Smart data preprocessing method for remote vehicle diagnostics to increase data compression efficiency

Lorenz Görne1 · Hans-Christian Reuss2 · Andreas Krätschmer1 · Ralf Sauerwald1

Received: 29 October 2021 / Accepted: 28 June 2022 / Published online: 17 August 2022
© The Author(s) 2022

Abstract

The increasing number of functions in modern vehicle leads to an exponential increase in software complexity. The validity and reliability of all components must be ensured, making the use of appropriate vehicle diagnostics systems indispensable. The purpose of such systems is to collect and process data about the vehicle. To find issues during vehicle development, the OEMs will usually have a development fleet of thousands of vehicles. The challenge for diagnostic systems is to detect issues during these tests, as well as collecting as much data as possible about the circumstances that led to the fault. A single-vehicle produces hundreds of gigabytes of data per month. The required data bandwidth cannot be fulfilled by current mobile network subscriptions as well as WIFI or cable-based infrastructure. This limits the amount of data that can be collected during field tests and hinders big data analysis like AI training or validation. Hence a software solution for data reduction is necessary. The authors present a method for data handling that drastically reduces the amount of data consumption and optimizes the transfer delay between a remote-diagnostic systems and the cloud. Using a pipeline of data preprocessing as well as an established compression algorithm, the amount of transmitted data is reduced by a factor of nearly ten. This method will allow to collect more data in field testing and improve the understanding of issues during vehicle development.

Keywords Vehicle signal compression · Vehicle diagnostics · Data-driven design

1 Introduction

Today’s vehicles are becoming increasingly complex. The customer of current vehicles expects a broad field of features and customization to choose from. This leads to an acceleration of the development cycle of new vehicles, which brings new challenges for original equipment manufacturers (OEMs). One of the new challenges is to achieve reliability and safety for each possible product variation.

A modern vehicle has drastically more software code as it incorporates many more computation units than its predecessor [1]. Extensive field testing can be used to ensure reliability. If any faulty behavior is detected, a detailed error report is generated and sent to the responsible developer, who then can find a solution. However, there may be a significant time delay between the error occurrence and the effect getting noticeable, which makes finding the root cause difficult. Intensive data collection can help the developer to find the error more quickly. If data is lost, it is possible to miss failure events which would then have to be reproduced in a costly manner.

In addition, it is desirable to always collect data during initial testing. Even if no error is currently detected, the data can later be used for reference if defects in the product start to become noticeable. Furthermore, the data is required for training of artificial intelligence (AI) based applications, as they need several hundred thousand of virtual driving kilometers to achieve reliability [2]. Therefore, test driving data without error occurrences serve as valuable reference material for AI training.

1 FKFS, Research Institute for Automotive and Engine Design, Pfaffenwaldring 12, 70569 Stuttgart, Germany
2 IFS Stuttgart, Institute for Automotive Technology, Pfaffenwaldring 12, 70569 Stuttgart, Germany

* Hans-Christian Reuss
Hans-Christian.Reuss@ifs.uni-stuttgart.de
Lorenz Görne
Lorenz.Goerne@fkfs.de
Andreas Krätschmer
Andreas.Kraetschmer@fkfs.de
Ralf Sauerwald
Ralf.Sauerwald@fkfs.de
It is crucial not to lose valuable vehicle testing data during transmission from the vehicle to the data backends. However, transmitting data in field conditions is non-trivial. Current technology includes data loggers that may use WIFI or mobile internet to upload the data that has been collected during driving. Both technologies depend on infrastructure, which shows varying quality depending on the location. For Germany, the median connection speed in the upload direction is currently 8 Mbit/s [3], which is the fourth-highest value globally (see Fig. 1). This number includes mobile internet as well as other technologies like cable-based internet. The usable bandwidth for a moving vehicle is expected to be worse due to mobile station handover [4] and variation in the quality of service in remote locations [5, 6]. Common protocols like HTTP or MQTT, which are necessary for actual data upload, add overhead and further reduce the usable bandwidth.

The data generated within the vehicle is roughly two to ten Mbit/s and easily exceeds 100 Mbit/s for autonomous vehicles [7]. This means that with further advances in vehicle automation, the generated data within a vehicle will not be transmitted in real-time using the current infrastructure. The use of external storage devices may only mitigate the problem of local data connection issues for a short time. The data must still be transferred via the internet using static infrastructure or mobile data connections at some point. Even uploading the data during still-stands like overnight can be insufficient if the available bandwidth is many times lower than the data generation bandwidth, as it would be expected based on the average upload speed currently available.

