End-to-end data deduplication for the mobile Web

Ricardo Filipe
Instituto Superior Técnico
Technical University Lisbon/INESC-ID
Email: rfilipe@gsd.inesc-id.pt

João Barreto
Instituto Superior Técnico
Technical University Lisbon/INESC-ID
Email: joao.barreto@inesc-id.pt

***Abstract*—The emergence of affordable mobile devices with rich interfaces and high-bandwidth wireless connectivity has revolutionized the mobile Web. However, such new trends also imply downloading larger data volumes from the Web, with considerable battery and, often, monetary costs that inevitably degrade user experience.**

The mobile Web calls for end-to-end data deduplication that is able to achieve both high precision and negligible computational cost on the battery-constrained client side. We propose dedupHTTP, a novel deduplication solution that leverages the generic approach of Cache-Based Compaction to achieve the above requirements. Using a full-fledged implementation of dedupHTTP with real workloads from popular Web sites, we obtained savings in traffic consumption of up to 94,5% when comparing to plain HTTP transfer.

I. Introduction

The Web has had a major growth in recent years. While the volume of data available through the Web is doubling every year [1], the average Web page size and number of objects per page are steadily increasing [2]. We resort to Web-based services for more and more tasks that we used to perform offline. Examples range from Web-based mail, home banking and online news, to document editing and social networking.

The emergence of affordable mobile devices allows one to perform such web-based tasks anywhere, through smartphones, handheld and laptop PCs. With the evolution of the mobile world, mobile users have started to access increasingly richer Web content, downloading more and more information.

However, such new trends of the mobile Web come at a cost, often twofold. First, many wireless Internet providers charge a per-byte monetary cost. Second, battery autonomy is not growing as fast as the number of bytes that devices exchange across power hungry wireless connections [3], [4], [5].

Hence, if we want to deliver a rich Web surfing experience to mobile users, we need to dramatically reduce the actual payload that mobile devices download across wireless links. One natural direction to tackle such a challenge is to exploit the substantial redundancy that Web contents exhibit [6].

Web *deduplication* is not a new problem, as well established techniques such as data compression [7] and classical Web caching [8] solve it to some extent. The former eliminates redundancy within the resource being downloaded, whereas the latter avoids downloading a resource that is entirely identical to a previous version present in the client's cache.

This work was partially supported by FCT (INESC-ID multi-annual funding) through the PIDDAC Program funds and by the projects LSDMOG (PTDC/EIA-EIA/113993/2009) and Byzantium (PTDC/EIA/74325/2006)

However, there is strong evidence that a considerable amount of redundancy within downloaded web contents arises from other forms of redundancy, which escape both data compression and classical Web caching. The most common example is when a resource changes little by little over time, which is typically called *resource modification* [6].

To address the Web deduplication problem, Rhea et al. have proposed Value-Based Web Caching (VBWC) [9], subsequently improved by Irmak and Suel [10]. Their results show that, by detecting resource modification and other redundant data that classical caching and data compression do not, they achieve substantial network consumption savings [9], [10].

One can consider employing VBWC in the mobile scenario as a means of alleviating the battery consumption of increasingly energy-demanding clients. However, some characteristics of VBWC would be crucial limitations when deployed in the mobile Web scenario. Firstly, VBWC relies on a dedicated middlebox proxy deployed at the client's ISP, which intermediates all connections of the client with any Web server. This assumption may not always be a reasonable one for the mobile scenario. Mobile clients connect to distinct access points as they move through different geographic locations, which often causes them to roam across different ISPs.

A second disadvantage of the middlebox based approach is that it cannot exploit redundancy in page loads that are served through an HTTPS connection. Finally, VBWC requires the client to maintain a hash table of every resource block it currently caches, as well as to perform look ups to such a table in order to decode the deduplicated contents it receives from the proxy. On devices with severe battery constraints, such load has an energy cost that should be far from negligible.

In this paper we propose a novel deduplication system for the battery-constrained mobile Web, called *dedupHTTP*[1]. dedupHTTP eliminates the above limitations of VBWC when deployed in such a scenario, while retaining comparable deduplication effectiveness. The key insight behind our solution is the design of a (i) purely end-to-end [11] deduplication system (without relying on any intermediate middlebox proxy) which (ii) pushes all redundancy detection steps to the server. Requirement (i) makes dedupHTTP well suited to roaming clients and allows it to work even with encrypted communication between client and server. Requirement (ii) means that dedupHTTP places minimal computational load on the client.

