Poster: Co4U: Efficient and Robust HTTP Message Compression for Edge Computing Networks

Shiguang Zhang[1], Jiahao Cao[1], Menghao Zhang[2], Mingwei Xu[1], Wei Han[3], Yanbin Zhai[3], Zhiming Zhang[3]
[1]Tsinghua University [2]Beihang University [3]ByteDance

CCS CONCEPTS

• **Networks** → *Application layer protocols*; • **Computer systems organization** → *Distributed architectures*; • **Information systems** → *Data compression*;

KEYWORDS

Edge Computing; Data Compression; HTTP

ACM Reference Format:
Shiguang Zhang, Jiahao Cao, Menghao Zhang, Mingwei Xu, Wei Han, Yanbin Zhai, Zhiming Zhang. 2024. Poster: Co4U: Efficient and Robust HTTP Message Compression for Edge Computing Networks . In *ACM SIGCOMM 2024 Conference (ACM SIGCOMM Posters and Demos '24), August 4–8, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3672202.3673753

1 INTRODUCTION

Edge computing networks integrate computing, storage, and network resources at network edge nodes to provide infrastructure closer to users, enabling faster data processing, transmission, and service response. An increasing number of companies are embracing this paradigm, e.g., ByteDance has deployed over 500 edge nodes [1] to reduce service response time. In these edge computing networks, most business data transmitted is HTTP messages, which consume a significant portion of bandwidth. For example, according to ByteDance's internal reports, HTTP messages account for over 60% of the network bandwidth. To provide high transmission quality, companies often use proprietary links to connect numerous edge nodes across different regions. Despite effective, these proprietary links usually incur high expenses. Therefore, compressing HTTP messages between edge nodes is essential for saving substantial bandwidth costs.

Currently, various compression algorithms can be used for HTTP messages, including gzip [6], Brotli [4], Shared Brotli [5], SDCH [3], and Zstd [2]. Despite these efforts, achieving a high compression ratio for HTTP messages between edge nodes during transmission remains challenging. The first challenge is to improve the compression ratio for a real-time HTTP message, whose limited content between edge nodes varies and cannot be predicted. Our previous real-world experiments in ByteDance's edge nodes demonstrate that existing lossless compression methods [2–6] generally achieve unsatisfactory compression ratios for a single HTTP message, typically less than 40%. Therefore, we need to develop a method to achieve high compression ratios under these circumstances. The second challenge is to ensure the robustness of this compression method. Since the content of HTTP messages between edge nodes is unpredictable and varies with business needs, the compression ratio may decrease over time. Therefore, we need a method that can adjust the compression strategy to accommodate dynamic HTTP content.

This work was supported by the National Natural Science Foundation of China (Grant 62202260 and 62221003) and the Beijing Science and Technology Plan Project (Grant Z231100010323013). Corresponding Author: Jiahao Cao

ACM SIGCOMM Posters and Demos '24, August 4–8, 2024, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0717-9/24/08.
https://doi.org/10.1145/3672202.3673753

Table 1: Comparison between compressing a real-time HTTP message and compressing a large number of aggregated HTTP messages into a single file.

Algorithms	Average Compression Ratio	
	Single HTTP Message	Aggregated HTTP File
gzip	33.86%	49.51%
Brotli	34.42%	50.04%
Zstd	34.20%	56.33%
SDCH	37.49%	51.43%

While traditional lossless compression algorithms are unsatisfactory, we found that aggregating 38,000 past HTTP messages from ByteDance's real edge computing networks into a single large file for compression significantly improved the compression ratio, increasing it from 33% to more than 50%, as shown in Table 1. Therefore, it may be promising to utilize the *common content among HTTP messages* to improve the compression ratio for a real-time HTTP message.

This paper introduces Co4U, an efficient and robust HTTP message compression system for edge computing networks. As shown in Figure 1, Co4U consists of two main components: one *Co4U manager* and *agents*. The agents compress or decompress a real-time HTTP message based on a pre-trained dictionary that derives from multiple past HTTP messages. The manager periodically samples HTTP messages to pre-train a compression dictionary based on their common content. Meanwhile, it updates and distributes the dictionary based on compression ratio information and sampled HTTP messages for robust compression. Our real-world experiment results indicate that, compared to existing compression methods, Co4U achieves at least a 15% improvement in compression ratio and better robustness for dynamically changing HTTP messages.

2 SYSTEM DESIGN

Pre-trained Compression Dictionary Construction. Based on our previous findings, one possible method to achieve a high compression ratio for HTTP messages is to aggregate multiple messages for compression. However, this approach is impractical in edge computing networks. Different HTTP messages may need to be sent to different edge nodes via different paths, making it hard to accurately

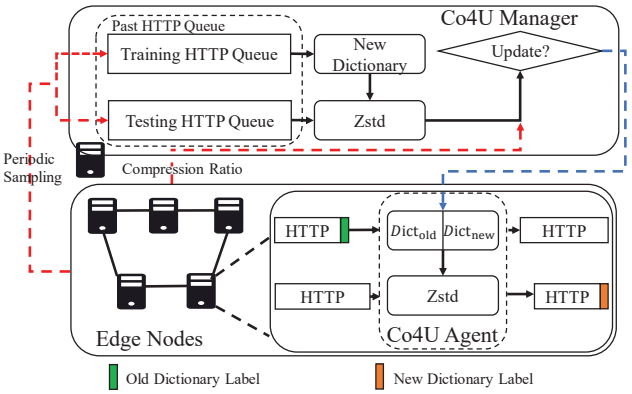

Figure 1: System Overview of Co4U.

