Available onlinewww.ejaet.com

European Journal of Advances in Engineering and Technology, 2019, 6(10):60-66

Research Article ISSN: 2394-658X

Enhancing Firmware Upgrade Efficiency: The Impact of Compression on ECU Firmware Binary Files

Roopak Ingole

Manager – Advanced Embedded Software
Corporate Research & Technology,
Cummins Inc.
Columbus IN, USA
roopak.ingole@cummins.com

ABSTRACT

Firmware upgrades are critical for ensuring the performance, security, and feature enhancements of electronic control units (ECUs) in modern automotive systems. However, the increasing complexity of these systems poses significant challenges in terms of upgrade timing and data management. This paper explores the role of compression techniques in optimizing firmware upgrade processes for ECUs. By compressing firmware binary files, we hypothesize that the overall time required for firmware upgrades can be significantly reduced, thereby improving the efficiency of automotive software management. Through a series of experiments and simulations, we assess various compression algorithms and their impact on upgrade timing, integrity, and performance of ECUs.

Keywords: ECU Firmware, Electronic Control Units

INTRODUCTION

Electronic Control Units (ECUs) are embedded systems in automotive electronics that control one or more of the electrical systems or subsystems in vehicles. As automotive technology advances, ECUs have become more complex, requiring frequent firmware updates to improve functionality, enhance security, and fix bugs. Traditional firmware upgrade processes often involve significant downtime due to the large size of firmware files and the limited bandwidth of automotive networks. Compression of firmware binary files presents a promising solution to accelerate the upgrade process without compromising the reliability or integrity of the system.

OBJECTIVES

- To analyze the effectiveness of different compression algorithms in reducing the size of ECU firmware binary files.
- To evaluate the impact of compressed firmware files on the time required for firmware upgrades in automotive ECUs.
- To assess the implications of compression on the integrity and performance of the firmware after decompression and installation.

LITERATURE REVIEW

A. Firmware Upgrade Challenges in Automotive ECUs

Several studies have outlined the challenges associated with firmware upgrades in automotive systems, such as bandwidth limitations, upgrade integrity, and system downtime. The need for efficient methods to reduce firmware file size and upgrade time is critical, especially as vehicles become more reliant on software. For this we did some literature study on compression technology as described below.

[1]. DIANA-HEP Project on ROOT I/O Compression:
This article outlines the efforts of the DIANA-HEP project to optimize data handling in the ROOT framework, which is widely used for data analysis in high-energy physics. The project's focus is on improving the I/O (input/output) efficiency through the use of advanced compression techniques. By integrating better compression methods, the project seeks to tackle the challenges associated with handling extremely large datasets, such as those generated by particle accelerators like the Large Hadron Collider. Enhancements in compression not only reduce storage demands but also speed up data processing and retrieval, crucial for timely analysis in physics experiments. In their review, they compared the performance of Zlib, LZMA & LZ4 compression techniques against their developed ROOT IO compression. [1]

[2]. Comparison of Compression Algorithms:
This comprehensive review provides a comparative analysis of multiple compression algorithms by evaluating their performance across different metrics such as speed, compression ratio, and memory usage. The article meticulously details each algorithm's strengths and weaknesses, providing a nuanced view of where each algorithm performs best. For instance, algorithms like LZ4 are noted for their exceptional speed, making them ideal for applications where time is a critical factor, whereas algorithms like XZ offer higher compression ratios at the cost of slower performance and higher CPU usage. This guide is aimed at helping users in selecting the right algorithm based on their specific requirements, balancing between speed and compression efficiency. [2]

[3]. Efficient Text Compression with Brotli:
The Brotli compression algorithm is examined in this vignette, which was created by Google to improve compression techniques specifically for web content. The Brotli algorithm is particularly adept at compressing small files and is optimized for internet speeds, making it highly suitable for mobile web applications. The vignette emphasizes Brotli's effectiveness in reducing data transfer sizes, which can enhance web page load times and reduce bandwidth consumption. [3]

