous work by other researchers and especially the 'data bundling' techniques described in References [7, 8]. In this work the authors propose the bundling (or binding) of data that changed in similar ways or showed similar characteristics. This technique can considerably improve Web caching.
- The Web page development technique that uses Server Side Includes. This technique can be very helpful to data intensive Web sites as described in References [11, 12, 19]. Every Server Side Include changes independently. If a Web proxy or a Web client could have the ability to treat Server Side Includes independently, Web page changes could be predicted in a more accurate way.

The fragmentation algorithm can be viewed as an HTML filter. This filter, fragmented the Web page based on the <BODY>, </BODY>, <IMG>, <TABLE> and </TABLE> tags. The <TABLE> and </TABLE> tags were selected as fragment delimiters because they are the most popular data structuring tags used in 'contemporary' HTML pages. The filter was able to break-down the HTML of the page, initially into two parts:

- the HTML outside the <BODY> tag and
- the HTML inside the <BODY> and </BODY> tags.

Then, the HTML inside the <BODY> and </BODY> tags was broken down recursively into all the tables that it was made up of. The HTML inside the <BODY> and </BODY> tags was also stripped of all the <IMG> tags and their contents. The results of this manipulation technique were:

- the whole HTML file,
- the contents of the <BODY> and </BODY> tags,
- the contents of the <BODY> and </BODY> tags, stripped of all <IMG> tags and their contents,
- all the individual tables contained in the <BODY> and </BODY> tags. These tables were called fragments.

All nested tables were treated according to the following example:

Example 1
Let us assume that we have the following portion of an HTML page: The nested table was treated as an independent fragment and the parent table (the whole code) together with the

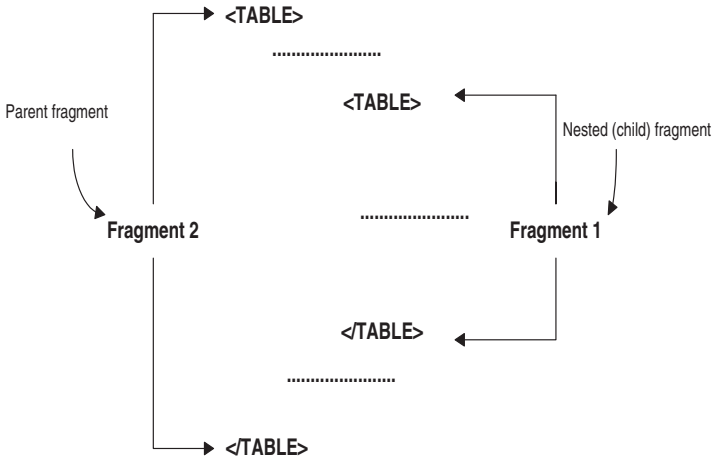

Figure 2. Example of fragment nesting identification.

nested table was considered as another independent fragment. The nested table data structure is very accurate and representative of the information contained in a Web page since all fragments make up the whole Web page when connected according to the fragment tree structure. However complex this tree may be (in cases of large Web sites) it will not become a resource or programming issue since there are many programming language data structures that can be used to accommodate such a service (Figure 2).

After the fragmentation of the HTML, an ascii conversion process was carried out for every fragment of every page. The ascii equivalents of every character in a fragment were added and stored in a new text file (one for every site). At the end of the experimentation phase we ended up with a delimited text file that contained the ascii equivalents of all fragments and other HTML components (e.g. the whole HTML file, the contents of the $<$BODY$>$ and $<$/BODY$>$ tags and the content of the $<$BODY$>$ and $<$/BODY$>$ tags stripped of the $<$IMG$>$ tags and content) for every page request.

5. COMPUTATION PHASE

In this paragraph we present the computation phase of our methodology. The input of this phase is the delimited text files that were the output of the previous phase. The text files will be used to extract results, related to the change probability and change pattern of every page. At this point, we must point out that all our findings on fragments are directly related to the HTML elements that are connected to them. For instance if a fragment has not changed, the HTML elements (images, flash animations etc.) that are connected to it, have also not changed. This means that they do not have to be requested by a client, as shown later. For space saving reasons we will use a specific Web site home page from now on in this paper. The selected home page was that found at europe.cnn.com. This table was selected because the results (Table I) that we will extract from it are representative of all other Web site home pages that we worked on.

Table I. The page anatomy parameters for the home page of europe.cnn.com.

	Number of fragments	Number of images	Number of total fragments changing	Percent of data not in body tags
europe.cnn.com	29	82–85	17	~3

Table II. Fragments that changed in every change occurrence in the page.

Change occurrence	Fragments changing
1	F3, F9, F20, F21, F26
2	F3, F9, F17, F20, F21, F22, F25
3	F3, F9, F12, F17, F20, F21, F22
4	F3, F9, F12, F17, F20, F21
5	F3, F9, F17
6	F3, F9, F12, F20, F21
7	F3, F9, F12, F17, F20, F21

5.1. *The fragment change probability and relative change probability matrixes*

The fragment change probability matrix contains the change probabilities for all the identified fragments of the Web page. It also contains the relative change probabilities for all the fragments. The change probability identifies the probability with which a fragment changes according to our experimental results. The notation $\mathbf{F}_1, \mathbf{F}_2, \ldots, \mathbf{F}_n$ is used to identify the fragments in the matrix.

The change probabilities were derived with the use of the following simple probability definition: this probability of change for a fragment observed $N_{\mathbf{F}_n}$ times, during which $K_{\mathbf{F}_n}$ changes were observed is equal to (Table II).

$$P(\mathbf{F}_n) = \frac{K_{\mathbf{F}_n}}{N_{\mathbf{F}_n}} \tag{1}$$

The relative change probability of the fragments was computed with the use of formula 2 that provides the relative change probability of fragment F_B when fragment F_A changes

$$P(F_B/F_A) = \frac{P(F_A F_B)}{P(F_A)} \tag{2}$$

With the use of Formulae 1 and 2, we constructed the change probability and the relative change probability matrixes shown in Tables III and IV, respectively, for the home page of europe.cnn.com.

5.2. *Page anatomy identification*

The goal of this step was to identify the 'anatomy' of a Web page. This step intended to analyse the Web pages in order to give us an idea of the fragments and their placement inside the pages. In this paragraph we, once more, use the home page of europe.cnn.com as an example. During this step we measured the following values:

- number of fragments identified in the Web page,

Table III. The fragment probability matrix for the home page of europe.cnn.com.

	F3	F9	F12	F17	F20	F21	F22	F25	F26
Change Probability—$P_{\mathbf{F}_n}$	0.033	0.033	0.019	0.024	0.029	0.019	0.014	0.005	0.005
Standard deviation—$\sigma_{P_{\mathbf{F}_n}}$	0.013	0.013	0.085	0.012	0.012	0.012	0.090	0.006	0.009
$P_{\mathbf{F}_n^+}\sigma_{P_{\mathbf{F}_n}}$	0.046	0.046	0.104	0.036	0.041	0.031	0.104	0.011	0.014

Table IV. The fragment relative change matrix for the home page of europe.cnn.com.

