# Context-Aware Encoding & Delivery in the Web

Benjamin Wollmer[1][0000−0002−0545−8040], Wolfram
Wingerath[2][0000−0003−3512−5789], and Norbert Ritter[1]

[1] University of Hamburg, Hamburg, Germany
`{wollmer, ritter}@informatik.uni-hamburg.de`
[2] Baqend GmbH, Hamburg, Germany
`wolle@baqend.com`

**Abstract.** While standard HTTP caching has been designed for static resources such as files, different conceptual extensions have made it applicable to frequently changing data like database query results or server-generated HTML content. But even though caching is an indispensable means to accelerate content delivery on the web, whether or not cached resources can be used for acceleration has always been a binary decision: a cached response is either valid and can be used or has been invalidated and must be avoided. In this paper, we present an early-stage PhD project on a novel scheme for content encoding and delivery. Our primary goal is minimizing the payload for client requests in the web by enabling partial usage of cached resources. We discuss related work on the topic and analyze why existing approaches have not been established in practice so far, despite significant gains such as reduced bandwidth usage and loading times for end users. We then present open challenges, derive our research question, and present our research goals and agenda.

**Keywords:** web caching · efficiency · compression algorithms · delta encoding · benchmarking · runtime optimization · user experience.

## 1 Introduction

In the web, performance is crucial for user satisfaction and business-critical metrics such as conversion rate or revenue per session [17]. But even though new devices and browsers are being developed year after year, the principles of data transfer in the web seem stuck. To illustrate this phenomenon, consider how traditional web caching is used in practice. For decades, the browser's decision whether to use a cache or to load a resource via network has been determined by (1) whether the resource's identifier is present in the cache, (2) whether it has been invalidated already, and (3) whether its cache lifetime (time to live, TTL) is still valid. However, this procedure is inefficient in certain scenarios. For example, state-of-the-art caching approaches do not make use of cache entries that are *similar* (but not exact matches) to a requested resource. Invalidated resources are discarded entirely, even if only a minor update occurred and major portions of the invalidated resource could be reused in theory. As another example, compression algorithms are typically not chosen based on runtime parameters, even though information such as available bandwidth or processing power may well be

performance-critical factors for choosing the right method in a given situation. Intuitively, though, performance and efficiency could be improved, if parameters such as image resolution or compression codec were chosen based on whether the user is on a high-bandwidth fiber cable or an unstable mobile connection with limited data allowance. We think these inefficiencies in the web content delivery chain are mere implementation artifacts that can be removed in practice. We intend to prove it.

This paper describes our research and development agenda to achieve this goal. In Sections 2 and 3, we briefly survey the state of the art in content encoding and delivery, list the most critical open challenges, and formulate our research question. In Section 4, we then present the main research goals of this PhD project and a brief outline of our research agenda. Section 5 concludes the paper.

## 2   Content Encoding: State of the Art

Even though introduced in the early 1990s, the purely text-based Gzip is still the most widely used compression method in the web: As of 2019, it is used for more than 30% of all mobile requests and for over 40% of requests for text-based web resources overall [4]. While more efficient approaches have been developed in the last decades, none of them has become as widespread in practice as Gzip which is natively supported by all relevant browsers and web servers today.

Classic compression algorithms like Gzip remove redundancy within individual transmissions, thus increasing efficiency compared with transmitting uncompressed raw data. However, redundancy between different transmissions is not addressed, so that requesting two very similar versions of the same resource will essentially double the transmitted amount of data, compared with requesting only one. **Delta encoding** addresses this weakness by transferring only changes whenever a prior version of the requested resource is already known to the client. Since changes in websites (HTML) and web assets (e.g. stylesheets or JavaScript files) are often small, delta encoding can have a significant impact on page load times: A study from the late 1990s [11] showed potential savings of more than 80% of transferred bytes for text-based web content.