With that in mind, we depart from the Cache-Based Com-

[1] dedupHTTP's open source is available at http://deduphttp.sourceforge.net

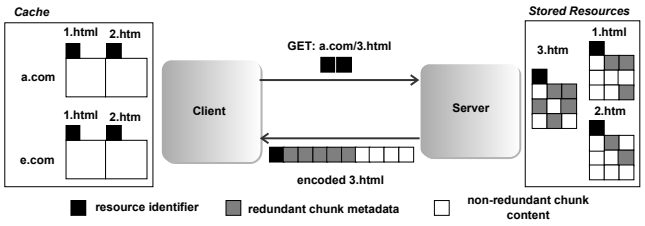

Fig. 1. Example of HTTP request using dedupHTTP

paction (CBC) approach [12], which inherently ensures requirements (i) and (ii), and we propose non-trivial extensions to CBC that allow us to build a complete and practical solution.

More precisely, the main contributions of this paper are:

- We propose an efficient and scalable solution for the problem of allowing the server to track each client's cache contents, a previously unsolved problem in CBC [12].

- We build a full-fledged deduplication system for the mobile Web. DedupHTTP shows savings of network consumption of up to 94,5% when comparing to plain HTTP transfer.

The remainder of the paper is organized as follows. Section II describes dedupHTTP's architecture, departing from the generic CBC approach and then detailing the improvements and modifications that dedupHTTP introduces. Section III evaluates an implementation of our algorithm. Section IV surveys related work. Finally, Section V draws some conclusions.

II. Architecture

DedupHTTP runs in two separate modules, one inside the client browser and another inside the Web server. The server maintains a set of resources (e.g., html pages), each one addressable by a given URL.

Whenever the server creates a new version of a resource that the server already stores, the server archives the previous version. Hence, the server actually stores a set of resource versions for each URL, the current version together with older ones. This increases the system's ability to detect redundancy.

Upon receiving an HTTP request for a given URL, the server responds with the contents of the most recent resource version the server currently stores for that URL. The client stores a cache with the resources it downloaded from any servers, indexed by the resource's URL.

At a glance, dedupHTTP follows the generic CBC approach [12], which we describe next in three main steps, which we call A, B and C. Figure 1 depicts the whole algorithm.

A. Step A: Determining common resources

Each resource version is indexed by an unique identifier, assigned by the server on the first time any client downloads that resource version. Such an identifier is given by a *(host domain, serial number)* pair. When the client sends the HTTP request for a given URL, it fetches all identifiers from the resources stored in the client's cache that belong to the same host domain as the requested resource. These are the resources the server will use as reference resources.

In order to ensure that the resources identified on a request are not evicted, the client locks the affected cached resources until the response arrives or the server request times out.

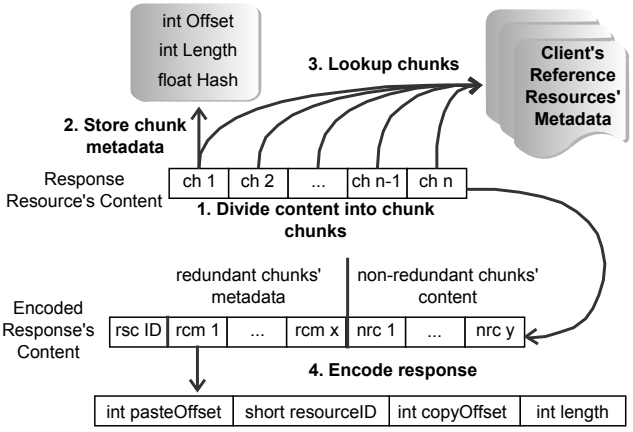

Fig. 2. Schematic of the server-side redundancy detection algorithm

B. Step B: Server-side redundancy detection

When the server receives a request for some URL, it locates and fetches the most recent version of the target resource. Then, the server determines which resources it currently stores in common with the client's cache, by crossing the resource identifiers present in the HTTP request with the ones in its cache. We call these resources the *reference resources* for the target resource. Finally, we run a local redundancy detection algorithm to find chunks in the target resource that are redundant with chunks of the reference resources. These chunks don't need to be transferred and can be replaced in the response by references that the client will understand.