Europe.cnn.com	F3	F9	F12	F17	F20	F21	F22	F25	F26
Whole page	1	1	0.57	0.71	0.86	0.86	0.43	0.14	0.14
When F3 changes	1	1	0.57	0.71	0.86	0.86	0.43	0.14	0.14
When F9 changes	1	1	0.57	0.71	0.86	0.86	0.43	0.14	0.14
When F12 changes	1	1	1	0.75	1	1	0.50	0	0
When F17 changes	1	1	0.60	1	0.80	0.80	0.60	0.20	0
When F20 changes	1	1	0.67	0.67	1	1	0.50	0.16	0.16
When F21 changes	1	1	0.67	0.67	1	1	0.50	0.16	0.16
When F22 changes	1	1	0.67	1	1	1	1	0.33	0
When F25 changes	1	1	0	1	1	1	1	1	0
When F26 changes	1	1	0	0	1	1	0	0	1

- number of images,
- number of total fragments changing,
- percent of data outside the <BODY> and </BODY> tags.

The number of fragments remained the same during the whole experiment for europe.cnn.com. This is a clear indication of Web page structure stability. The same indication comes from the number of images that only vary from 82 to 85 during a whole day. During the whole experiment the structure of the home page of europe.cnn.com remained the same as the one shown in Figure 3.

Out of 29 total fragments, changes were observed in 17 of them. This number included fragment nestings. This means that a change to a nested fragment was also recorded as a change to the parent fragment. By taking a closer look at the structure of Figure 3 we intuitively expect that the fragments that actually change will be less than 17. We will prove this in the following paragraphs. Finally, only 3% of the total data of the page was data that could be found outside the <BODY> and </BODY> tags. This indicates that the page is data rich and that data outside the body tags cannot cause significant redundant transfers even if it changes frequently (it was actually never observed to change).

5.3. The '*reason of change*' *identification*

This step aimed at identifying the 'reason of change' of a Web page. We look at every request that was identified as 'changed' from the previous request. The goal was to identify why the page was identified as 'changed' (meaning what actually changed in the page). The delimited file that holds the results of the experimental phase also holds the fragment dependencies.

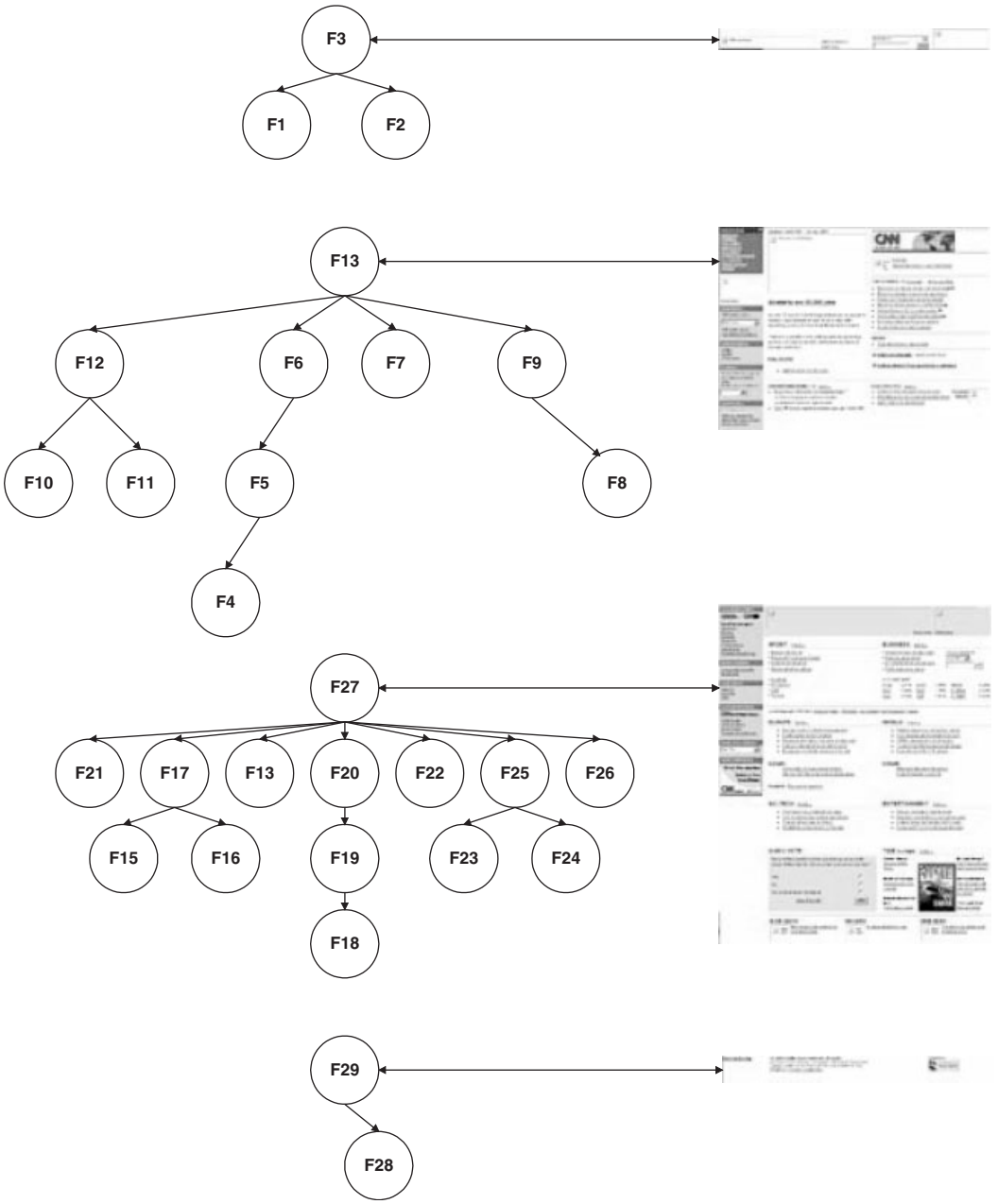

Figure 3. The fragment tree of the home page of europe.cnn.com and the corresponding visual representations. It is clear that the page is constructed with the *header*, *main body*, *footer* approach.

The fragment dependencies represent table nestings and help us construct the fragment tree for the Web page under consideration. The fragment tree for the europe.cnn.com site is shown in Figure 3.