[4]. Linux Compressors Comparison on CentOS 6.5:
This article offers a detailed performance comparison of several popular Linux compression tools on a CentOS 6.5 system. Through systematic benchmarking, it assesses the practical impact of these tools in terms of compression and decompression speed, CPU load, and memory usage. The article provides practical insights into how each compressor performs under different conditions and with different types of data. For example, while BZIP2 offers high compression ratios, it does so at the expense of speed and higher CPU usage, making it less ideal for time-sensitive applications. Conversely, tools like LZ4 and LZO are highlighted for their speed, suitable for environments where performance is a priority. [4]

B. Compression Techniques

Through the literature review, various compression algorithms have been explored in the context of embedded systems, including lossless algorithms like LZ77, LZMA, LZW, Huffman coding, and more recent techniques like Brotli and Zstandard [5]. Each algorithm offers different benefits in terms of compression rate, speed, and resource usage. For our ECU firmware upgrade use-case we focused only on lossless algorithms.

[1]. LZ4 Compression Algorithm: The LZ4 repository on GitHub serves as a resource hub for one of the fastest compression algorithms available today. The documentation provided explores the algorithm's design, which is tailored for very high-speed compression and decompression scenarios. LZ4 is particularly noted for its minimal CPU usage and its ability to provide a good balance between speed and compression ratio, making it suitable for real-time applications such as gaming and video streaming where processing speed is crucial. The repository not only offers the source code but also extensive benchmarks that demonstrate LZ4's performance across various data types and systems. LZ4 supports compression techniques that is suitable for streaming kind of applications. [6] [7]

[2]. LZMA Compression Algorithm: LZMA compression is a type of data-compression algorithm. It was designed by Igor Pavlov as part of the 7z project and was first implemented in 1998. The name "LZMA" stands for "Lempel-Ziv Markov chain Algorithm". LZMA compresses files using both statistical modeling and dictionary techniques. Statistical modeling allows for the analysis of entire blocks of text, whereas dictionary techniques compress small pieces simultaneously. Like most algorithms of this sort, it works by replacing strings that frequently occur in the uncompressed data with pointers to previous occurrences. In LZMA compression, frequency estimates can be calculated given a set of symbols or substrings. Once you have done this, you use the resulting estimates to find matches and replace them with pointers. LZMA is designed to be extremely fast on a wide range of hardware, from traditional hard disks to modern solid-state drives and embedded devices with limited CPU power. [8] [9]

[3]. LZW The LZW algorithm is commonly used to compress GIF and TIFF image files and occasionally for PDF and TXT files. It is part of the Unix operating system's file compression utility. The method is simple to implement, versatile and capable of high throughput in hardware implementations. Consequently, LZW is

often used for general-purpose data compression in many PC utilities. The LZW compression algorithm reads a sequence of symbols, groups those symbols into strings and then converts each string into codes. It takes each input sequence of bits of a given length -- say, 12 bits -- and creates an entry in a table for that particular bit pattern, consisting of the pattern itself and a shorter code. The table is also called a dictionary or codebook. It stores character sequences chosen dynamically from the input text and maintains correspondence between the longest encountered words and a list of code values. [10] [11] [12] [13]

Based on the selected attribute below, we used Pugh matrix to select the algorithm for our experiment. We find LZ4 is best suited algorithm for firmware upgrade over serial link interface (Figure 1).

Compression Algorithm Pugh Matrix							
Critical Quality	***Weight*** *(1 being least important, 10 being most important)*	**LZ4**	**LZMA**	**LZW**	**Zlib**		
Compression Ratio	6	-1	1	1	1		
Compression Time	2	1	1	-1	-1		
Decompression Time	7	1	-1	-1	0		
Low RAM Required	8	1	-1	-1	-1		
Low Flash Required	4	0	0	0	0		
No Filesystem Required	10	1	-1	0	0		
Streaming Support	9	1	-1	-1	-1		