1) Chunk division algorithm: We start by creating per-byte hashes of the resource content. These hashes represent smaller chunks of the resource. This is done by using the rolling hash function of Karp-Rabin fingerprinting [13].

Now we have to select the hashes that represent the resource. We chose the Winnowing [14] technique, since it has been experimentally proven to give the best redundancy detection [15]. To avoid pathological cases, we enforce a minimum and maximum chunk size. After selecting the chunk boundaries, we hash each larger chunk using a 64 bit MurmurHash [16]. This choice was driven by the fact that MurmurHash outperforms cryptographic hash functions such as MD5 or SHA1, while retaining a low collision rate [16].

2) Detecting redundant chunks: The meta-information returned after applying the chunk division algorithm to the target resource (chunk hash, offset within the resource and length) is stored in a hash table indexed by the chunk's hash.

The server iterates through all chunk hashes of the target resource searching for redundant chunks with its reference resources and within itself (Figure 2). It gets each reference resource's chunk hash table from a resource pool indexed by the resource identifier, and for each chunk hash, the server looks it up in the hash table of each reference resource.

C. Step C: Sending the final response

The final response, to be sent by the server to the client, begins with a metadata section with a sequence of chunk references to the redundant chunks found between the target resource and its reference resources.

Each chunk reference is a four-part tuple: the offset in the current response where the chunk is to be appended, the

	32	64	128	256	512	1024	2048
CNN	68,8%	81,9%	81,8%	79,3%	76,3%	71,4%	65,7%
Engadget	46%	57,6%	50,7%	38,5%	32,4%	25,4%	22,3%
HuffPost	58,7%	69,6%	66,6%	60,7%	54,7%	47,1%	33,7%

TABLE I
REDUNDANCY DETECTED VARIATION WITH DEDUPHTTP

	64	128	256	512	1024
CNN	93,9%	94,5%	94,4%	93,8%	92,4%
Engadget	87,6%	88,5%	88,9%	88,4%	87,3%
HuffingtonPost	91%	91,6%	91,4%	89,9%	88,3%

TABLE II
REDUNDANCY DETECTED VARIATION WITH SEVERAL CHUNK SIZES FOR DEDUPHTTP+COMPRESSION

resource identifier of the reference resource where the client can find the chunk content, the offset in the reference resource where the chunk content can be found and the length of the chunk. The tuples are arranged in the same order as the corresponding chunks appear in the original response. At the end of the metadata section we append the non-redundant response content chunks, in the same order as they were present in the original response.

When the client receives the response it only has to go through the metadata, copy the chunks referenced by the tuples to the final response, from the locally cached resources, and copy the non-redundant content from the received response to the final response, in the corresponding order.

1) Metadata coalescing optimization: We have observed that most times there are consecutive redundant chunks detected in the same reference resource. Since we check the client's resources in the same order for every chunk, we can save on metadata by considering a sequence of consecutive redundant chunks as a single chunk. Hence, we only send one metadata block for the large coalesced redundant chunk.

A particular case where such an optimization is very effective is when a requested resource is aliased in the domain. The only thing that will be sent to the client is a chunk's worth of metadata with the whole resource length.

III. EVALUATION

For evaluation purposes we developed prototypes of dedupHTTP's client and server modules. For ease of implementation, these modules were implemented as proxies, using the OWASP Proxy [17] library, which takes care of every aspect in HTTP communication.

We also implemented a delta-encoding algorithm to test against dedupHTTP. We store each requested resource on both the client and server. When a request is made for a new version of a resource the client has, we encode a delta between those two versions at the server. Then we send only the delta to the client, who reconstructs the new resource using the old resource version and the delta. We used xdelta3 [18] as the encoding/decoding algorithm.

The core question we aim to answer with dedupHTTP's evaluation is: how much traffic does dedupHTTP save for the client? From here we expect to impact the client's battery life and its other resources in a meaningful way.