The amount of data being collected within vehicles will increase in the future. The available internet bandwidth will most likely not increase at the same rate. Data loss may be acceptable for some application right now. However, due to the increased amount of data in the future, a suitable data upload method will become more and more obligatory. Many efforts have been made to incorporate compression into the data loop for better usage of the upload data bandwidth [8–10]. However, the authors are not aware of any method that focuses on the use of well-established compression algorithms as a backbone for data uploads.

Unlike other methods, specific optimization is done to the data format of vehicle bus communication, prior to using a well-established compression algorithm. A preprocessor of data structure prior to a compression stage has proven effective in other fields of research [11, 12] and increases the efficiency of compression algorithms significantly. Different preprocessing techniques are presented and evaluated, based on their benefit on the overall compression rate of the vehicle communication data when compressed using the popular LZMA2 compression algorithm.

The presented method improves the compression rate for vehicle communication data. State-of-the-art and general-purpose compression algorithms like LZMA2 and DEFLATE are able to achieve a factor of roughly six at best, which can be increased through the presented method to a factor of nearly ten. Further improvements of the compression rate can be expected through the usage of more preprocessing techniques as well as finding better compression algorithms.
2 Background and related work

As the vehicle data are collected during field testing, it is not feasible for the vehicle developer to be present during testing. Therefore, a diagnostic system is incorporated into the vehicle that continuously collects and processes the data that is generated within the vehicle.

There are methods in the literature that reduce the amount of data being uploaded by filtering off specific information [13, 14], or by use of time slices around an event of interest, like a detected error. However, the reduced amount of data does not give the developer a full picture and may miss information that is essential for the diagnosis of the error.

Another drawback of such methods is that errors may be discovered during test driving, which cannot be diagnosed by the developer because the filters have not been setup to detect them yet. As time progresses, these errors are noticed by the developer and filters are updated to detect and diagnose the error, but new test driving must be carried out to gather the necessary information. For example, a faulty motor control unit may wear down the engine sooner than expected and therefore lead to an engine breakdown a few thousand kilometers later. Tracking down the root cause, in this case, may be difficult if there is only data available during or just before the engine breakdown.

Another motivation for continuously monitoring the vehicle state is to train or to validate artificial intelligence (AI). As AI needs training data to learn how to behave during normal operation, it is indispensable to have hundreds of thousands of driving kilometers for the AI to process. In addition, capturing the whole vehicle data during the operation of the vehicle makes it possible to justify decisions that the AI has made in a particular situation.

2.1 In-vehicle data flow

The different electronic control units (ECUs) of a vehicle usually communicate through data buses. These buses have a cost advantage when it comes to wiring the connecting lines through the vehicle. At the same time, they make it possible to intercept the communication using a diagnostic device and therefore monitor or record the ongoing vehicle communication, see Fig. 2.

This bus communication transports basically all information that is necessary to determine the current state of the system—sensor values, state variables or control requests are exchanged multiple times per second between the ECUs. Common bus technology being used in modern vehicles is the controller area bus flexible data rate (CAN-FD) or the Automotive Ethernet. In this paper, only the CAN-FD bus is considered, however, the method will work in principle for Automotive Ethernet as well.

Any data that is sent through a digital medium must be encoded. This step is usually taken place within the ECU at the time that it intends to send a message on the bus. The actual data is sampled, translated using specific rules such that the communication partner can decode the information, and finally packed into the final CAN-FD message. Multiple signals may be packed into one CAN-FD frame to reduce the impact of the message overhead.

A commonly used standard to describe the encoding of bus messages is called SAE J1939 database can—or DBC for short [15]. It defines so-called signals, which bundle into signal groups, which together form a single CAN-FD message. A DBC file may describe the whole communication of a data bus, or even describe multiple busses.

---

**Fig. 2** Data flows within the vehicle and into a diagnostic system. The blocks represent different ECUs, the arrows indicate either point-to-point or bus communication. From [7]
2.2 Data congestion due to insufficient infrastructure

To specify the necessary performance of the presented method, it is assumed that the vehicle is connected to a cloud server primarily via mobile internet. Also, the operation of the vehicle is assumed to be eight hours of continuous driving during the day, while at the other time during the day, the vehicle is shut off. This will stop most of the data-generating components in the vehicle.