Summary Table						
Total "1s"	5	2	1	1	0	0
Total "0s"	1	1	2	3	0	0
Total "-1s"	1	4	4	3	0	0
Total	20	-16	-20	-13	0	0

Figure 1: Pugh Decision Matrix

METHODOLOGY

A. Experimental Setup

We aim to utilize compression techniques to improve the time it takes to upgrade the Engine Control Unit (ECU). In our system, the ECU upgrade is conducted via a CAN link, which operates at a baud rate of 500kbps (Figure 2). The upgrade process begins with a service tool that either retrieves the firmware from a server or uses a locally stored version. This tool initiates the download and places the ECU into ROM boot mode following the authentication and verification of hardware and software compatibility. The ROM boot of the ECU is designed to receive the firmware and transfer it to Flash memory. The bootloader processes the firmware in a serial stream and sequentially writes it to the flash memory.

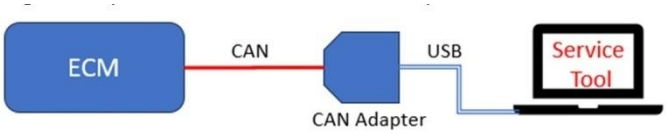

Figure 2: ECU Upgrade Setup

Our ECU utilizes the NXP's MPC5554 [14] microcontroller and employs the Intel HEX file format [15] for the firmware binary. An Intel HEX file is an ASCII text file consisting of lines that conform to the Intel HEX format (Figure 3), where each line contains one HEX record composed of hexadecimal numbers that signify machine language code and/or constant data. The HEX file is downloaded one record at a time, and the bootloader writes the data to the specified address in each record. The download occurs over the CAN link, which has a baud rate of 500kbps. This link becomes the critical constraint in accelerating the firmware flashing process, leading to ECU upgrade times ranging from 15 to 20 minutes—a significant inconvenience for both service technicians and end customers.

```
:10246200464C5549442050524F46494C4500464C33
|||||||||||                                  CC->Checksum
||||||||DD->Data
|||||||TT->Record Type
|||AAAA->Address
|LL->Record Length
:->Colon
```

Figure 3: Intel HEX Record

Due to the bootloader's lack of file system support and the record-by-record download approach, we have chosen to implement LZ4 decompression for its performance, which is highly suited for this application. We have integrated the LZ4 'C' library into the bootloader and utilized its Streaming API [16] to decompress the received compressed HEX record and write it directly to the ECM Flash. The entire procedure is illustrated flowchart (Figure 4) and message sequence chart shown below (Figure 5):