After creating the fragment tree we first eliminate all fragments that have not changed at all during our experimental phase. These are: F1, F2, F10, F6, F7, F14, F16, F24, F29. All fragments nested in fragments that never change (e.g. F28 in F29) are also eliminated. We then follow a 'conservative' approach in the detection of totally dependent fragments. A totally dependent fragment is one whose changes can be safely attributed to a fragment nested in a lower level in the fragment tree. It is not easy to eliminate fragments in this way. Our conservative approach states that a parent fragment can be eliminated as totally dependent on a nested fragment(s), only when, after removing the nested fragment(s), no HTML remains inside the parent fragment. In any other case we eliminate the nested fragment and keep the parent fragment, that is a data super-set in every case. For example in our study F20, F19 and F18 have shown six changes, in total, each. After a closer examination we found that after removing F18 and F19 HTML actually remained in F20. This means that a change in F20 could be attributed to:

- a change inside F18 or F19,
- a change in the HTML outside F18 and F19 but inside F20.

Thus changes to F20 could not be explicitly attributed to either F19 or F18. In this case we eliminated F18 and F19 and kept F20 as their 'representative' fragment. After this second step, we ended up with the smallest possible sub-set of fragments that could change when a change was observed in the Web page. These fragments are: F3, F12, F9, F26, F25, F22, F21, F20 and F17.

It is interesting to show which tables these fragments actually represent in the page of europe.cnn.com. The fragments' as seen by the users are shown in Figure 4. The europe.cnn.com home page is clearly cut down to change zones representing fragments. The Web page is initially cut down to four zones representing four main tables. It is clear that the developers have followed a header, main body and footer approach. It is likely that the page is constructed with the use of four Server Side Includes that change independently.

Our experimental results show that a change in the europe.cnn.com home page can safely be attributed to one of the fragments shown in red circles in Figure 4. But not all fragments changed every time a change was observed in the Web page. Table II shows the fragments changing, every time a change was observed in the page.

Our goal here is to identify the redundant data transferred, as part of the unchanged fragments and the static HTML parts, inside the Web page, every time a change was observed. The following formula provides this percentage:

$$\mathrm{Data}_{\mathrm{red}} = \mathrm{Data}_{\mathrm{all}} - (\mathrm{Data}_{\mathrm{fnc}} + \mathrm{Data}_{\mathrm{fcc}}) \tag{3}$$

In formula 3, $\mathrm{Data}_{\mathrm{red}}$ is the percent of the redundant data transferred, $\mathrm{Data}_{\mathrm{all}}$ is all the data in the page (100%), $\mathrm{Data}_{\mathrm{fnc}}$ is the data contained in data fragments that never change and $\mathrm{Data}_{\mathrm{fcc}}$ is the data in fragments that have been observed to change but have not changed in the current change.

Formula 3 was used for every occurrence of a change in the page and the results are shown in Figure 5.

It is clear from Figure 5 that the data transferred inside a Web page of europe.cnn.com when the page actually changes, has not changed on average by more than 43%. This means that the

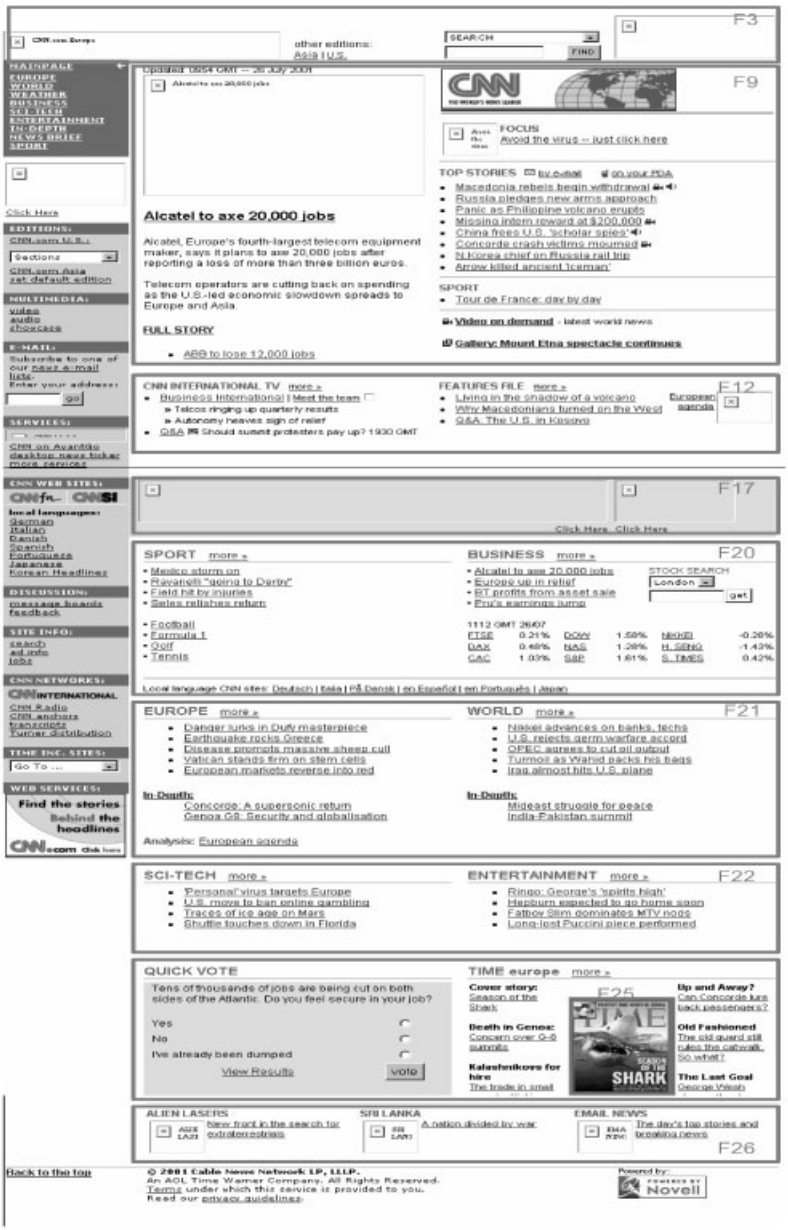

Figure 4. The visual representations of all independent fragments in the europe.cnn.com home page.

amount of redundant (or useless, or unchanged) data transferred together with the changed data inside a changed page amounts to 57% of the total data on the page. This fact confirms our intuitive approach and provided us with enough motivation to find a solution that could prevent this phenomenon.