Regular browsing still represents a considerable amount of Internet traffic [1]. The workloads for our experiments were created by downloading every news and comments page linked from the main pages of three worldwide popular sites: www.cnn.com, www.engadget.com and www.huffingtonpost.com.These pages are good representatives of dynamic content sites. We discard all image resources from the workloads, leaving only plain text, html, xml and javascript resources. We downloaded these resources on two consecutive

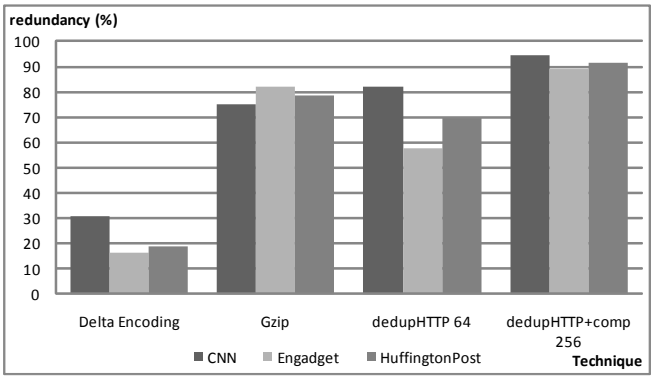

Fig. 3. Average redundancy for several techniques

days at the same hour in order to compare the redundancy retained from one day to the other, mimicking a regular user that reads the news once every day.

All results here shown average the downloading of a workload's resources from our Web server to our Web client. The techniques we compared dedupHTTP to were delta-encoding, plain HTTP transfer with and without Gzip encoding enabled and an hybrid dedupHTTP with Gzip compression algorithm.

A. Minimizing transferred volume of data

We evaluated different expected chunk sizes, ranging from 32 bytes up to 2048 bytes (2 KB). Table I presents our results. The smaller the chunk size, the more metadata the server has to store, since we have more chunks per resource.

There is a threshold below which the attained precision does not compensate the increased metadata overhead. This is the case when we have a 32 byte chunk size, since there can be chunks down to 8 bytes, because of the chunk size lower bound. Clearly the lowest feasible expected chunk size is 64 bytes, because then we will have a minimum size of 16 bytes for the chunk size, which is larger than the communication overhead per chunk of our implementation (14 bytes).

We also tested our hybrid algorithm with chunk sizes ranging from 64 to 1024 bytes (1 KB) (Table II).

The redundancy detected with the hybrid is better for 128 and 256-byte chunk sizes, because of fewer metadata while still detecting most of the redundant data from other resources.

B. Overall transferred volume results

We selected dedupHTTP with 64-byte chunks and hybrid with 256-byte chunks to represent those techniques against the competition, since they yielded the best results.

When comparing redundancy detection of all the techniques (Figure 3), delta-encoding yields the worse results for all workloads, since it can act only on part of the resources, the ones with a previous version in the client's cache.

Gzip compression values are very consistent over all resources and workloads. For the CNN workload, dedupHTTP is better than Gzip, but not for the other two workloads.

This happened because we witnessed the number of resources prone to delta-encoding being bigger on the CNN workload. These are the resources where most redundancy will be found with dedupHTTP, since there is a previous version of the resource in cache with much redundant content.

Finally, the hybrid algorithm surpasses all of the other techniques, with 6% to 19% more redundancy detected than Gzip, which should indicate the amount of data retrieved from resources other than the one being requested, i.e. the amount of redundant data which dedupHTTP detects and Gzip doesn't.

We have also studied the variation of redundancy detected over time, with workloads for several consecutive days, and dedupHTTP's effect in the Time-to-Display for the user over the Internet. These results are not presented here for space limitations, but are available elsewhere [19].

IV. Related work

Chan and Woo proposed the generic Cache-Based Compaction (CBC) approach [12]. It combines two main ideas: a selection algorithm to choose reference objects for redundancy detection, and an encoding/decoding algorithm that acts upon a new object using the selected reference objects. They proposed a dictionary-based solution for encoding/decoding. When there are no common resources across the client's cache and the server storage, CBC acts as traditional data compression. While when there is only one common resource, CBC behaves similarly to delta encoding. Their original solution does not scale well to more than a few common resources (they consider a maximum of three resources, including the object being downloaded). In contrast, dedupHTTP scales gracefully to higher numbers of common resources. CBC's proposal [12] is incomplete, as crucial aspects are left unsolved. For instance, the problem of expressing the client's cache contents to the server in an efficient way was not addressed. CBC also has a security problem in the presence of malicious clients, which we have also solved but could not explain here due to space limitations [19].