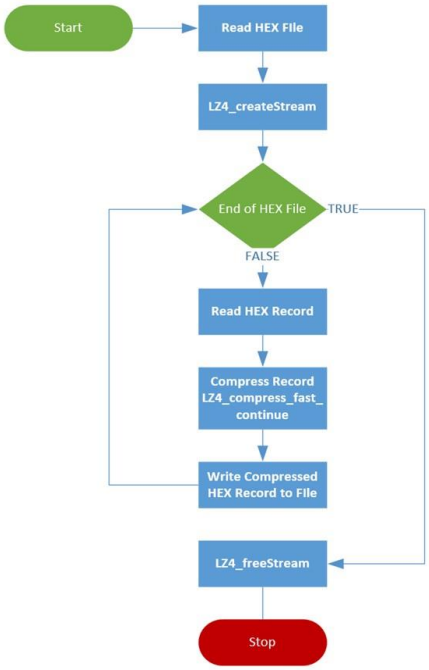

Figure 4: LZ4 HEX File Compression Flowchart

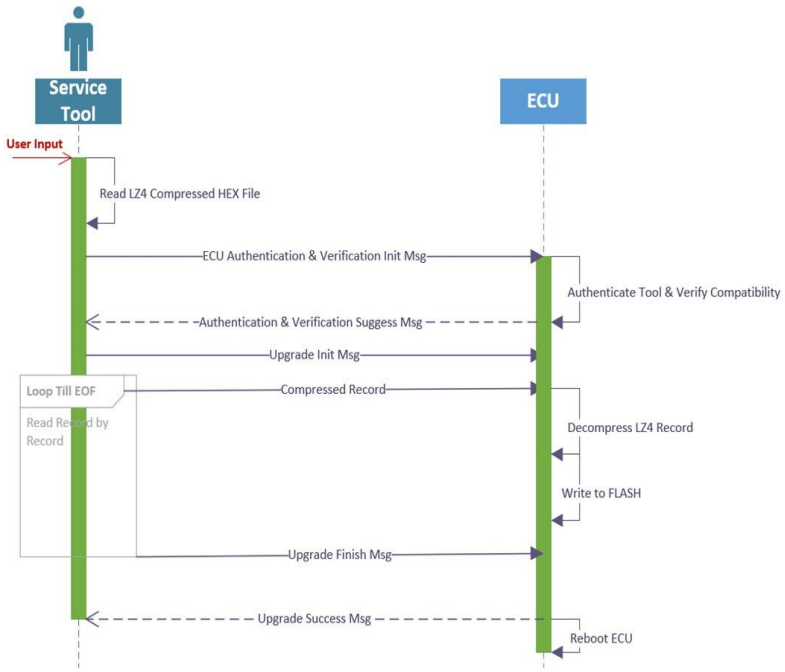

Figure 5: ECU Upgrade Message Sequence Chart

4. RESULTS

A. Compression Efficiency

In evaluating the effectiveness of compression techniques for ECU firmware updates, our analysis specifically looked at the performance of the LZ4 compression algorithm. Although LZ4 does not achieve the highest compression ratio when compared to other available compression algorithms, it delivers substantial compression efficiencies that are quite significant for practical applications. Specifically, LZ4 achieves an average

compression ratio of about 50% across various types of ECU firmware files. This ratio indicates that LZ4 typically reduces the file size by half, which is a considerable reduction in data that needs to be transmitted over the CAN link during firmware upgrades.

The benchmarking results of LZ4 compression are particularly noteworthy:

Compression with LZ4_compress_fast: The original file size of 8,364,153 bytes was compressed to 4,302,002 bytes, achieving a compression ratio of 51.43%. This operation was performed at a speed of 359.7 MB/s.

Decompression with LZ4_decompress_fast: The decompression speed was recorded at 2599.6 MB/s, starting from the compressed file size back to its original.

These results highlight LZ4's strength in providing fast compression and decompression speeds. The high speed is crucial for time-sensitive applications such as ECU firmware updates where every second counts. The faster compression and decompression allow the firmware upgrade process to be completed more quickly, which is beneficial not only in reducing downtime during maintenance but also in enhancing the overall user experience by minimizing the vehicle's out-ofservice time.

Given these characteristics, LZ4 is particularly suited for scenarios where speed is more critical than achieving the absolute best compression ratio. The trade-off between speed and compression efficiency is justified in the context of ECU firmware updates, where the priority is to minimize upgrade time without excessively compromising on data transfer sizes. This makes LZ4 an ideal choice for automotive systems where both time and bandwidth are of essence.

B. Upgrade Timing

In the context of optimizing the firmware upgrade process for Engine Control Units (ECUs) via CAN links, our exploration into the use of LZ4 Frame compression has yielded significant improvements in reducing the total upgrade time. The use of LZ4, a compression algorithm known for its exceptional speed and decent compression efficiency, has demonstrated a profound impact on the efficiency of the firmware update process.

When LZ4 Frame-compressed files are employed, the data that needs to be transmitted across the CAN network is substantially reduced. This reduction is crucial given the bandwidth limitations of the CAN link, which typically operates at a baud rate of 500kbps. By compressing the firmware files, the amount of data sent over the network is decreased, thereby reducing the time required for data transmission.

Our findings indicate that the use of LZ4 compressed files leads to a reduction in firmware upgrade times by approximately 50% on average. This is a significant enhancement compared to using uncompressed files. For instance, if an uncompressed firmware upgrade process typically takes about 20 minutes, using LZ4 compressed files could reduce this to around 10 minutes. This halving of the upgrade time represents a major improvement in operational efficiency and has practical implications for automotive service operations.