It is very likely that the diagnostic system will not be able to be still powered on to upload further data, as not to drain the starter battery [16]. On the other hand, if an overnight charging infrastructure is present, it can be assumed that the diagnostic system can transmit data during the night as well.

Therefore, there are two cases considered to determine whether data congestion is likely to happen. In the first case, all upload happens during driving hours via mobile internet, where a constant upload speed of 8 Mbit/s (German average) is assumed. This is a best-case assumption, as the actual speed will most likely be lower as argued before. In Fig. 3, you can see how a diagnostic system may have to cope with mobile access interruptions, as well as the varying quality of service if there is no static infrastructure present. The mobile internet case better represents driving that is carried out by customers, as these will most likely not connect their vehicle to WIFI or similar infrastructure during non-driving hours. However, this case could also represent test driving carried out by OEMs in remote locations like Scandinavia or Africa.

The second case is when an overnight charging infrastructure can be used to upload data at non-driving hours. The same overall upload speed is assumed, as the average upload speed of 8 Mbit/s incorporates stationary as well as mobile internet. This may not hold true for all infrastructures, as there is faster internet bandwidth available for cable-based internet. However, it can be argued that the usable bandwidth might be shared between multiple vehicles during charging.

In any case, this calculation should rather give a rough estimation. For both cases, it is assumed that the data is generated by a vehicle with a low automation level, which is equipped with five high-speed CAN bus systems. Each bus has a connection speed of 500 Kbits/s, so the overall data generation bandwidth is assumed to be up to 2.5 Mbit/s.

Although not considered in this paper, a second calculation is carried out, this time for a vehicle with automated driving capabilities. The vehicle is therefore assumed to have an Automotive Ethernet (AE) primary data bus with a data generation bandwidth of up to 100 Mbit/s.

Data congestion during the day arises when the daily upload bandwidth usage $U_{\text{daily}}$ is greater than one. This would mean that during the day, more data is generated than uploaded:

$$B_{\text{Generation}} = \text{Data generation bandwidth} \cdot \text{drive time}$$
$$B_{\text{upload}} = \text{data upload bandwidth} \cdot \text{available upload time}$$
$$U_{\text{daily}} = \frac{B_{\text{Generation}}}{B_{\text{upload}}}$$

This calculation assumes full usage of the available data generation bandwidth as well as data upload bandwidth. The data busses in actual vehicles will not be used up to full bandwidth and on the other hand, data upload will also not take place with all the available bandwidth due to protocol overhead. However, this will be neglected for this rough approximation.

As can be seen in Table 1, for low automation level vehicles, there is still headroom as the daily usage is less than one, even if there is no overnight charging infrastructure present. When looking at the automated driving vehicle case with Automotive Ethernet, the usage is roughly four to twelve. This means that without further methods, for

| Case                        | Infrastructure       |
|-----------------------------|----------------------|
| Low automation level vehicle | 0.1                  |
| Automated driving vehicle   | 4.2                  |

Fig. 3 The model for the diagnostic infrastructure includes the vehicle, the diagnostic system, the mobile internet access point, the cloud as central data storage, and the end-user accessing said data via the cloud servers.
automated driving vehicles it is infeasible to upload the data generated during the day within one day, meaning that data congestion is unavoidable. The infrastructure will certainly improve, but it is unlikely that it will catch up the speed with that vehicle automation and therefore in-vehicle data generation increase.

If data congestion is of no concern as there is either sufficient data storage available or if the infrastructure is greatly above average, the calculation will be too pessimistic. Nevertheless, it is desirable to be independent from hardware or infrastructure limitations as these drive up the cost of testing.

2.3 Current data transfer algorithms

There are numerous ways how to store recordings of vehicle data communication. In this paper, the focus is on CAN-FD communication only. The Linux Kernel incorporates the Socketcan library as a gateway towards a CAN-FD network [17]. A straightforward approach for uploading the vehicle communication is to store all received frames in the Socketcan native data format, bundling them into a block of frames, storing them in a file and then uploading said file. Therefore, the Socketcan protocol serves as a baseline for the comparison. One FD Frame as used in Socketcan takes up to 76 bytes per frame (see [18] for documentation of the format).

