Performance Comparison of Data Serialization Schemes for ETSI ITS Car-to-X Communication Systems

Sebastian Bittl, Arturo A. Gonzalez, Michael Spähn, and Wolf A. Heidrich
Fraunhofer ESK
Munich, Germany
Email: {sebastian.bittl, arturo.gonzalez, michael.spaehn, wolf.heidrich}@esk.fraunhofer.de

***Abstract*—Wireless Car-to-X communication is about to enter the mass market in upcoming years. The non-licensed, but reserved bandwidth for such communications is relatively narrow in terms of the few usable channels and the expected number of communicating entities. Among other communication aspects, data serialization schemes that allow compact encoding as well as fast encoding and decoding are required. Compact representation of data helps in keeping a minimal bandwidth overhead in the wireless channel. Moreover, since delay is an important performance parameter in Car-to-X communication, the processing time of the serialization/deserialization schemes shall be kept to a minimum. So far, no detailed analysis of different data serialization schemes for Car-to-X communication regarding important properties such as runtime, memory consumption and encoded output length has been published. In this work, we provide a performance comparison analysis between the standardized ASN.1 , the binary representations, Google Protocol Buffers and Efficient XML Interchange format (EXI), all of them as alternative strategies for data serialization. Standardized data content for CAM, DENM and the security envelope are used in the conducted study. We conclude that ASN.1 encoding on the facility layer shows the best performance, outperforming Google Protocol Buffers and EXI. However, for the case of encoding the security envelope, ASN.1 is outperformed by a binary encoding scheme in most cases, while EXI encoding outperforms all other schemes. This implies that standardization efforts for the security envelope should reconsider the recent shift from binary encoding towards usage of ASN.1. Instead, the Efficient XML Interchange format should be considered for this purpose.**

Keywords-ETSI ITS; data serialization; ASN.1; Google Protocol Buffers; Efficient XML Interchange format.

I. INTRODUCTION

Car-to-X (C2X) communication systems, often called Vehicular Ad-Hoc Networks (VANETs), are gaining increased attention in the awake of their upcoming deployment. Considering the high number of vehicles that are to be interconnected and the narrow radio spectrum being a limited resource for communication, efficient data representation is an important aspect, which has not received much attention prior to our work in [1].

However, the kind of data representation used for sending content over the wireless channel significantly influences the overall network performance. One of the key impacts is the one on the availability of the channel, since a longer packet implies a longer usage of the spectrum at a given transmission rate. Since the ITS-G5 and WAVE systems both use a carrier sense collision avoidance mechanism (CSMA-CA) for multi-user channel access, longer packets imply longer transmission times and hence a higher channel busy ratio. This means that nodes near a transmitter, which are willing to transmit as well, must wait longer until the channel is free to use. Moreover, the wireless channel has a limited capacity allowing only a limited number of transmitted bits per unit of time. The lower the number of transmitted messages, the higher the number of users that could use the given frequency spectrum in parallel.

The data serialization schemes also influence processing power requirements in both ETSI Intelligent Transport Systems (ITS) in Europe [2] and Wireless Access in Vehicular Environments (WAVE) in the United States [3].

Like most modern communication systems, C2X communication happens digitally. The latter implies that messages between the involved nodes are represented as a series of bits, i.e., as bit streams, in a platform independent way. As in any software implementation of a communication system, the format of the messages exchanged between two communication end points must be well known by them. That means that nodes should be able to represent messages as bit streams and to interpret them as the original messages as well. The generation of a bit stream from a message is defined as encoding. Hence, we refer to an encoded message as the bit stream representation of such a message. Following the same logic, decoding is defined as the (re-)generation of the original message from its bit stream representation.

Several encoding schemes exist nowadays, and some of them are used extensively in everyday data communications. Depending on the application requirements, one scheme may be suited better than another. The requirements for these encoding schemes range from human readability (e.g., XML [4], JSON [5]) or the space the bit stream takes up in memory (e.g., ASN.1 PER encoding, binary encoding), up to system performance, i.e., encoding/decoding processing delay (e.g., binary encoding, ASN.1 OER encoding).