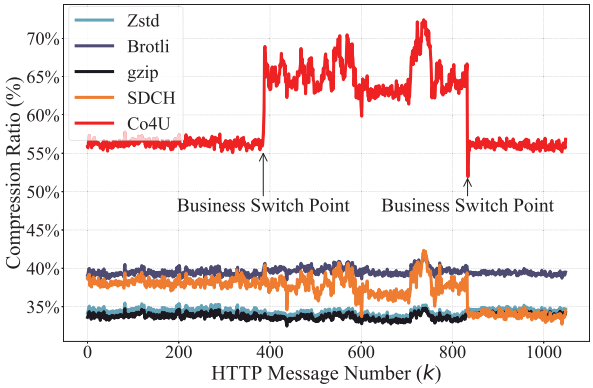

Figure 2: Results from Real HTTP Messages.

aggregate HTTP messages for a specific edge node. Additionally, aggregating multiple HTTP messages for compression can introduce unacceptable latency.

Fortunately, we found that consecutive HTTP messages within a period tend to share similar content patterns. Therefore, rather than aggregating multiple HTTP messages for compression, Co4U utilizes content patterns in past HTTP messages to pre-train a compression dictionary for incoming HTTP messages. Specifically, the Co4U manager, deployed on a server outside the edge computing networks, samples HTTP messages from edge nodes and stores a portion of the sampled messages in an HTTP cache queue. Using Zstd's dictionary training algorithm, the manager generates a compression dictionary from the cached messages. If the dictionary is deemed useful, the manager distributes it to agents, who then use the Zstd algorithm with this dictionary to compress or decompress HTTP messages.

Greedy Dictionary Update. Although the above method can achieve a high compression ratio for a real-time HTTP message using a pre-trained compression dictionary, the dictionary cannot always remain effective in practice. It is inevitable that compression ratios will decrease due to the unpredictable and dynamic nature of future HTTP messages. To address this issue, the Co4U manager continuously retrains and updates the dictionary based on sampled HTTP messages, ensuring high compression ratios.

This way introduces a new question: when should the dictionary be updated? An intuitive method is to set a compression ratio threshold t. If the current compression ratio is below t, the manager would update and distribute a new dictionary. However, this approach has its issues: if t is set too low, the system could become stuck at a local optimum, missing potential bandwidth compression gains. Additionally, setting t too high could lead to overly frequent updates without bringing higher gains.

To find the optimal threshold for updating dictionaries, we propose a simple yet practical solution *greedy dictionary update*. Specifically, the Co4U manager uses a portion of the sampled messages to retrain the dictionary, while another portion is used for evaluating the average compression ratio of the newly re-trained dictionary, denoted as R_{new}. If R_{new} is higher than the compression ratio R_{cur} of the current dictionary used in agents, the manager will distribute the new dictionary to edge nodes, likely achieving a higher compression ratio.

Although we can distribute new compression dictionaries uniformly to all edge node agents through a manager, some in-flight HTTP messages might have been compressed using old dictionaries. If these old dictionaries are not retained, the agents will not be able to decompress these HTTP messages correctly. Therefore, we add an identifier to each compressed HTTP message to represent the dictionary used, i.e., the SHA-256 hash of the dictionary. When a compressed HTTP message reaches its destination edge node, the agent on that node will accordingly select the correct dictionary for decompression based on the dictionary identifier.

3 EVALUATION & FUTURE WORK

We evaluated the compression ratio and robustness using real HTTP business data from ByteDance's edge computing networks. The data consists of over 1000k HTTP messages, with a total size exceeding 4.2 GB, and during the collection period, there were two instances of proactive business transitions. We compared Co4U with existing compression methods. As illustrated in Figure 2, Co4U can achieve a compression ratio improvement of at least 15% and has consistently outperformed other methods. Moreover, during business transitions, such as the first business switch, Co4U rapidly increases the compression ratio, while during the second business switch, Co4U quickly prevents a decrease in the compression ratio. This demonstrates the effectiveness and robustness of Co4U.

In the future, we plan to: 1) explore better dictionaries or other compression methods to achieve higher compression ratios; 2) investigate the optimal timing for dictionary updates under different circumstances; and 3) design and implement all details, and evaluate Co4U with more metrics.

REFERENCES

[1] ByteDance. 2023. https://www.volcengine.com/product/veen. (2023).
[2] Facebook. 2016. Zstandard Compression. https://facebook.github.io/zstd/. (2016).
[3] Google. 2008. SDCH: Shared Dictionary Compression over HTTP. https://chromium.googlesource.com/chromium/src/+/53.0.2744.1/net/sdch/README.md. (2008).
[4] Google. 2015. Brotli Compression. https://github.com/google/brotli. (2015).
[5] Google. 2023. Shared Brotli. https://chromestatus.com/feature/5124977788977152. (2023).
[6] GNU Project. 1992. Gzip Compression. https://www.gnu.org/software/gzip/. (1992).