Can Beacons be Compressed to Reduce the Channel Load in Vehicular Networks?

Miguel Sepulcre, Pedro Tercero, Javier Gozalvez
UWICORE Laboratory, http://www.uwicore.umh.es/
Universidad Miguel Hernández de Elche (UMH), Spain
msepulcre@umh.es, pedro.tercero@alu.umh.es, j.gozalvez@umh.es

***Abstract*—Significant efforts have been devoted to date to the congestion control problem in vehicular networks. The solutions proposed so far have been designed to adapt the communication parameters to reduce and control the channel load. A totally different approach would be the compression of the data generated by each vehicle. This paper proposes and explores for the first time the use of data compression to reduce the channel load in vehicular networks. By compressing and decompressing V2X messages, the channel load generated could be reduced, thereby decreasing the interference and packet loses due to collisions. We apply this idea in this study to CAMs using existing data compression tools to have a first estimate of the compression gain that could be achieved, and the time needed to compress and decompress. The results obtained show that the CAM length could be reduced by up to around 14%, which is a non-negligible percentage given the relevance of the congestion control problem. The data compression and decompression times obtained demonstrate its potential for its integration in V2X devices. The results obtained motivate to more deeply investigate the compression of V2X messages in vehicular networks.**

Keywords—vehicular networks, data compression, beaconing, congestion control, DCC, V2X communications, channel load.

I. INTRODUCTION

Bandwidth is a scarce resource in any communication system or network. Vehicular networks are not an exception and significant efforts have been dedicated to date to design congestion control protocols that reduce and control the channel load. One of the most relevant ones is the DCC (Decentralized Congestion Control) solution defined by ETSI in Europe that spans over multiple layers of the protocol stack [1]. The solutions proposed to date normally adapt the communication parameters, e.g. transmission power, the message transmission frequency or the data rate to control the load. This type of solutions can influence the application's effectiveness, since they affect e.g. the communication range.

Data compression is an alternative approach to reduce the channel load of vehicular networks that would not require the modification of the communication parameters. Data compression is widely used in communication systems to improve the bandwidth utilization. For example, in HTTP, data is compressed before it is sent from servers. Browsers are in charge of downloading and decompressing the received data. The most common compression schemes used for HTTP compression include Gzip and Compress [2]. According to [3], HTTP compression can provide a compression gain of around 75% for text files (HTML, CSS, and JavaScript) with file sizes between 26.000 Bytes and 74.000 Bytes, approximately.

This paper proposes and explores for the first time the use of data compression to reduce the channel load in vehicular networks. We propose that the payload of V2X messages is compressed after being generated by the upper layers to reduce the number of bits of each message without reducing the amount of information to be sent. This could be implemented at the Facilities layer of the ETSI or ISO ITS Architectures, or above the WSMP Transport Layer of the 1609/WAVE Architecture. At the receiver, compressed V2X messages will need to be decompressed to recover the original messages. A new module could be incorporated to the protocol stack in all vehicles for the data compression and decompression. The proposed data compression would not require any other significant modification of the protocol stack, which increases its potential for standardization and real-world implementation.

The compression gain depends significantly on the size of the input data and the type of data itself. For example, large text files normally have repeated substrings (e.g. words) and can be significantly compressed. This is the case because compression algorithms are often designed to replace repeated substrings with a pointer to the previous occurrence of the repeated substrings. However, V2X messages are characterized by having a relatively small size (hundreds of Bytes). In addition, the content of V2X messages can significantly vary and the existence of repeated substrings has not been studied yet. Another issue is that V2X communication devices integrated in vehicles have limited processing power. Therefore, compression and decompression processes would have to meet the strict latency requirements of vehicular applications. All these factors could limit the feasibility of using data compression in vehicular networks and therefore require detailed investigations.

In this paper, we provide first results of data compression and decompression in vehicular networks by evaluating the compression gain and the time needed to compress and decompress of different algorithms using existing open software tools. In this study we apply the data compression to CAMs (Cooperative Awareness Messages), also known as beacons or BSMs (Basic Safety Messages), since they consume a large portion of the vehicular control channel.