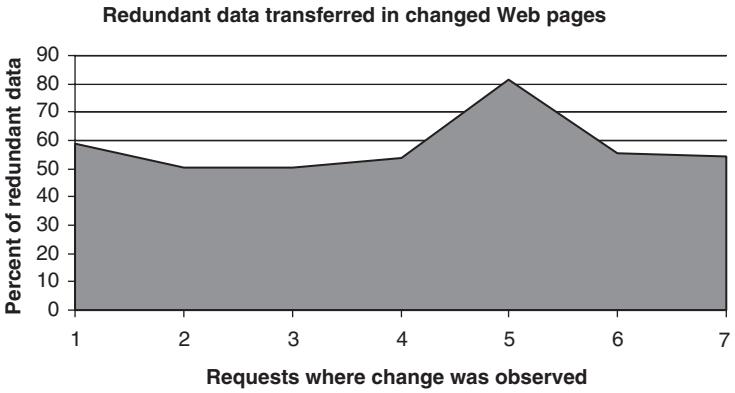

Figure 5. Redundant data transferred in Web pages that have changed.

6. PROPOSED SOLUTION BASED ON CLIENT SIDE INCLUDES

In this section, we propose techniques that could help solve the problem of redundant data transfers over the Web. The basic cause of redundant transfers is the absolute binding of redundant data to useful (changed) data, in the context of Web pages. The problem can be solved if all portions of data (fragments) could, in some way, be treated (and requested) independently, and not under a unified page request scheme. The proposed solution is a straightforward one, and can be immediately put to work, since it utilizes current Web techniques and practices. Our solution consists of two techniques and is based on the concept of Client Side Includes. The basic idea is that a client requests a page that is actually a page template. This template consists of many Client Side Includes that represent fragments. Each time the client requests the page an algorithm is executed. With the use of this technique, after a 'slow-start' period, possibly caused by insufficient initialization data, the client will be able to accurately identify the Client Side Includes that should be requested. The rest will be requested by local cache since they will not have changed. The basic problem of this approach is that all pages must be coded with the Client Side Includes approach. This would require major changes to current Web page design techniques and is totally unacceptable. But this is true only for pages on Web servers. Let us consider the case of an intermediate server (e.g. a proxy server) that is able to fragment and re-build Web pages originating from different Web servers. The general model of this approach is shown in Figure 6.

The clients request pages from different Web servers through a proxy server. The proxy server has the ability to fragmentize each page (in the way that we have described in this paper). The fragmented page is then re-constructed with the use of Client Side Includes at the proxy server and sent to the requesting clients together with the change probability or the relative change probability matrixes. The whole procedure is transparent to the client.

The techniques that we will describe in the following paragraphs can be implemented at the clients or at the proxy server. Both techniques are based on a matrix (change probability matrix or relative change probability matrix). These matrixes are initially constructed at the proxy server. If the techniques are implemented at the clients, the appropriate matrix must be transferred to the client before execution. Both techniques presented in this paper are

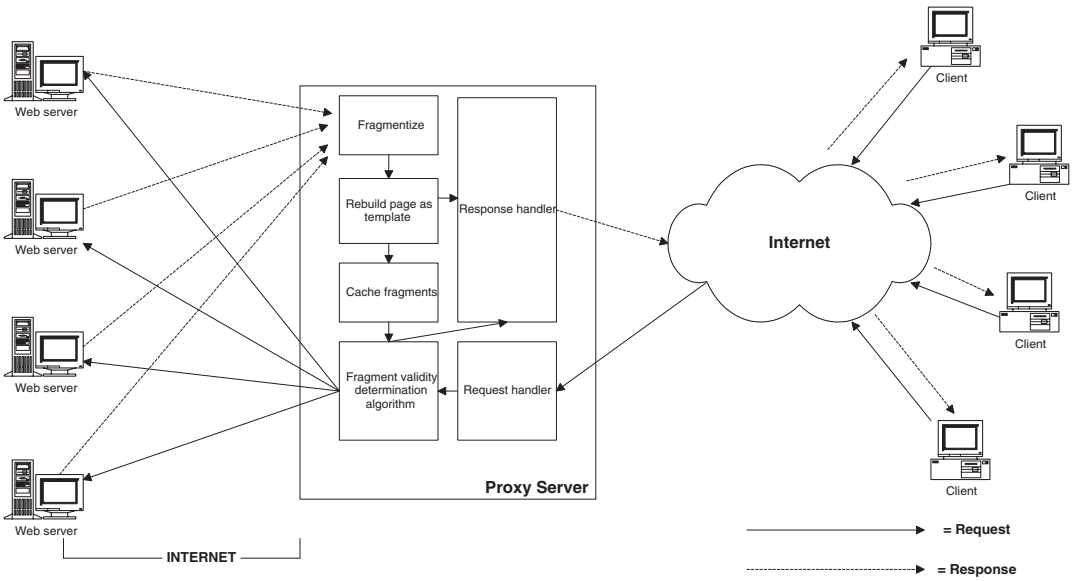

Figure 6. The client/proxy/Web server architecture that can support the Client Side Includes approach.

implemented at the clients after the transfer of the matrixes from the proxy servers. This approach does not exploit the maximum potential of the Client Side Includes technique, since the proxy server itself must request the whole page from the originating server when needed with the use of the if-modified-since header. The actual gain is in the transfers between the proxy server and the clients. In the following sections we will thoroughly explain and quantify the procedure.

In order to show the efficiency of our approach we will work on a real example. The site that we will use in the example will be europe.cnn.com. We will be using the home page of the site as sample data. The first problem that we will work on is the transformation of the home page HTML to an equivalent that uses Client Side Includes. In order to achieve this we recursively replace every table in the HTML with a Client Side Include. The procedure is clearly shown in Figure 7.

It is clear that the nested table structure will be replaced by a Client Side Include structure, but the content will remain unchanged. Having the new page structure in mind, we present two techniques that can significantly improve data transfer over the Web.

6.1. Technique 1: dependent on the change probability matrix

This first technique depends on the fragment change probability matrix. This matrix holds the change probabilities of every fragment and the standard deviation of every fragment change probability. The standard deviation is computed with the use of formula 4.

$$\sigma = \sqrt{\mathrm{Var}[(P(Fn)]} \tag{4}$$

Formula 4 computes the standard deviation of the change probability of fragment F_n as the square root of the variance of the change probability.