Rhea et al. devised Value-Based Web Caching (VBWC) [9], a solution that provides deduplication for clients connected across low bandwidth links (less than 80 Kbps). VBWC relies on an ISP proxy. VBWC's proxy maintains, for each client, a loosely synchronized set of the hashes of the chunks that such client is caching. The proxy does not store the actual data, which ensures the scalability of VBWC, as long as chunk size remains at coarse levels (2 KBytes in [9]).

When a server sends some resource to the proxy, as a response to a given client's request, the proxy divides such response in chunks and checks if that client already holds each chunk. If so, it only transfers the chunk's hash. The client, in turn, maintains a local chunk hash table that indexes every chunk the client currently caches. For each chunk hash the client receives from the proxy, the client looks up the local hash table to confirm whether that chunk is effectively cached. Irmak and Suel [10] further improved VBWC by employing an hierarchical redundancy detection scheme, enabling smaller chunks while retaining acceptable storage and network overheads. Both solutions suffer from important limitations when deployed in the mobile Web scenario. Namely, their reliance on an ISP proxy and look up load that both place on the battery-constrained client side.

V. Conclusions

The new trends of the rich-content mobile Web imply downloading larger data volumes from the Web than before. While wireless bandwidth is keeping up with the pace of such a revolution, owners of battery-constrained devices are experiencing shorter device autonomy and, often, paying higher bills when connected through access points that charge them per each transferred byte.

We propose dedupHTTP, a deduplication solution that leverages the generic approach of Cache-Based Compaction to fight the problems above. Using a full-fledged implementation of dedupHTTP with real workloads from popular Web sites, we obtained savings in network consumption of up to 94,5% when comparing to plain HTTP transfer.

References

[1] 2010. [Online]. Available: http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html

[2] J. Charzinski, "Traffic properties, client side cachability and cdn usage of popular web sites," in *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*, B. Mller-Clostermann, K. Echtle, and E. Rathgeb, Eds., 2010, vol. 5987, pp. 136–150.

[3] J. Flinn, E. d. Lara, M. Satyanarayanan, D. S. Wallach, and W. Zwaenepoel, "Reducing the energy usage of office applications," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, ser. Middleware '01, 2001, pp. 252–272.

[4] A. Rice and S. Hay, "Measuring mobile phone energy consumption for 802.11 wireless networking," *Pervasive and Mobile Computing*, vol. 6, pp. 593 – 606, 2010, special Issue PerCom 2010.

[5] S.-L. Tsao and C.-H. Huang, "A survey of energy efficient mac protocols for ieee 802.11 wlan," *Computer Communications*, vol. 34, pp. 54 – 67, 2011.

[6] J. C. Mogul, F. Douglis, A. Feldmann, and B. Krishnamurthy, "Potential benefits of delta encoding and data compression for http," in *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, 1997, pp. 181–194.

[7] D. Lelewer and D. Hirschberg, "Data compression," *ACM Computing Surveys*, pp. 261–296, 1987.

[8] J. Wang, "A survey of web caching schemes for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 29, pp. 36–46, 1999.

[9] S. C. Rhea, K. Liang, and E. Brewer, "Value-based web caching," in *Proceedings of the 12th international conference on World Wide Web*, ser. WWW '03, 2003, pp. 619–628.

[10] U. Irmak and T. Suel, "Hierarchical substring caching for efficient content distribution to low-bandwidth clients," in *Proceedings of the 14th international conference on World Wide Web*, 2005, pp. 43–53.

[11] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Trans. Comput. Syst.*, vol. 2, pp. 277–288, 1984.

[12] M. C. Chan and T. Woo, "Cache-based compaction: a new technique for optimizing web transfer," ser. INFOCOM, 1999, pp. 117–125.

[13] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Dev.*, vol. 31, pp. 249–260, 1987.

[14] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of ACM SIGMOD international conference on Management of data*, 2003, pp. 76–85.

[15] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: findings and implications," in *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '09, 2009, pp. 37–48.

[16] A. Appleby, "Murmurhash 2.0," 2009. [Online]. Available: http://sites.google.com/site/murmurhash/

[17] R. Dawes, "Owasp proxy," 2010. [Online]. Available: http://www.owasp.org/index.php/Category:OWASP_Proxy

[18] J. Macdonald, "xdelta3 vcdiff implementation," 2002. [Online]. Available: http://tools.ietf.org/html/rfc3284

[19] R. Filipe and J. Barreto, "End-to-end data deduplication for the mobile web: extended report," INESC-ID, Tech. Rep. 28, June 2011.