The authors acknowledge in part the support of the *Conselleria de Educación, Investigación, Cultura y Deporte* of *Generalitat Valenciana* through the project AICO/2018/A/095, the support of the Spanish Ministry of Economy and Competitiveness and FEDER funds under the project TEC2017-88612-R, and the support of the Horizon2020 Programme through the TransAID project (Grant Agreement no. 723390).

II. Data compression

The two data compression tools analyzed in this study are Compress [4] and Gzip [5]. Both tools are open source solutions and widely used. They are based on the well-known Lempel-Zip algorithm. This algorithm is a universal lossless data compression algorithm that achieves compression by looking for repeated substrings in the data.

Gzip makes use of the original Lempel-Ziv algorithm, also known as LZ77 [6]. The LZ77 algorithm looks for repeated substrings based on the concept of sliding window. As the data is compressed, LZ77 only looks for repeated substrings in a window of previously compressed data. The window can be divided into a search buffer containing the data that has already been compressed, and a lookahead buffer containing the data yet to be compressed. As the data is compressed, the window slides along, removing the oldest compressed data from the search buffer and adding new uncompressed data to the lookahead buffer. Once a substring in the lookahead buffer is found to be completely contained in the search buffer, it is replaced by its position in the search buffer and its length. The output format produced by Gzip is described in RFC 1952 [7] and includes a 10-byte header, some optional extra headers, the compressed data and an 8-byte footer containing a CRC-32 checksum and the length of the original uncompressed data.

Compress uses the Lempel-Ziv-Welch algorithm or LZW [8]. The LZW algorithm makes use of a dictionary that is built based on the input data, where each entry in the dictionary has an index. If the algorithm is configured to operate using bytes, it is initialized with one entry for each of the 256 possible values. When a substring, *S*, of the data being analyzed is found in the dictionary, it is replaced by its index and a new entry is added to the dictionary that contains *S* and the next symbol in the data. This means that new entries are only added if a prefix one byte shorter is already in the dictionary (e.g. "sun" is only added if "su" had previously appeared in the data).

The compression gain depends on the size of the input data and the distribution of common substrings. Typically, LZ77 (Gzip) is able to achieve a compression gain of 60-70% for text such as source code or English, while LZW (Compress) is able to achieve 50-60% also for text [4][5]. The compression gain that these algorithms and software tools could provide when compressing CAMs or other V2X messages needs yet to be studied, because they significantly depend on their length and their content (e.g. existence of repeated sequences).

III. CAM structure and format

A CAM is composed of one common header and multiple containers [9]. The common header includes the protocol version, the message type and the ID of the vehicle or RSU (Road Side Unit) that transmits the CAM. For vehicles, a CAM must contain one Basic container and one High frequency container (both are mandatory), and may also include one Low frequency container and one or more other Special containers. The Basic container includes information related to the transmitting vehicle, such as the type of vehicle or its geographic position. The High frequency container contains highly dynamic information of the transmitting vehicle, such as its heading or speed. The Low frequency container contains static and not highly dynamic information of the transmitting vehicle, such as the status of the exterior lights. The Special vehicle container includes information specific to the vehicle role. Each container is composed of a sequence of optional or mandatory data elements (DE) and/or data frames (DF).

The CAM makes use of the ETSI common data dictionary [10] and is formally defined in ASN.1 language in [9]. ASN.1 is a formal notation used for describing data transmitted by telecommunications protocols, regardless of language implementation and physical representation of these data. ETSI also specifies that CAMs have to be encoded and decoded following unaligned packed encoding rules (PER) [11]. These rules describe how the values defined in ASN.1 should be encoded for transmission (i.e. how they should be translated into bits). At the transmitting vehicle, the output of the PER encoding process would be the input of our data compression. The corresponding security certificates would be added to the compressed CAM following the ETSI specifications.

The length of the CAM depends on the number of optional containers considered. Since many containers, DEs and DFs are optional in the CAM, we have classified the CAMs generated in three different groups with different lengths:

- Group BH (Basic and High frequency): CAMs belonging to this group only contain the mandatory DEs and DFs, i.e. the Basic container and the High frequency container.
- Group BHL (Basic, High and Low frequency): CAMs that belong to this group contain the same DEs and DFs than group BH, plus the Low frequency container.
- Group BHLS (Basic, High and Low frequency and Special): CAMs belonging to this group contain the same DEs and DFs than group BHL, plus the Special vehicle container and additional optional DEs and DFs not included in previous groups.