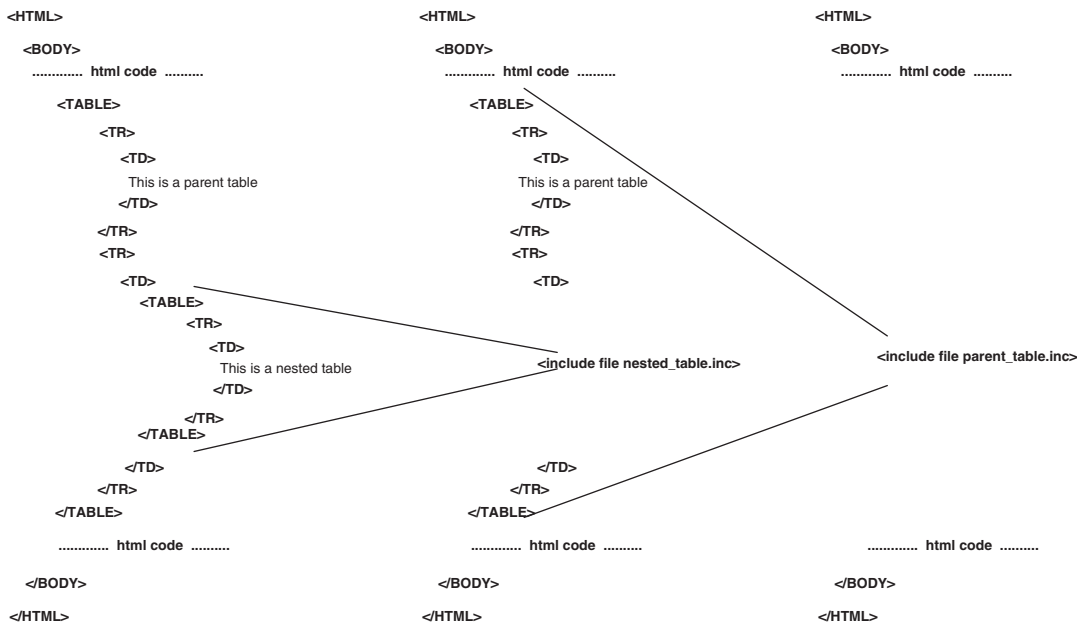

Figure 7. The recursive table replacement process.

As shown in previous paragraphs, after the fragment elimination phase we have come up with nine fragments that are responsible for all changes observed at the europe.cnn.com site home page and can be considered independent. These are the fragments that we will use in the change probability matrix.

This technique uses the change probability matrix to 'discover' if a fragment has changed or not, at every client request. The technique is described by the following algorithm (in pseudocode):

```
1  if page Pi has been requested
2        If Current_Time < Time_Threshold then
3           Construct fragment documentation matrix
4           Define fragment structure and dependencies
5           Eliminate non-changing and dependent fragments
6           Construct fragment change probability matrix
7        End if
8        If Current_Time>Time_Threshold then
9           For all independent Fragments
10                If Fragment change probability > P_Threshh then
11                   Request fragment
12                End if
13          Next
14       Update fragment change probability matrix
15       End if
16 End if
```


The algorithm that we have presented above has a unique characteristic. It can be executed with very similar results on Web servers, Web proxies and Web crawlers. Here, we look at a Web proxy. The basic goal of the algorithm is to minimise the amount of redundant data that a Web proxy queries, caches and transfers to clients.

In order to efficiently explain the algorithm proposed we will present a step-by-step explanation.

The first step is the data collection phase (*lines 2–7*). During this phase, the algorithm only collects data. If we see it from the perspective of a Web proxy, for a specific time window, the proxy requests the europe.cnn.com home page every time one of its clients requests it. It is clear that during this period of time, the users are viewing fresh data at every request. At the same time the proxy is executing certain set-up phases of the algorithm. These phases are:

- *The fragment identification and documentation phase* (*line* 3)
- Every time the proxy receives a request for the europe.cnn.com page, it requests a fresh copy of the page from the Web server. The HTML of the page is analysed and cut down into fragments. Every fragment corresponds to a Client Side Include as mentioned in previous paragraphs, and in turn every Client Side Include corresponds to a table in the HTML of the page. During this phase of the data collection process, the HTML of every fragment identified, is put through an ascii conversion filter. The filter returns an ascii equivalent (the sum of the ascii equivalents of all the characters contained in the fragment) of every fragment. All ascii equivalents are saved in a matrix (fragment documentation matrix) on the proxy server. The fragment documentation matrix also contains the identified connections (nestings) of the fragments.
- *The fragment structure and dependency documentation phase* (*line* 4)
- After documenting all fragments and their ascii equivalents, the proxy server goes on to the fragment structure and dependency documentation phase. During this phase the Proxy server determines the structure of the html page by recording the structure of the fragments. This phase relies on the nesting of every fragment.
- *The fragment* (*node*) *elimination phase* (*line* 5)
- This module is executed immediately after the termination of the data collection phase and just before the normal execution phase begins. During this phase the proxy server initially uses the fragment documentation matrix to compute the changes in every identified fragment during the data collection period. The computation relies on the comparison of consecutive values of documented fragment ascii equivalents. Different values indicate the occurrence of a change. Of course a mistake may occur if the ascii values (meaning the corresponding characters in the html) have changed in a way that their sum remains the same. This is possible but by an empirical investigation we determined that this may happen in very rare cases. In any case the occurrence of such an event never occurred during the investigation of the europe.cnn.com site and in any case can be attributed to a statistical (and minor) error.
- After determining the changes in every fragment, during the data collection phase, the nodes that correspond to fragments that showed absolutely no change (number of changes/total requests = 0) were removed from the fragment structure matrix and the fragment structure graph.
- *The final pre-execution phase* (*line* 6)

- During this final phase the algorithm produces the fragment change probability matrix from data in the fragment documentation matrix. This matrix holds the probability of change for all independent fragments, that change at least once during the data collection phase.

The fragment probability matrix for europe.cnn.com is shown in Table III.

The second phase of the algorithm is the standard execution phase (*lines* 8–15). The input of the standard execution phase is the fragment probability matrix. This phase is executed on a 'per request' and a 'per fragment' basis. This means that the algorithm functions in a 'passive' mode, based on user request and is executed for every fragment in a page. The decision that must be made at the client is whether to request a fragment or not from the proxy server. This decision is made for all fragments contained in the page. During the standard execution phase the algorithm also considers a variable called the request threshold probability. This variable is initially set by the administrator (usually set to 1) but is updated at run-time. The request threshold probability indicates the probability value in the temporary probability matrix that a fragment must be equal to, in order to be requested.

In this paper we will set the request threshold probability to 1. Intuitively, the request threshold probability is the flag that initiates a fresh fragment request. What we are actually stating when setting it to 1 is that a fragment must be requested when its change probability is at least equal to the request threshold probability. Thus

$$P(F_n) \geqslant P_{\text{Thresh}} \text{ in order for } P(F_n) \text{ to be requested} \tag{5}$$

The temporary probability matrix holds the current values of the fragment change probabilities. Each time the page is requested, the temporary probability matrix is updated with a new value for all the fragments in the page. The update of the temporary probability matrix is executed according to the following formula:

$$P(F_n)\text{new} = (P(F_n)\text{old} + P(F_n)\text{current} + \text{Var}[P(F_n)])/2 \tag{6}$$

Formula 6 represents a totally conservative approach. The approach is conservative because fragment 'freshness' has been valued higher than redundant data transfer reduction. What is the meaning of reducing redundant data transfers over the Web if clients keep getting stale data. Formula 6 initially adds the probability variance to the value of the fragment change probability. By doing this, we maximize the fragment change probability. This intuitively maximizes the times that a fragment will be computed as 'changed' and thus, maximizes the times that it is requested. The current value of the fragment probability is added in order to capture the current 'trend' of change for the fragment. It is obvious that according to the time of day, fragments change in different intervals. For example, a stock quote changes frequently when the stock market is open but stops changing when it closes. Our technique adopts an initialization phase that computes fragment change probabilities over a long period of time. It is clear that the overall change probability of the stock quote would not provide accurate results for all times of the day. In order to avoid this as much as possible we averaged the value of the fragment change probability with the current change probability in order to be consistent.