This substantial decrease in upgrade time can be attributed to several factors:

Reduced Data Transmission Time: The primary benefit of using LZ4 compression is the reduction in the size of the firmware files, which directly decreases the time spent transmitting data over the CAN link.

Efficient Decompression: LZ4's high decompression speed ensures that the time to decompress the received data on the ECU side remains minimal, preventing potential bottlenecks in the upgrade process.

Streaming Process: With smaller data packets, the process of writing to the ECU's flash memory can be executed more swiftly, as the system handles less data overall during the upgrade.

These improvements not only reduce the downtime associated with firmware upgrades but also enhance the overall throughput of service operations, allowing service centers to handle more vehicles within the same timeframe. Additionally, the reduction in upgrade time significantly improves the service experience for both technicians and customers, minimizing the vehicle's downtime and potentially increasing customer satisfaction with the service process.

In conclusion, the implementation of LZ4 Frame compression in the ECU firmware upgrade process marks a substantial advancement over the status quo, offering a more efficient, less time-consuming alternative that leverages fast data compression and decompression capabilities to meet the demands of modern automotive systems.

DISCUSSION

The results of our investigation into the use of the LZ4 compression algorithm for ECU firmware upgrades demonstrate that LZ4 effectively reduces file sizes, which is critical for optimizing the upgrade process over CAN links. However, the choice of a compression algorithm must consider various factors beyond mere reduction in file size. These factors include compression and decompression speeds, as well as the computational resources available on the ECU hardware, which can vary significantly across different vehicle models and generations.

LZ4 is particularly noted for its balance between high decompression speeds and a satisfactory reduction in file size, making it suitable for time-sensitive applications such as ECU firmware upgrades. Nonetheless, the specific requirements of a given application may necessitate evaluating other compression algorithms that might

offer either faster compression speeds or higher compression ratios, depending on what aspect of performance is more critical under certain operational constraints.

An important aspect of our findings is the confirmation that using LZ4 compression does not compromise the integrity or functionality of the ECU after the upgrade. Post-upgrade assessments show that ECUs remain fully operational, with all functionalities performing as expected without any degradation. This ensures that the compression process, while reducing file size and transmission time, does not alter the essential data or introduce errors during the decompression and writing stages. Maintaining the operational integrity of ECUs post-upgrade is paramount, as any compromise could affect vehicle safety and performance.

Looking ahead, there is potential to achieve even greater compression efficiencies by exploring the High Compression capability of LZ4. This variant of LZ4 is designed to compress data at higher ratios, which could further reduce transmission times and improve overall upgrade efficiency. However, this approach typically requires more processing power for both compression and decompression, which might not be feasible on all ECU models due to hardware limitations.

Further evaluation is necessary to determine the feasibility and benefits of implementing LZ4's High Compression capabilities in the ECU upgrade process. This evaluation would need to carefully balance the improved compression ratios against the increased resource demands, ensuring that any gains in file size reduction do not adversely impact the upgrade process speed or the functional performance of the ECUs. Such investigations will help refine our understanding of the optimal use of compression technologies in automotive firmware updates, aiming to enhance efficiency without sacrificing reliability or safety.

CONCLUSION

The research conducted on the application of compression techniques to ECU firmware upgrades has underscored the critical role these technologies play in enhancing the efficiency of automotive software management. By deploying LZ4 compression, we have demonstrated a substantial reduction in the firmware upgrade time by approximately 50%, marking a significant improvement over traditional methods that involve larger, uncompressed files.

This achievement not only streamlines the upgrade process but also minimizes the downtime for vehicles undergoing software enhancements, thereby enhancing service throughput and reducing inconvenience to both technicians and end-users. The success of LZ4 in this context is primarily attributed to its exceptional balance of high-speed compression and decompression capabilities, which are crucial in scenarios where upgrade time is of paramount importance.