To analyze the performance of data compression on CAMs, we have generated 10^4 CAMs for each of the three groups identified that are compliant with the ETSI specifications. To this aim, we have mapped the ASN.1 definition of the CAM to a Java data structure using ASN.1 Studio software tool. Using Java, we have created our own code to generate CAMs. To avoid limiting this study to a small set of CAMs generated with similar values in the DEs and DFs, we have randomly generated all the values in the DEs and DFs, i.e. random speeds, geographic position, heading, etc. We have verified that the CAMs we generate follow the ETSI specifications by decoding them using online decoding tools provided by OSS (http://asn1-playground.oss.com/) and MARBEN (https://www.marben-products.com/decoder-asn1-automotive).

Fig. 1 shows the PDF (Probability Density Function) of the length of the CAMs generated for the three groups defined. The average length of the CAMs belonging to each group is 136 bytes, 195 bytes and 268 bytes for groups BH, BHL and BHLS, respectively. The length of the CAMs depend on the number of optional containers included, and is therefore different for the three groups of CAMs defined. In addition, the specific values of the DEs and DFs also have an influence on

the CAM length because of the unaligned PER rules. As a consequence, a CAM composed of the same containers does not always have the same length depending on the content of the DEs and DFs. As a result, the length of the CAMs is not exactly the same for all CAMs belonging to the same group.

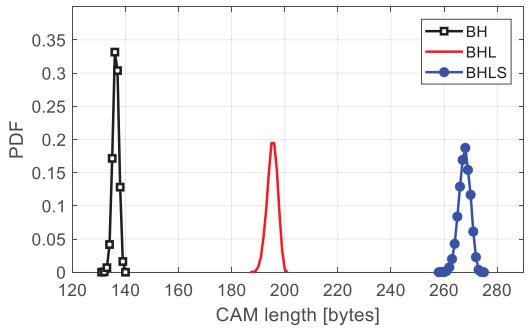


Fig. 1. Length of the CAMs generated in this study.

IV. Results

The results presented in this paper have been obtained using Gzip 1.6-4 and Compress 4.2.4.4-15 with their default parameters. The implemented software was executed in a virtual machine running Ubuntu 16.04.5 over Windows 10 with 64 bits. All the tests were run in a desktop computer with an Intel Core i5-4440 processor of 3.1GHz and 8GB of RAM DDR3 (1GB of RAM was reserved for the virtual machine).

A. Compression gain

Using point clouds, Fig. 2 depicts the length of the compressed CAMs as a function of their original length for the three groups of CAMs considered in this study (BH, BHL and BHLS) and the two data compression tools used (Compress and Gzip). Each point in the figure represents one or more CAMs because they often overlap. The dashed line in the figure separates the CAMs that could be compressed (i.e. the compressed length was lower than the original one) and the CAMs that could not be compressed (i.e. the output of the compression process had more bytes than the original CAM). Therefore, all points that are below the dashed line represent CAMs that could be successfully compressed. The results presented in this figure demonstrate the potential of Compress and Gzip to compress CAMs. For the shortest CAMs (group BH, Fig. 2a), Gzip was not able to reduce the message size and the output was larger than the original one. This effect is normally referred to as negative compression, and is produced due to a low redundancy in the data (low number of repeated substrings) and the overhead produced by headers and footers. It could be avoided by transmitting the original messages instead of the compressed ones when a negative compression is detected. Fig. 2 also shows that the length of the compressed CAM does not only depend on the length of the original CAM, but also on its content (i.e. the same original CAM length does not always result in the same compressed CAM length).