The basic idea of this technique is to use the fragment change probability matrix to determine if a fragment has changed or not. Let us assume that clients are connected to a proxy server as in Figure 6. The proxy server has passed the initialization phase and has already constructed a

fragment change probability matrix for the europe.cnn.com home page. The clients start requesting the page. As a response, the clients receive a templated page consisting of Client Sided Includes and the change probability matrix of the requested page. At the first request, the clients do not have the includes in their cache and must request them. After the first request, the clients receive the same templated Web page. Only this time, the clients use the change probability matrix to determine whether to request a fragment from the proxy or use the locally cached copies. This technique requires a substantial period of time in the initialization phase in order to provide consistent results. In our experiments, we have used only 20 hours as the initialization phase. The redundancy and staleness graphs are shown in Figures 8 and 9, respectively.

This technique attempts to statistically map the results of the initialization phase to current requests. The initialization phase is used as the 'history of change' and is constantly updated by current findings. In our experiments we used an initialization phase of 20 hours and tested it with 210 requests. The results showed that redundant data were still transferred to the clients. On the other hand the technique is much better than the transfer of a new copy of a Web page

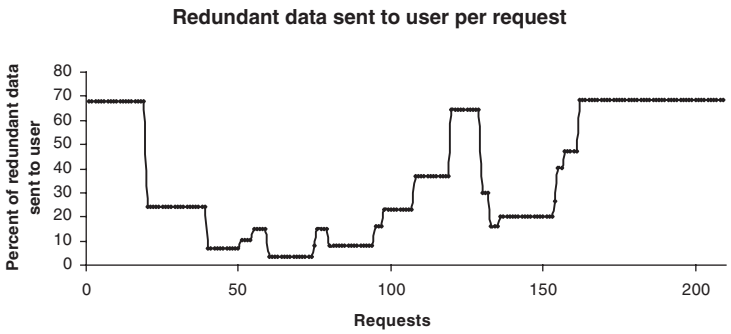

Figure 8. The amount of redundant data sent to clients.

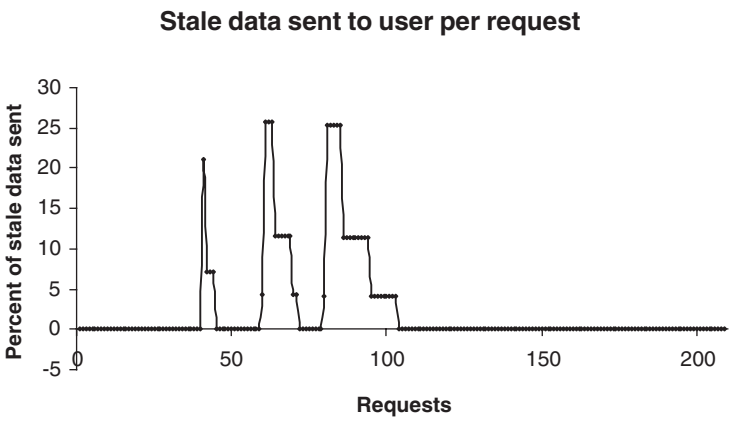

Figure 9. The amount of stale data sent to clients.

every time it is requested, which would cause the total (100%) page data transfer. We expect that this technique will show much better results under real conditions with an initialization period of more than a week, since the fragment change probabilities will be much closer to their true values.

As shown in Figure 9 certain requests initiate the transfer of stale data. The amount of stale data is small and confined to specific requests. We also expect the amount of stale data to drop for larger initialization phases.

6.2. Technique 2: dependent on the fragment relative change matrix

This technique is similar to the previous technique but does not consider the fragment change probability matrix but the fragment relative change matrix in the execution phase. The algorithm executed in this technique is the following:

```
1  if page Pi has been requested
2      If Current_Time < Time_Threshold then
3          Construct fragment documentation matrix
4          Define fragment structure and dependencies
5          Eliminate non-changing and dependent fragments
6          Construct fragment relative change probability matrix
7      End if
8      If Current_Time>Time_Threshold then
           Rank fragments
9          For all ranked fragments
               If fragment not already requested and not marked NOT CHANGED
10                 Request fragment
11                     If fragment has changed then
12                         Request all fragments with a relative change frequency equal to 1
13                         Mark NOT CHANGED all fragments with a relative change
       frequency equal to 0
14                     End if
15             End if
16         Update fragment relative change probability matrix
17     Next
18 End if
```

During the initialization phase the fragment relative change probability matrix is constructed with the use of formula 2. This matrix provides the relative change probabilities of all fragments to all fragments. It also provides the relative change probability of all fragments to the Web page. The first data line of the matrix provides the relative change probability of all the page's fragments when the whole page changes. For example Fragment 17 (F17) has a change probability of 0.71 when the whole page changes and Fragment 9 (F9) always changes when the whole page changes. The fragment relative change probability matrix for the home page of europe.cnn.com is shown in Table IV.

After constructing the fragment relative change probability matrix, the algorithm passes over to the execution phase. The client uses the matrix to decide whether to request a fragment from

the proxy server or use the one in local cache. We assume that the if-modified-since header is available for the page. After requesting the header the client knows whether the whole page has changed or not. If it has not changed it uses the local copy. If the page has changed, the client uses the fragment relative change probability matrix to determine the fragments that have changed. The fragments whose relative change probabilities are equal to 1, when the whole page changes, are immediately requested. These are always F3 and F9. After these requests the client must decide which fragment to request next, since all other fragments may or may not have changed since their relative probabilities are smaller than one. Obviously the client must rank the rest of the fragments in a 'smart' way. The ranking must be made in such a way that the best ranked fragments can give a lot of information about the rest of the fragments. The ranking can be done in various ways. In this paper we examine two fragment rankings:

The fragment probability and information based ranking. This ranking uses the fragment probability matrix described in technique 1 to rank the fragments and then, re-ranks the fragments according to the information that can be extracted by their relative change probabilities. The fragment with the greater probability value is ranked first and so on. Intuitively this can be justified by the fact that we want the next fragment that we request to have a large change probability in order for it to be useful. If the fragment has not changed we cannot use its information from the fragment relative change probability matrix, and we must go on to the next ranked fragment. This ranking provides the following order for the rest of the fragments: F20, F17, F21, F12, F22, F25, F26. If we look closer at the fragment relative change probability matrix we will find that F20 and F21 do not provide any information for any other fragment except themselves. Thus, we eliminate them. Also, F17 provides information for only one fragment. We also eliminate it. Our final fragment ranking is F12, F22, F25, F26, F17, F20, F21.