In the C2X realm, it is significantly relevant to use a bandwidth efficient encoding scheme since C2X communications operate under quite strict bandwidth constraints. As an example, in Europe only one 10 MHz channel is available for safety critical applications [6]. Therefore, an encoding scheme that generates short bit streams out of messages is favored. Moreover, safety related C2X applications have strict end-to-end delay requirements. Therefore, encoding/decoding delays

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

should be minimal such that their contribution to the end-to-end delay may be considered negligible.

In this work, we focus on the comparison of the performance metrics of three coding schemes, namely Abstract Syntax Notation 1 (ASN.1) encoding rules, Google Protocol Buffers (protobuf) and the Efficient XML Interchange format (EXI). We apply the mentioned data serialization schemes to the two most common C2X message types in ETSI ITS systems, which are Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM). Furthermore, four different encoding schemes for the ETSI ITS security envelope are compared, which are binary encoding, ASN.1 encoding rules, protobuf and EXI.

The remaining part of this work is organized as follows. An overview of related work is given in Section II. Afterwards, performance requirements and the applied measurement mechanisms are described in detail in Section III. The target platforms' description is summarized in Section IV. The obtained results of the extensive performance study are described in Section V. Finally, Section VI provides a conclusion about the achieved results.

II. Background

The background of this work regarding platform independent data encoding, especially in the area of ETSI ITS, is provided in this section. Additionally, a comparison to the limited number of other published performance studies in the area of this work is given.

A. Data Encoding Rules

Cooperative Awareness Messages (CAMs) and Decentralized Environmental Notification Messages (DENMs) are the two most important standardized ETSI ITS messages defined in [7] and [8], respectively. A CAM contains basic information about the transmitting vehicle, for example position and speed. Depending on the purpose of the C2X application, received CAMs are processed in specific ways. One example of an application consuming the information contained in CAMs, is the use case of a C2X based collision risk warning. Here, the receiver nodes asses the risk of a collision with the transmitter given the speed, position and heading information of the sender, which is included in the CAM. The CAM generation period is defined in the standard to be between 1 and 10 Hz, being able to adapt depending on several parameters, such as vehicle velocity and channel busy ratio [7].

In contrast to CAMs, DENMs are event-triggered messages. As its name suggest, a DENM contains information about an event in the vicinity of the originating ITS-Station (ITS-S). One example of an event could be a stationary vehicle on the road. In this case, the stationary vehicle generates a DENM and depending on the configuration might broadcast this message periodically afterwards. The contents of the DENM include, among other information, the position and type of the event [8].

According to ETSI ITS standards, the encoding of CAM and DENM is done using ASN.1 UPER (Unaligned Packed Encoding Rules) encoding rules. There are several encoding rules specified for ASN.1, including Basic Encoding Rules (BER), Packed Encoding Rules (PER), Canonical Encoding Rules (CER), Distinguished Encoding Rules (DER), Octet Encoding Rules (OER) and XML Encoding Rules (XER) among many other flavors. Each of them is providing advantages and disadvantages from the point of view of a specific application.

Since PER provides a more compact encoded message than the older BER and its subsets DER and CER, it is often used in systems where bandwidth conservation is important [9]. This might be the reason why the CAM and DENM standards specify that the encoding rules to be used should be Unaligned PER (UPER). In aligned PER fields are aligned to 8-bit octet boundaries by inserting padding bits, whereas in UPER padding bits are never inserted between fields, hence allowing a higher bit stream size reduction.

A widely used alternative to ASN.1 encoding rules are the so called Google Protocol Buffers (protobuf), which are used by Google extensively in their production environment [10], [11]. Therefore, they can be regarded as a stable and reliable mechanism. Protobuf offers a more simplistic approach to the platform independent data encoding, making it easier to manipulate and implement [10]. Additionally, they can be configured to do encoding optimized for either fast processing or small memory footprint. The latter is also a common feature provided by standard ASN.1 implementations. For example, the software provided by OSS Nokalva provides this feature [9]. Therefore, protobuf can be seen a well comparable alternative to ASN.1 for data representation. Hence, the performance study provided in Section V includes usage of protobuf.

Another data representation technology, which is well-known in the automotive domain, is the so called binary Extensible Markup Language (XML) via the Efficient XML Interchange (EXI) standard [12]. EXI is used for the communication between electric vehicles and charging stations during the charging process as defined in the new ISO 15118 standard [13]. EXI is a machine-to-machine protocol, which aims in removing all overhead coming along with XML, to enable human readability such as indentation, whitespace and even byte alignment (like for ASN.1 UPER). In schema less mode, EXI tries to minimize overhead by the use of string tables, thus reduces encoding size very rapidly for repetitive message structures.