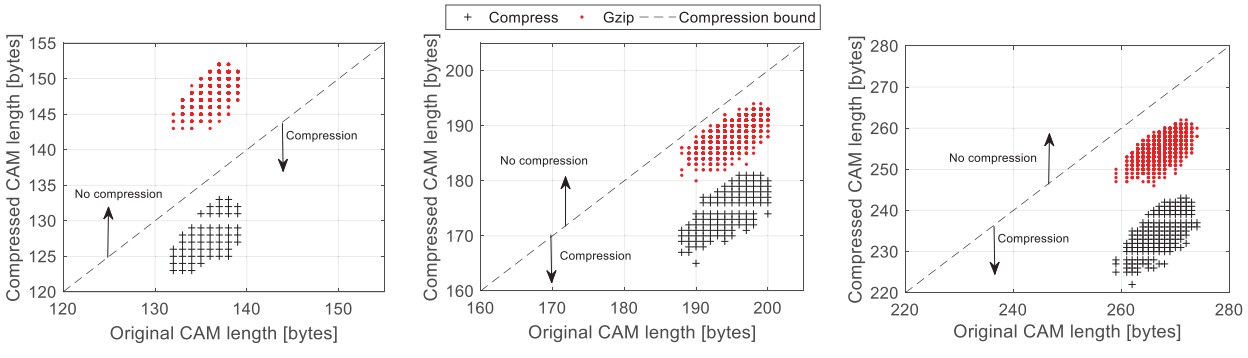
The compression gain (CG) has been calculated in this study for each CAM with the following equation:

$$CG = 100 \cdot \frac{L_o - L_c}{L_o} \quad (1)$$

where L_o is the length of the original CAM and L_c is the length of the compressed CAM. This means that e.g. if CG=10%, the length of the original CAM has been reduced by 10% thanks to the data compression; similarly, a CAM of 100 bytes would be reduced to 90 bytes if CG=10%. Fig. 3 plots the PDF of the compression gains obtained. The results obtained show that Compress outperforms Gzip for all groups of CAMs analyzed in this study. As it can be observed, Compress produces compression gains of around 5% for BH, around 10% for BHL and around 12% for BHLS. The maximum compression gain provided was around 14%. Gzip presents lower compression gains, and it actually increases the length of the CAM around 10% for BH. Both Compress and Gzip have in general higher compression gain as the length of the CAM increases (e.g. higher gain for BHLS than for BHL or BH) due to the higher possibility of finding repeated substrings.

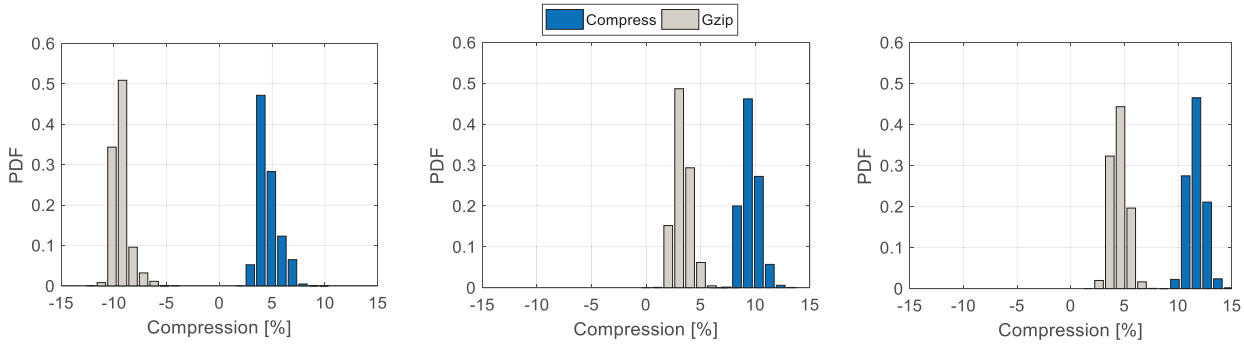
B. Compression and decompression times

An important factor to determine the feasibility of data compression for vehicular networks is the time needed to compress and decompress each V2X message. This is the case given the strict latency requirements of the V2X message generation and reception. Fig. 4 depicts the average time needed to compress and decompress a CAM obtained in this study. These results have been obtained by directly measuring the time needed by the desktop computer to compress/decompress the CAMs generated. As it can be observed in Fig. 4a, the average time needed for CAM



(a) Group BH (b) Group BHL (c) Group BHLS

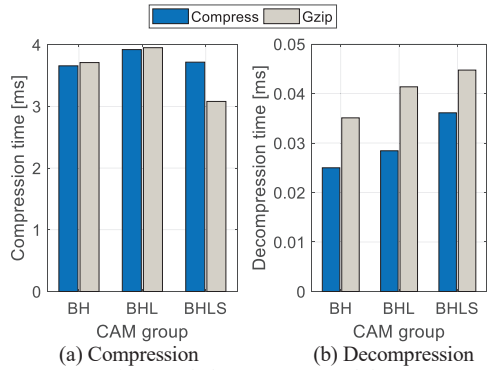
Fig. 2. Point clouds of compressed vs. original CAM lengths.