Additionally, our findings confirm that the integrity and functionality of ECUs remain unaffected post-upgrade, reinforcing the reliability of compression as a method to transmit and install firmware updates without compromising system performance. The potential for even greater efficiencies through the exploration of LZ4's High Compression options suggests a promising avenue for future research, albeit with considerations for the varying computational power of different ECU models.

In conclusion, the integration of compression technologies like LZ4 into the ECU firmware upgrade process represents a significant technological advancement, offering a more efficient, faster, and reliable alternative to the status quo. This adaptation not only meets the demands of modern automotive systems but also sets a benchmark for future innovations in ECU firmware management. Further investigations will continue to refine these approaches, aiming to optimize both the performance and reliability of vehicle electronic systems in an increasingly software-dependent automotive landscape.

REFERENCES

[1]. Diana-hep, "Exploring Compression Techniques of ROOT IO," DianaHep, [Online]. Available: https://dianahep.org/pages/project_root_io_compression.html.html.

[2]. Linux Reviews, "Comparison of Compression Algorithms," Linux Reviews, [Online]. Available: https://linuxreviews.org/Comparison_of_Compression_Algorithms.

[3]. E. K. Z. S. a. L. V. Jyrki Alakuijala, "Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms," Google Inc., 22 Sept. 2015. [Online]. Available: https://cran.rproject.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf.

[4]. G. Danti, "Linux compressors comparison on CentOS 6.5 x86-64: lzo vs lz4 vs gzip vs bzip2 vs lzma," Linuxaria, 29 May 2014. [Online]. Available: https://linuxaria.com/article/linux-compressors-comparisonon-centos-6-5-x86-64-lzo-vs-lz4-vs-gzip-vs-bzip2-vs-lzma.

[5]. M. Zhang and H. Lee, "A Comparative Study of Lossless Compression Algorithms for Embedded Systems.," Journal of Digital Technology, 2019.

[6]. lz4.org, "LZ4," [Online]. Available: https://github.com/lz4/lz4.

[7]. Wikipedia, "LZ4 (compression algorithm)," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/LZ4_(compression_algorithm).

[8]. Lloyd, "easylzma," [Online]. Available: https://lloyd.github.io/easylzma/.

[9]. Winzip, "LZMA: What is LZMA Compression?," [Online]. Available: https://www.winzip.com/en/learn/tips/what-is-lzma/.
[10]. R. Awati, "LZW compression," [Online]. Available: https://www.techtarget.com/whatis/definition/LZW-compression.
[11]. J. Davies, "How LZW (GIF) Compression Works," [Online]. Available: https://commandlinefanatic.com/cgi-bin/showarticle.cgi?article=art010.
[12]. M. Dipperstein, "Lzw," [Online]. Available: https://github.com/MichaelDipperstein/lzw.
[13]. Wikipedia, "Lempel–Ziv–Welch," [Online]. Available: https://en.wikipedia.org/wiki/Lempel%E2%80%93Ziv%E2%80%93We lch#:~:text=Lempel%E2%80%93Ziv%E2%80%93Welch%20(LZW,in %20the%20GIF%20image%20format..
[14]. NXP, "32-bit MCU for Powertrain Applications MPC5554," [Online]. Available: https://www.nxp.com/products/processors-andmicrocontrollers/legacy-mpu-mcus/mpc55xx-mcus/32-bit-mcu-forpowertrain-applications:MPC5554.
[15]. ARM, "GENERAL: Intel HEX File Format," ARM Developer, [Online]. Available: https://developer.arm.com/documentation/ka003292/latest/#:~:text=The %20Intel%20HEX%20file%20is,code%20and%2For%20constant%20d ata..
[16]. lz4.org, "LZ4 Block Format Description," [Online]. Available: https://github.com/lz4/lz4/blob/dev/doc/lz4_Block_format.md.
[17]. Embedded, "Lossless Data Compression for Embedded Systems," Embedded, 9 June 2009. [Online]. Available: https://www.embedded.com/lossless-data-compression-for-embeddedsystems/