The information based ranking. This ranking only considers the possible information given by a fragment change for other fragments through the relative change probability matrix. In other words in ranks the fragments according to the ones (they are valued higher) and zeros (0) they contain in their row in the fragment relative change probability matrix. According to this ranking the fragments are ranked as follows: F25, F22, F26, F12.

After ranking the fragments (other rankings may be applied), the client requests the fragment ranked first. After receiving it, it checks if it has actually changed. If it has, then its corresponding line in the fragment relative change probability matrix is used to extract information about other fragments. When this is over and a decision has not been reached on how to treat all fragments, the client requests the next ranked fragment. This goes on until a decision has been reached for all fragments in the fragment relative change probability matrix.

It is obvious that this technique also values 'freshness' more than the reduction of redundant data transfers. This technique actually guarantees 100% fresh data and at the same time reduces Web data transfers. Since the if-modified-since header is provided, no data transfers are initiated when the page has not changed. The only interesting requests to look at are those that lead to a new version of the page. In Figure 10, we show the percent of redundant data transfers, in cases when the page has changed. The reduction in redundant transfers is clear compared to the classic if-modified-since proxy server approach. It is also clear that information based ranking gives better results than the combined approach of probability and information based ranking.

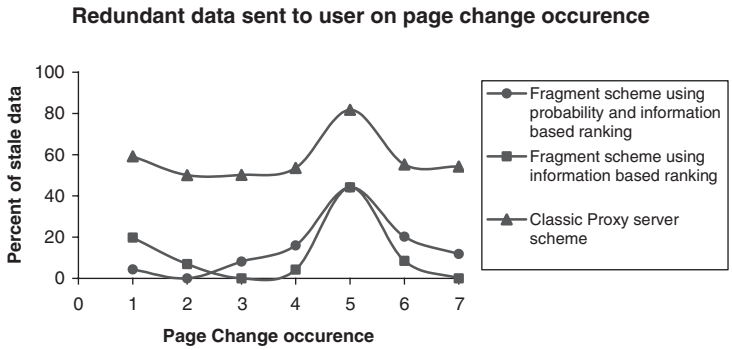

Figure 10. The redundant data sent to the user when the page changes.

7. FUTURE WORK

Future work can be carried out in all aspects of the problem that we have presented in this paper. The HTTP protocol inherently treats Web pages as entities. We are already working on a scheme that proposes minor changes to HTML and HTTP in order to be able to apply our fragment framework directly to the client/server model with or without the intervention of a properly tuned proxy server.

What we are actually looking for is a way that a client would be able to request only portions of an HTML page that would be identified as changed, directly from a Web server. Our future approach consists of three additions to HTML and HTTP. These are:

- The addition of an HTML tag that would be used as a separator of fragment. This tag would be set by the page administrator.
- An HTTP request method that would cause the transfer of a certain portion of the page. The server would be able to do this with the use of the HTML tag proposed in the first proposed addition. This can be combined with the range requests that have already been proposed.
- An if-modified-since header field for portions of a Web page and not the whole page. This way a client would know whether to request or not a certain portion of a page.

Of course there are other approaches that can satisfy our main goal, which is to be able to request only portions of a Web page. We have shown that this can provide a substantial decrease in redundant data transfers over the Web.

In order to demonstrate the fragment approach, we used the <TABLE> and </TABLE> tags to break down pages. This is an efficient approach, but can surely be enhanced. We intend to use approaches from the semantic Web, RDF and XML in order to be as efficient as possible in page fragmenting. The contemporary HTML approach was used in this work because the vast majority of Web sites have not migrated to current standards such as XML. It would be a lot easier to fragment an XML compliant Web page. We believe that this paper has also shown the need for migration to XML as an inherent characteristic of HTML. Web page developers need standardized technologies to structure and inter-relate their Web data. The efficient and repeated use of the <TABLE> tag in Web pages is a reality that proves this.

In this work we have used very simple statistical definitions of fragment change probability and fragment relative change probability. We believe that the use of ideas and results based on related work, such as [4], will considerably improve our results and especially the first technique proposed in our solutions. It has been proved that Web page changes follow a Poisson model. We intend to study this result in the context of fragments and find if this assumption stands.

In our future work we also aim at studying our solutions in relevance with the required client requests. The Client Side Includes solution that we have proposed causes the additional requests for these includes. If the network is congested, problems may occur. Even so we must also consider the requests caused by images and other included components. A study on the effect of the Client Side Includes to network traffic will surely be part of our future work.

In this work we have used only the element of table nesting to extract fragment relations. Semantic and content based relations must also be investigated. It is clear, if one considers the change patterns of a page that fragments are related not only by their place on the page but also by the meaning of their content. This is an area where the semantic Web concept can be very useful.

Finally, we aim at measuring the efficiency of our proposed solutions under real Web traffic conditions. We aim at using proxy server log files to replay Web request patterns with the use of our proxy algorithms. This way we will be able to extract more concrete results on our algorithms' efficiency.

We also aim at using different ranking algorithms to enhance the second technique proposed in this paper.

8. CONCLUSIONS

In this paper we have proved that redundant data transfers are carried out on the Web, together with required data transfers, in the context of Web pages. Our research was mainly triggered by the observation that only some portions of Web pages change every time we request them. We have quantified the amount of redundant data transferred, through a specific example, and have showed that it is substantial. The solutions that we have proposed have shown that proxy servers, Web servers and Web crawlers can benefit from treating pages through a fragment approach. The techniques proposed in this paper show that by applying a purely statistical approach under a fragment scheme, a proxy server can efficiently deal with Web page changes and at the same time substantially reduce the amount of redundant data transferred over the Web.

Another basic conclusion of this paper is that the migration to XML can substantially reduce the amount of stale data that are moved over the Web. XML inherently provides data structuring and relation mechanisms that can easily be manipulated to provide information on specific data changes.

In this work we have concluded that there are two basic reasons causing redundant data transfers on the Web. The first reason is the inefficient estimation of Web page change frequencies. If the change frequencies could be estimated accurately, the problem of redundant data transfers would be constrained to redundant data transferred together with useful or 'fresh' data in Web pages. This brings us to the second cause of redundant data transfers, which is the nature of the HTTP protocol. The HTTP protocol deals with Web pages as entities. As shown, this causes redundant data transfers.