(a) Group BH (b) Group BHL (c) Group BHLS

Fig. 3. Probability Density Function of the CAM compression gain.

compression is between 3ms and 4ms (both Compress and Gzip provide similar compression times). Since CAMs are typically generated every 100ms, these results seem more than reasonable for the practical implementation of data compression on the transmitting vehicle.



(a) Compression (b) Decompression

Fig. 4. Average time needed to compress and decompress a CAM.

On the receiving vehicle, the decompression process is simpler because it does not require to look for repeated substrings, but just to replace the indexes/pointers by previous occurrences of repeated substrings. The average time needed to decompress a CAM is one order of magnitude lower (between 0.25ms and 0.45ms approximately, see Fig. 4b) than the compression time. This means that roughly between 2200 and 4000 CAMs could be decompressed per second under the considered conditions. The number of beacons that can be successfully transmitted over the vehicular control channel is around 1200 for a channel load of 60% when using IEEE 802.11p in a 10-MHz channel [12] (the load of the control channel will be controlled and maintained around 60% thanks to the congestion control algorithms). Therefore, the decompression times achieved in this study would allow decompressing fast enough all the messages that could be received through the radio interface.

V. Conclusions

This paper proposes and explores for the first time the compression of V2X messages to reduce the load in vehicular networks. To do so, existing data compression tools have been used to compress CAMs and evaluate their compression gain and the time needed to compress/decompress. The results obtained in this study show that CAMs can be compressed between 4% and 14% approximately. However, the compression gain notably depends on the CAM size, its content and the compression algorithm used. The data compression and decompression times obtained could be sufficient to enable its implementation in current V2X devices. This paper motivates further studies on data compression to optimize the compression gain and the time needed to compress/decompress CAMs. To increase the compression gain, the security overhead could be compressed together with the payload to augment its length and the possibility of having repeated substrings. Also, unnecessary headers and footers could be removed to maximize the compression gain. Other types of compression algorithms and V2X messages could also be studied.

References

[1] ETSI TC ITS, "Intelligent Transport Systems (ITS); Cross Layer DCC Management Entity for operation in the ITS G5A and ITS G5B medium", ETSI TS 103 175, v1.1.1, June 2016.

[2] IETF, "Hypertext Transfer Protocol -- HTTP/1.1", The Internet Society (1999). Online available: https://tools.ietf.org/html/rfc2616

[3] Andrew B. King, "Speed Up Your Site: Web Site Optimization", *New Riders Press*, ISBN-10: 0735713243, January 2003.

[4] Ubuntu Documentation: Compress. Online [last access on Sept. 2018]: http://manpages.ubuntu.com/manpages/xenial/man1/compress.1.html

[5] GNU Documentation: Gzip. Online [last access on Sept. 2018]: https://www.gnu.org/software/gzip/

[6] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression", *IEEE Transactions on Information Theory,* vol. 23, no. 3, pp. 337–343, May 1977.

[7] IETF, "GZIP file format specification version 4.3", The Internet Society (1996). Online available: https://tools.ietf.org/html/rfc1952

[8] Terry A. Welch, "A Technique for High Performance Data Compression", *IEEE Computer*, vol. 17, no. 6, pp. 8-19, June 1984.

[9] ETSI TC ITS, "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service", ETSI EN 302 637-2 V1.3.2, Nov. 2014.

[10] ETSI TC ITS, "Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary", ETSI TS 102 894-2, v1.2.1, Sept. 2014.

[11] ITU-T, "Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)", *Recommendation ITU-T X.691*, August 2015.

[12] G. Bansal, et al., "LIMERIC: A Linear Adaptive Message Rate Algorithm for DSRC Congestion Control", *IEEE Transactions on Vehicular Technology*, vol. 62, no. 9, pp. 4182-4197, Nov. 2013.