There were plans to standardize delta encoding [10] and even full-fledged user-facing implementations (e.g. [9]). All these efforts failed in practice, though, because calculating deltas fast enough to be used by requesting clients turned out prohibitive. To the best of our knowledge, the only commercial implementation of delta encoding is Cloudflare's **Railgun™** which merely optimizes data transfer between web servers and the Cloudflare CDN. Since end users still request their web content through standard HTTP requests in this scheme, however, they do not profit from delta encoding directly [5]. As another limitation, Railgun™ is reportedly difficult to deploy and operate: According to architectural lead Nick Craver [6], Railgun™ had been evaluated at Stack Exchange for over a year, but was eventually canceled as the deployment never became stable.

Another way to exploit similarities between data entities is **shared dictionary compression** [15]. As the basic idea, client and server share a common

dictionary, so that portions of the payload can be encoded as references to dictionary entries instead of the actual content. As the only implementation we are aware of, Google's **SDCH** (pronunciation: "sandwich") [3] was supported by Chrome-based browsers and tested at LinkedIn where it outperformed Gzip compression by up to 81% [13]. Unfortunately, though, support was never added to other browsers and was eventually removed from Chrome [14], because virtually no website provider apart from LinkedIn overcame the high technical complexity of computing and maintaining shared dictionaries. To address this challenge, Google developed **Brotli** [1] compression as a derivative of SDCH where the dictionary is shipped with the library itself and does not have to be tailored to individual websites. Brotli's dictionary is still tuned for web content[3], but generic enough to be used across different websites. While this makes it more widely applicable than SDCH, it also bars Brotli from exploiting frequently occurring page-specific strings that would be efficiently encoded in SDCH.

Choosing a compression method or **compression level** always is a trade-off between minimizing computation time and minimizing transmitted bytes [8]. But even though the sweet spot depends on dynamic parameters such as the available computing power of both parties and the bandwidth between them, modern web servers typically use static heuristics like "always use default compression levels" or "use Brotli when available and Gzip otherwise" [4].

There are many **other evolutionary optimizations** to content encoding and delivery mechanisms such as HTTP/2 (incl. server push, header compression, pipelining, multiplexing) [2] or eTags [12, Sec. 14.19], and even advanced approaches for caching dynamic data [7]. However, new technologies are often adopted slowly, because they are complex (and thus expensive) to integrate with legacy architectures or because they are only supported by a relatively small share of the end users' devices and browsers.

## 3   Open Challenges & Research Question

Today's web infrastructure relies on technology that is several decades old. While more advanced compression algorithms than Gzip do exist, none of them has gained broad adoption. Delta encoding and other advanced approaches have failed, because they are hard to deploy or not noticeably useful for end users. Summing up, we see several critical challenges for content delivery in the web:

$C_1$ **Lack of Client Support.**   While delta encoding with Railgun™ optimizes communication between backend servers and the CDN, it does not provide an actual (noticeable) benefit for users of an enhanced website. Approaches that do improve performance for users significantly, in contrast, typically also rely on browser support for broad adoption and cannot succeed without it. The history of SDCH illustrates this dilemma.

$C_2$ **Lack of Backend Support.**   Disregarding support for end users, just implementing the backend for advanced technologies already is a major challenge: The technical complexity alone can be prohibitive (cf. Railgun™ at

---

[3] Brotli's dictionary contains frequent terms from natural and programming languages.

Stack Exchange), but even with that resolved lack of third-party support can still cause a project to fail (cf. SDCH at LinkedIn).

$C_3$ **Inefficient Cache Usage.**  State-of-the-art caching discards a data item entirely as soon as it expires or is invalidated by an ever so slight change. Delta encoding exploits similarities between the current and expired/invalidated versions of the same entity, but no current approach exploits similarities between *different* entities: Requesting two similar resources (e.g. two different product pages in a web shop) always means transmitting highly redundant data.