To complement the Socketcan messages, the exact time of arrival of each message is captured for the exact reconstruction of events. Signals are usually sent at a constant rate [19]. The start time of the cycle of each ECU is however different on every power cycle of the vehicle, and small inaccuracies in the ECU’s internal clock make for a drift of the cycle over time. Therefore, a timestamp for every message is stored with that message.

This baseline Socketcan encoding can be compressed by different approaches. There are two common approaches in literature: either by using common compression algorithms on a “per-message” basis [10, 20] or by exploiting common patterns of the messages being recorded [9]. These methods are lossless in that they can fully restore the original data. So-called “lossy compression” is also possible, for example by reducing the quality of measurements, but outside the scope of this paper.

For relevant compression algorithms, both Deflate as well as LZMA2 were considered. Both are general-purpose compression algorithms that are commonly used in many applications [21]. They both can effectively compress files that are generated in the mentioned Socketcan data format.

The standardized ASAM MDF4 format provides a different yet promising approach. It takes additional information about the raw communication and translates this into an alternate representation in the format of signals. These signals correspond to information bits that ECUs exchange on the vehicle bus. They can either take on physical information like temperatures or pressures, but also communicate control signals. The method that is presented in this paper tries to improve on this method.

3 Data compression preprocessing method

As proposed by other authors [8, 22], the data should first be preprocessed and then compressed before being transmitted via the mobile internet. The backend server collects the data, decompresses and finally reconstructs the original communication trace.

LZMA2 is used as the main compression algorithm, as it is well-documented, freely available and has outstanding compression capabilities [11, 21]. The primary compression mechanism of LZMA2 is run-length encoding, which in principle tries to find similarities and patterns within a series of bytes. It then stores the data in a less redundant representation. The effectiveness of the compression strongly depends on the data being compressed. For example, written language can be compressed well as there is redundancy and repeating patterns. Binary data like digital measurements are already present in a compact machine-readable format and therefore are harder to compress.

The preprocessing of the data is motivated through examining the data structure and trying to exploit certain patterns. ECUs are exchanging data via the bus, so the recording vastly consists of signals being transmitted as specified in the respective DBC definition from the vehicle manufacturer. Therefore, large parts of the communication are physical measurements or control signals. These signals are not random, and they can be compressed efficiently. Also, the timing information can be predictable in some circumstances, since signals are often sent at a predefined rate. Therefore, the time delta between two signal communications is almost constant.

Similarly, to the approach of the ASAM MDF4 data format, if a significant portion of the signals are known to the algorithm, it is possible to decode raw messages that may otherwise seem random, by the transformation of the data into their respective physical signal representation and storing them accordingly.

As shown in Fig. 4, there are certain message frames on the data bus that cannot be decoded using the DBC standard. Currently, these are handled separately and stored in an unaltered format in the same result file. The algorithm samples a predetermined amount of data over time and generates result files every few seconds.

The decoded part of the result file is further pre-processed to increase compression performance. Finally, all processed
data it is passed to the LZMA2 algorithm, which creates a compressed file for transmission to the backend.

### 3.1 Preprocessing

There are three techniques being used to increase the effectiveness of compression. The first one has already been mentioned. It is the technique of *Signal Transformation* (ST), which transforms the binary bus communication into meaningful signals.

The second technique takes advantage of the fact that the bus signals are transmitted in an absolute manner (according to the DBC standard), meaning that each encoded value $v_{encoded}$ can directly be translated into its decoded actual value using this formula:

$$v_{decoded} = v_{encoded} \cdot a + b$$  \hspace{1cm} (2)

The signal $v_{encoded}$ is transmitted as an unsigned integer of a fixed bit length, for example as 32-bit integer. However, if the value only slightly changes in each transmission, it is still transmitted with the full fixed bit length. Certain patterns like monotonically increasing or decreasing signals can be made more obvious to the LZMA2 algorithm. For this, they are transformed from absolute into relative format by storing and subtracting $v_{abs}^{-1}$, the predecessor value:

$$v_{relative} = v_{abs} - v_{abs}^{-1}$$  \hspace{1cm} (3)

The relative value must be stored as an unsigned integer in order to contain negative values as well. This technique is called *Differential Transform* (DT) since consecutive signals are stored relative to their predecessors. As an example, the absolute signal time series of \{1, 2, 3, 4, 3, 2, 1\} after the DT becomes the relative series \{+1, +1, +1, +1, −1, −1, −1\}, making the pattern much easier to detect.