The techniques that we propose in this paper aim at recognizing the redundant transfers in a Web page in order to prevent the redundant portions of data from being transferred over the Web. The procedure of decomposing Web pages into fragments and the recognition of the redundant fragments require certain processing resources. In this paper we propose the methodology that can be followed in order to stop redundant data transfers. The whole procedure of fragmentation, redundant fragment recognition and decision making can be serviced on a Web server, on a Proxy or even on the client machine. The actual trade-off here has to do with server resources versus network utilization. The proposed technique does not add to response time (on average) since (as shown in Section 6 of the paper) the techniques that we propose rely on historical data. As time goes by the algorithms that we propose are more accurate. The only computation procedure that needs to be executed is actually very simple for Web servers. The performance of Web servers in our days can easily cope with a service such as the one we propose. We must not forget that the gain in response time by the proposed system is much more important than any loss in response time due to computation. Of course, server performance is always an issue and can be monitored in order to quantify possible problems. This will be part of future work. In this paper we only aim at making the problem of redundant data transfers clear and introducing possible solutions. These solutions may have side effects but since computational resource costs (memory, CPU, etc.) are declining, it is clear that with a specific server configuration our service may significantly improve response times.

It is certain that the problem discussed in this paper must be treated from all aspects of the client/server architecture. We have proposed a solution that can be integrated, with minor changes, into all Web servers, proxies but also Web crawlers. Our solution can surely be further enhanced and adjusted to current Web trends and techniques.

REFERENCES

1. Brewington BE, Cybenko G. How dynamic is the Web? *Proceedings of the 9th World-Wide Web Conference*, 2000.
2. Douglis F, Feldman A, Krishnamurthy B. Rate of change and other metrics: a live study of the World Wide Web. *USENIX Symposium on Internetworking Technologies and Systems*, 1999.
3. Brewington BE, Cybenko G. Keeping up with the changing Web. *IEEE Computer Magazine* 2000; **33**(5):52–58.
4. Cho J, Garcia-Molina H. Estimating frequency of change. *Technical Report*, Stanford Database Group, 2001-09-22.
5. Taylor HM, Karlin S. *An Introduction to Stochastic Modeling* (3rd edn). Academic Press: New York, 1998.
6. Bouras C, Konidaris A. Web components: a concept for improving personalization and reducing user perceived latency on the World Wide Web. *The 2nd International Conference on Internet Computing* (*IC2001*), Las Vegas, Nevada, USA, 25–28 June 2001.
7. Wills CE, Mikhailov M. Towards a better understanding of Web resources and server responses for improved caching. *Proceedings of the Eighth World-Wide Web Conference*, 1999.
8. Wills CE, Mikhailov M. Studying the impact of more complete server information on Web caching. *5th International Web Caching and Content Delivery Workshop*, Lisbon, Portugal, 22–24 May 2000.
9. Web service at Octopus.com found at http://www.octopus.com/
10. Lambrinidis A, Roussopoulos N. On the materialization of WebViews. *Proceedings of the ACM SIGMOD Workshop on the Web and Databases* (*WebDB '99*), Philadelphia, Pensylvania, USA, June 1999.
11. Challenger J, Dantzig P, Dias D, Mills N. Engineering highly accessed Web sites for performance. In *Web Engineering*, Deshpande Y, Murugesan S (eds). Springer-Verlag: Berlin.
12. Challenger J, Iyengar A, Witting K. A publishing system for efficiently creating dynamic Web content. *INFOCOM 2000*, Tel Aviv, Israel, 26–30 March 2000.
13. Bouras C, Konidaris A. Run-time management policies for data intensive Web sites. *International Workshop on Web Dynamics* (in conjunction with the *8th International Conference on Database Theory*), London, U.K., 3 January 2001; 1–10.
14. Arlitt M, Friedrich R, Jin T. Performance evaluation of Web proxy cache replacement policies. *HP Technical Report HPL-98-97R1*, *991122*, External (Available at http://www.hpl.hp.com/techreports/98/HPL-98-97R1.pdf.)

15. Habib Md A, Abrams M. Analysis of sources of latency in downloading web pages. *WebNet 2000*, San Antonio, Texas, U.S.A., October 30–November 4, 2000.
16. Tong SL, Bharghavan V. Alleviating the latency and bandwidth problems in WWW browsing. *Usenix Symposium on Internet Technologies and Systems '97*, Monterey, CA, December 1997.
17. Palpanas T, Krishnamurthy B. Reducing retrieval latencies in the Web: the past, the present, and the future. *Technical Report CSRG-378*, Graduate Department of Computer Science, University of Toronto.
18. Cho J, Garcia-Molina H. Synchronizing a database to improve freshness. *Proceedings of the 2000 ACM SIGMOD*, 2000.
19. Challenger J, Iyengar A, Dantzig P. A scalable system for consistently caching dynamic Web data. *Proceedings of IEEE INFOCOM'99*, New York, March 1999.

AUTHORS' BIOGRAPHIES

Christos Bouras obtained his Diploma and PhD from the Computer Science and Engineering Department of Patras University (Greece). He is currently an Associate Professor in the above department. Also he is a scientific advisor of Research Unit 6 in Research Academic Computer Technology Institute (CTI), Patras, Greece. His research interests include Analysis of Performance of Networking and Computer Systems, Computer Networks and Protocols, Telematics and New Services, QoS and Pricing for Networks and Services, e-learning, Networked Virtual Environments and WWW Issues. He has extended professional experience in Design and Analysis of Networks, Protocols, Telematics and New Services. He has published 150 papers in various well-known refereed conferences and journals. He is a co-author of five books in Greek. He has been a PC member and referee in various international journals and conferences. He has participated in R&D projects such as RACE, ESPRIT, TELEMATICS, EDUCATIONAL MULTIMEDIA, ISPO, EMPLOYMENT, ADAPT, STRIDE, EUROFORM, IST, GROWTH and others. Also he is member of, experts in the Greek Research and Technology Network (GRNET), Advisory Committee Member to the World Wide Web Consortium (W3C), IEEE Learning Technology Task Force, IEEE Technical Community for Services Computing WG 3.3 Research on Education Applications of Information Technologies and W 6.4 Internet Applications Engineering of IFIP, Task Force for Broadband Access in Greece, ACM, IEEE, EDEN, AACE and New York Academy of Sciences.

Agisilaos Konidaris obtained his Diploma and Masters Degree from the Computer Engineering and Informatics Department of the Patras University in Greece. He is currently an R&D Computer Engineer and in attending his final year of his postgraduate studies. His research interests include Computer Networks, Telematics, Distributed Systems, Multimedia and Hypermedia. He is especially interested in the design, pricing and distribution of innovative Network Services. He has published several papers in well-known refereed conferences and journals. He is also co-author of two books (one with subject Multimedia and Computer Networks and one with subject Special Network Issues). He has participated in several R&D projects while working for the Research Academic Computer Technology Institute in Patras, Greece. He is also the Managing Director of kefalonia.net.gr, a news and tourist information portal.