$C_4$ **Inflexible Protocol Negotiation.**  Compression protocols and their parameterization are typically selected according to static rules, although performance ultimately depends on runtime parameters. For example, CPU-intensive Brotli compression is preferable for a user while on a flaky mobile connection, but using Gzip may be faster as soon as the user comes home and connects to the local Wi-Fi. We consider neglecting the runtime context for performance optimization a major flaw in current technology.

We are convinced that the above challenges can be resolved with a careful end-to-end system design. We therefore set out to address the following **research question**: *How can partial caching and encoding methods be used to accelerate data access in a distributed architecture with heterogeneous clients and servers?*

## 4   Research Goals & Agenda

In order to address these challenges and our research question, we aim to devise a unified system design that enables efficient and context-aware encoding methods, exploits partial cache hits, and builds on widely supported browser and web server features to facilitate widespread adoption. Our research goals are:

$R_1$ **Efficiency Gold Standard.**  To evaluate the potential gains of different encoding methods, we will collect real-world Internet traffic over a period of time and compute the optimal compression savings using hindsight knowledge: Our gold standard encoding will thus work under the unrealistic assumption of perfect knowledge of all relevant factors. We think this will help us assess the maximal possible benefit of our approach in concept and the efficiency of our implementation in practice.

$R_2$ **Pluggable Server-to-Client Content Encoding.**  Two major roadblocks for earlier approaches have been poor support for end users ($C_1$) and high complexity of deployments ($C_2$). To address both these challenges, we will design an extensible architecture for content encoding that only relies on widely available browser features on the user side (cf. $C_1$) and that does not require tight integration with web servers in the backend (cf. $C_2$). We will build our prototype on the JavaScript-based technology Speed Kit [16], because it allows hooking into the client-server communication in a transparent way and because it is supported by more than 90% of all browsers.

**$R_3$  Cross-Entity Delta Encoding.**   Current approaches for delta encoding only exploit similarities between different versions of the same entity, but disregard similarities between different entities (cf. $C_3$). We will develop a storage engine with the ability to exploit the similarity between stored entities: When a certain product page `htmlA` is queried, for example, our intended cross-entity storage engine may not respond with `htmlA` directly, but rather with the information required to construct `htmlA` from information that is already known to the client (e.g. from an old version of `htmlA` or from an entirely different product page `htmlB`).

**$R_4$  Context-Aware Runtime Optimization.**   Our approach will let the storage engine choose a content encoding based on client-provided context information at runtime (cf. $C_4$). From the context information provided in Query 1.1, for example, the storage engine could derive that (1) it may encode `htmlA` as a diff to either `htmlB` or `htmlC` (if that is more efficient than sending the full document) and that (2) Gzip may be preferable over Brotli, because the client has broadband Internet access but only limited CPU power.

```
GET htmlA WITH CONTEXT (
      inClientCache: [htmlB, htmlC]
    , bandwidth: high
    , processingPower: low
)
```

**Query 1.1.** Clients will provide context information with every request, so that our storage engine can choose the most efficient encoding on a per-request basis.

As the first step in our **research agenda**, we plan to evaluate the potential gains of different encoding algorithms ($R_1$). We are going to start with delta and cross-entity encoding as they are pivotal in our research plan. We expect to find that both approaches yield a significant performance uplift, given the unrealistic premise of perfect knowledge. Next, we will devise a client-to-server architecture for web content delivery ($R_2$). One of the critical challenges in our architecture will be the context-aware storage backend ($R_3$). We envision an implementation in different modules for different types of content (e.g. uncompressed text-based content, compressed/bundled scripts, images). To guide and evaluate our efforts, we will benchmark our implementation ($R_4$) against the gold standard ($R_1$) for the theoretically most efficient way of cross-entity encoding.

## 5   Wrapup

Choosing the right encoding for content delivery has a crucial impact on performance in any globally distributed architecture. But while many attempts have been made to establish more efficient content encoding and delivery methods in the web, only few have found widespread adoption. This paper presents an ambitious research plan for addressing this issue. Our basic idea revolves around (1) the client attaching runtime context information to every server request and (2) the server dynamically optimizing every response based on the given context.