The last technique being used is data alignment. The LZMA2 algorithm primarily works on byte patterns, but on the other hand DBC standard allows for signals of arbitrary bit length. Signals are padded with zeros until their bit length is divisible by eight. The technique is therefore called *Bit Padding* (BP). For example, if the signal time series of \{1, 1, 1, etc.\} is encoded in a three-bit number, its binary representation would be \{001, 001, 001, etc.\}. However, if this continuous bit string \{001001001 etc.\} is translated into a file before being passed to LZMA2, it is grouped into tuples of eight bits (one byte) and becomes \{00100100, 100100010, 01001000, etc.\}. As the LZMA2 algorithm primarily looks for byte patterns, this repeating bit pattern is harder to detect. By using BP on the values, the repeating pattern of \{00,00,00,00,00,00,00,00,00,00,00\}, etc.) becomes obvious again.

### 3.2 Results

The proposed method was evaluated on real vehicle communication during driving in an urban environment. The test drive consisted of several accelerations and stops, cruising periods and less than 30 s of still-stand. As vehicle, a mid-class, current production hybrid vehicle was retrofitted with an outside CAN-FD interface and the complete communication on the main powertrain bus was recorded for five minutes of continuous driving. Afterwards, the whole data file of 607,585 frames with a file size of 11.6 Mb in the *Socketcan* data format was compressed using state of the art compression as well as the proposed method.

The primary comparison metric used is the compression ratio, it is defined as following:
A higher value is better; for example, a compression ratio of five means that the necessary bandwidth to transmit the data is reduced to one fifth. For the proposed algorithm, two scenarios are considered: one where all possible signals on the communication bus are known (best) and one where there is no information about any signals (worst).

As seen in Fig. 5, conventional compression can achieve a compression rate of roughly factor six at most, while the proposed method in the optimal case can reach up to a factor of nearly ten. The “Per-message” compressions are disregarded here as their nominal average-case compression rate is lower than that of the deflate algorithm, which has the lowest compression rate in this comparison [9, 10, 20].

In the worst-case scenario, meaning that the presented method has no information about the signals on the communication bus, the method has a nearly identical compression rate as the LZMA2 base algorithm of roughly six. In the best-case scenario, the individual contributions of the three described preprocessing techniques are summarized in Fig. 6. The ST technique can boost the base compression ratio by +25.7%. When ST has been done, the compression rate can further be increased by an additional +18.6% using the DT. PB only marginally increases compression rates by +1.3% when used alone or by +1.1% in conjunction with DT. With all techniques applied, the compression ratio could be improved by over 50%.

Similar results could be expected for other vehicle data scenarios, as long as the communication is taking place primarily through the exchange of physical signals. The overall compression ratio depends on the type of information that is exchanged, so usually data from the powertrain bus has a better compression ratio than data from for example the media bus. Additional factors are the file size of the data to be compressed. Larger files generally speaking have a higher compression ratio due to a higher probability of internal data redundancy.

As an evaluation of the overall performance of the algorithm, the individual parts of the final compressed file have been analyzed to calculate the compression rate of each component. This analysis is carried out for the two components timing information and signal information. The timing information carries the timestamp in microseconds, at which the individual message arrived. The signal information describes the payload of the message in the formerly described transformed manner. Both together can

Compression ratio = \frac{\text{raw size}}{\text{compressed size}} \quad (4)

![Graph: Compression Rate (CR) Comparison](image)

![Graph: Influence of Preprocessing Techniques on Compression Rate (CR)](image)
be used to fully recover the original recording of the bus communication.

Figure 7 shows that the timing information takes up six times more space after compression compared to signal information, although both being at parity before compression. If the timing information could be discarded, for example, if the chronological order of the communication is of no interest, the theoretical compression rate would be nearly a factor of twenty.

4 Conclusions

The results of the proposed preprocessing for the LZMA2 compression algorithm look promising in the case that signal information is provided. This information may be provided by the OEM in the DBC data format. As this is sensitive information for an OEM, the signals may be anonymized by stripping off ECU and signal names from the DBC file, as well as randomizing certain conversion values as these do not affect the compression performance. There is however the possibility for reverse engineering the DBC files [23]. Improvements to the effectiveness of the preprocessing algorithm will be made in the future. As seen in this paper, the preprocessing drastically reduced the amount of storage space necessary for the raw data being transferred on the data busses. However, the timing information of the data messages, which is necessary to chronologically reconstruct the communication, cannot be compressed as efficiently. This leads to the conclusion that the timing information is currently not being stored in an optimal data format for compression and that further improvements can be made specifically on this part.

It can be concluded that significant improvements to the state-of-the-art compression algorithms can be made if signal information is incorporated into a preprocessor for the LZMA2 compression. The compression rate could be increased by +50% when compared to using LZMA2 without preprocessing and is almost double that of the comparable MF4 algorithm.

This presented result depends on several factors: file size of the raw data, the vehicle network architecture, whether the vehicle is currently driving or at standstill. Some factors, like the file size, can be changed dynamically to find an optimum. Other factors, like the vehicle network architecture, need to be further investigated. Statistical data on this subject is still sparse, but it appears as if the compression ratio gains that are achievable by the presented method are consistent across different vehicle architectures. These effects will be investigated in future research.

5 Outlook

Through the advent of automotive Ethernet, the peak bus speed will increase more than ten-fold, while on the other hand, the average upload bandwidth is not likely to increase at the same rate as massive infrastructure investments would have to be made. Therefore, in the future application, especially in automated driving, state-of-the-art compression will not be enough for mobile transmission of vehicle communication data.

The presented method still cannot fully provide the compression ratio to a level where high bandwidth communication like Automotive Ethernet can be transmitted with mobile internet alone. Intermediate storages are still necessary to cope with a large amount of data, which could be mitigated.

![Fig. 7 The proposed compression algorithm can compress signals significantly better than timing information](image-url)
by higher compression ratios through better compression methods. Alternatively, through the usage of static infrastructure like WIFI, this issue could be mitigated to some extent.

With further improvements to the method, it will be possible to remove the requirements on static infrastructure to be present during certain times, as these not only require storage hardware, but also introduce lag between data generation and data availability in the backend. This could reduce cost on diagnostic hardware as well as speed up the data analysis in the backend. The proposed method could allow data analysis tools in the backend to react to data anomalies within seconds, given that the diagnostic system has a decent mobile connection at that time.

However, the method still lacks sufficient compression ratios for future applications. Further development will take place in understanding the complex relationship between the data format and the achievable compression ratios. Ideas for future research include the exploitation of the periodic nature of most signals, which could if properly detected by the underlying compression algorithm greatly enhance the compression ratio for timing information.

Another field of further investigation will be the feasibility of the real-time data upload under the constraints of mobile internet. This research must include the factors that network downtime, upload interruption and time-variant upload speed impose on the system. In an actual data upload scenario, spurious interruptions to the data connection will worsen the achieved upload speed. Special care must be taken in order not to waste upload bandwidth by having to re-upload large chunks of data.

Therefore, it is necessary to further improve the methods being used for data collection and handling. Better data upload will enable the automotive industry to keep collecting as much data as possible to deliver safe and reliable vehicles to their customers.

**Funding**

Open Access funding enabled and organized by Projekt DEAL.