To make this practically feasible, we strive for a system design that builds on widely available browser technologies and is easy to integrate for website administrators. While we plan to publish our follow-up research results in the future, we hope to spark interesting discussions on the topic right away.

## References

1. Alakuijala, J., Szabadka, Z.: Brotli Compressed Data Format. RFC 7932 (2016)
2. Belshe, M., Peon, R., M. Thomson, E.: RFC 7540. Hypertext Transfer Protocol Version 2 (HTTP/2) (2015)
3. Butler, J., Lee, W.H., McQuade, B., Mixter, K.: A proposal for shared dictionary compression over http (2008), `https://pdfs.semanticscholar.org/c53e/e3d44f1314c2c4d14dca7d25d1858cf55246.pdf`, accessed: 2020-02-20
4. Calvano, P.: Web Almanac: Compression (2019), `https://almanac.httparchive.org/en/2019/compression`, accessed: 2020-02-20
5. Cloudflare: Optimierung des Ursprungsnetzwerks mit Railgun™ (2018), `https://www.cloudflare.com/website-optimization/railgun/`, accessed: 2020-02-20
6. Craver, N.: HTTPS on Stack Overflow: The End of a Long Road (2017), `https://nickcraver.com/blog/2017/05/22/https-on-stack-overflow`, accessed: 2020-02-20
7. Gessert, F., Schaarschmidt, M., Wingerath, W., et al.: Quaestor: Query Web Caching for Database-as-a-Service Providers. PVLDB (2017)
8. Jarrod: Gzip vs Bzip2 vs XZ Performance Comparison (2015), `https://www.rootusers.com/gzip-vs-bzip2-vs-xz-performance-comparison/`, accessed: 2020-02-20
9. Korn, D., MacDonald, J., Mogul, J., Vo, K.: The VCDIFF Generic Differencing and Compression Data Format. RFC 3284 (June 2002)
10. Mogul, J., Krishnamurthy, B., Douglis, F., Feldmann, A., Goland, Y., van Hoff, A., Hellerstein, D.: Delta Encoding in HTTP. RFC 3229 (January 2002)
11. Mogul, J.C., Douglis, F., Feldmann, A., Krishnamurthy, B.: Potential Benefits of Delta Encoding and Data Compression for HTTP. SIGCOMM Comput. Commun. Rev. **27**(4), 181–194 (Oct 1997). https://doi.org/10.1145/263109.263162
12. Nielsen, H.F., Mogul, J., Masinter, L.M., Fielding, R.T., et al.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Jun 1999). https://doi.org/10.17487/RFC2616
13. Shapira, O.: Shared Dictionary Compression for HTTP at LinkedIn. (2015), `https://engineering.linkedin.com/shared-dictionary-compression-http-linkedin`, accessed: 2020-02-20
14. Sleevi, R.: Shared Dictionary Compression for HTTP at LinkedIn. (2016), `https://groups.google.com/a/chromium.org/d/msg/blink-dev/nQl0ORHy7sw/HNpR96sqAgAJ`, accessed: 2020-02-20
15. White, H.E.: Printed english compression by dictionary encoding. Proceedings of the IEEE **55**(3), 390–396 (March 1967). https://doi.org/10.1109/PROC.1967.5496
16. Wingerath, W., Gessert, F., Witt, E., Kuhlmann, H., Bücklers, F., Wollmer, B., Ritter, N.: Speed Kit: A Polyglot GDPR-Compliant Approach For Caching Personalized Content. In: 36th ICDE 2020, Dallas, Texas, April 20-24, 2020 (2020)
17. Young, J., Barth, T.: Akamai Online Retail Performance Report: Milliseconds Are Critical (2017), `https://www.akamai.com/en/us/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp`, accessed: 2020-02-20