**Open Access**

This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

**References**

1. Cars are made of Code. https://www.nxp.com/company/blog/cars-are-made-of-code:BL-CARS-MADE-CODE. Accessed 1 Oct 2021
2. Tiedemann, S., Mank, A.: Solving the validation challenge of automated driving with a holistic test center. Automatisiertes Fahren 2019. Proceedings. 155–164. (2019). https://doi.org/10.1007/978-3-658-27990-5_14
3. Visualisations: Global Data. https://www.measurementlab.net/visualizations/. Accessed 18 Jan 2021
4. Van Leeuwen, T., Moerman, I., Rogier, H.: Broadband wireless communication in vehicles. J Commun Netw 2, 77–82 (2003)
5. Peon-Quiro, M., Mancuso, V., Comite, V.: Results from Running an Experiment as a Service Platform for Mobile Networks. Proceedings of the 11th Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization 9–16. (2017). https://doi.org/10.1145/3131473.3131485
6. Bock, U., Seifert, N., Schneider, L.: "Always-on"- Moderne Fahrzeug-IT als Basis fuer zukunftsweisenden Schienenpersonenverkehr. ETR - Eisenbahntechnische Rundschau. 58–61 (2015)
7. Goerne, L.: Improved Scalability of Vehicle Diagnostic Software: Advantages of Service-Oriented Architecture. Proceedings of 21. Internationales Stuttgarter Symposium. 563–574. (2021). https://doi.org/10.1007/978-3-658-33521-2_38
8. Abo-Zahhad, M., Rajoub, B.: An effective coding technique for the compression of one-dimensional signals using wavelet transforms. Med Eng Phys (2002). https://doi.org/10.1016/S1350-4533(02)00004-8
9. Andras G, G., Levente, B., Zsolt, S.: Efficient lossless compression of CAN traffic logs. 2017 25th International Conference on Software, Telecommunications and Computer Networks (SoftCOM). (2017). https://doi.org/10.1109/softcom.2017.796005-z
10. Grimm, D., Leiner, S., Sommer, M., Pistorius, F., Sax, E.: Flow-based Aggregation of CAN Frames with Compressed Payload. 2020 IEEE International Conference on Smart Computing (SMART-COMP) 202–207, (2020). https://doi.org/10.1109/SMARTCOMP50058.2020.00046
11. Cuiling, L., Xu, J., Zeng, W., Feng, W.: Compound image compression using lossless and lossy LZMA in HEVC. 2015 IEEE International Conference on Multimedia and Expo (ICME):1–6. (2015). https://doi.org/10.1109/ICME.2015.7177430
12. Manz, O.: Gut gepackt—kein bit zu viel. Springer Spektrum, Wiesbaden (2020). https://doi.org/10.1007/978-3-658-31216-9
13. Troh, A., Ill, B.: CAN Message Filter Algorithms for Remote Diagnostics of Vehicles. III Central European Conference on Logistics. 213–220. (2013). https://doi.org/10.4028/www.scientific.net/AMM.309.213
14. Schoenberger, L., von der Brueggen, G., Schirmeier, H., Chen, J.: Design Optimization for Hardware-Based Message Filters in Broadcast Buses. Design, Automation & Test in Europe Conference & Exhibition, 606–609. (2019).https://doi.org/10.23919/DATE.2019.8714791
15. SAE J939 Standards Collection. https://www.sae.org/standardsdev/groundvehicle/j939a.htm. Accessed 1 Oct 2021
16. Automotive Power Requirements Span From "A Little" To "A Lot". https://www.electrondesign.com/markets/automotive/article21797859/automotive-power-requirements-span-from-a-little-to-a-lot. Accessed 3 Jan 2022
17. Kleine-Budde, M.: Socketcan - the official can api of the linux kernel. IEEE International Conference of Communication (2012)
18. Readme file for the Controller Area Network Protocol Family (aka SocketCAN). https://www.kernel.org/doc/Documentation/networking/can.txt. Accessed 1 Oct 2021
19. Specification of Communication. https://www.autosar.org/fileadmin/user_upload/standards/classic/21-11/AUTOSAR_SWS_COM.pdf. Accessed 10 Mai 2022

20. Yazdani, N., Nielsen, L., Lucani, D.: Memory-aware Online Compression of CAN Bus Data for Future Vehicular Systems. GLOBECOM 2020 - 2020 IEEE Global Communications Conference. 1–6. (2020). https://doi.org/10.1109/GLOBECOM42002.2020.9348074

21. Quick Benchmark: Gzip vs Bzip2 vs LZMA vs XZ vs LZ4 vs LZO. https://catchchallenge.first-world.info/wiki/Quick_Benchmark_-_Gzip_vs_Bzip2_vs_LZMA_vs_XZ_vs_LZ4_vs_LZO. Accessed 1 Oct 2021

22. Kim, Y., Yang, Z., Chung, K.: Improved CAN compression algorithm by data reordering. 2017 International SoC Design Conference (ISOCC). 264–265. (2017). https://doi.org/10.1109/ISOCC.2017.8368886

23. Mert, P.: LibreCAN: Automated CAN Message Translator. Proceedings of the 2019 ACM SIGSAC Conference on Computer. 2283–2300. (2019). https://doi.org/10.1145/3319535.3363190

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.