Moreover, EXI also allows the usage of XML schemes on the sender and the receiver, which allows to share common knowledge about the structure of the transferred messages in a convenient and flexible way. EXI encoding and decoding happens in a linear way, so that decoding can already start when only a first part of the full message has been received. Thus, EXI can be regarded a proper alternative to ASN.1 UPER and is therefore included in our performance study.

For the ETSI ITS security envelope two different sets of encoding rules have been proposed so far. At first, binary encoding with explicit definition of all data fields was proposed in [14]. Additionally, encoding using ASN.1 UPER was proposed recently in [15]. However, no in detail comparison of these two proposals is available, except of our prior work in [1]. As further reference schemes, protobuf and EXI are also used for the security envelope in the performance study, presented in Section V.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

There are important differences to consider when comparing different data representation schemes. Some of the most relevant are,

- how are optional fields within messages handled, i.e., how is a field's presence or absence represented,
- possibility of future backward compatibility when extending a message, i.e., adding of new mandatory or optional data fields,
- byte alignment,
- providing the functionality of data compression, for example variable length representation of integers.

These differences allow many different combinations which could be used and are actually to be found in various practically used mechanisms. An overview of this set of basic properties for the data representation schemes used later in this work is provided in Table I.

TABLE I. OVERVIEW OF BASIC PROPERTIES FOR DIFFERENT SCHEMES OF PLATFORM INDEPENDENT DATA REPRESENTATION.

	binary [14]	ASN.1 UPER	protobuf [16]	EXI
presence of optional fields	not encoded	encoded	encoded	encoded
extendability	no	no	yes	yes
byte alignment	yes	no	yes	no
compression	normally no only one type	yes	yes byte blocks	yes byte blocks

As a comparison example, the standardized binary encoding for the ETSI ITS security envelope uses no explicit encoding of optional fields [14], allowing no possibility of a backward compatible extension. It also uses data compression only for one specific data type (IntX [14]), which is a variable length integer type. This type is almost exclusively used to encode the length of data fields and not for encoding real data content. Protobuf and EXI both use variable length encoding for integers, but the size of an integer still has to be an integer multiple of one byte.

In regard to extendability, protobuf provides a backward compatibility property, which allows to add message structures without any change to the encoding and decoding code. EXI has a similar feature when using schema files for message specification. Here the schema file might be updated in a way that the old message specification is still valid (e.g., by adding an optional element). In this case, no change in the encoding and decoding code is necessary either.

Alternative publicly available data serialization tools for converting arbitrary data into a platform independent binary representation include systems like Apache Avro, Apache Thrift or Message Pack [17]–[19]. These systems are either less mature or deployed to a much smaller extent in professional environments compared to ASN.1 and protobuf (e.g., see [20] for protobuf vs. Thrift). Therefore, they are not studied in detail in this work. Additionally, serialization technologies like XML or JSON, which aim to achieve a human readable and easy to parse data representation at the price of increased encoding length are out of the scope of this work. Such systems are not appropriate to be used in bandwidth constrained communication systems.

B. State of the Art and Contribution of this Work

There are several publications comparing other encoding schemes, such as XML with ASN.1. For example, the authors in [21] compare the performance between binary encoded XML and ASN.1 by running the tests on PC machines. In [22], the authors compare the performance of XML against ASN.1 BER on digitally signed data. They conclude that for applications where high performance is required, ASN.1 BER may be a better choice.

In [23] authors compare the performance of XML, JSON and protobuf in terms of data size and coding speed. The authors conclude that protobuf requires less bytes for the message representation in comparison with XML or JSON. The authors also explore the possibility of compressing XML and JSON messages using gzip [24]. In the latter case, both compressed text formats perform better than protobuf in terms of data size. In terms of speed, the authors show that protobuf performs better than both text schemes.

In [25], authors perform a similar study to the one in [23] and expand it for performance in energy consumption, relevant for a smartphone use case. They also show that gzip-compressed protobuf, a variant not explored in [23], performs better in terms of encoded data size in comparison with compressed XML, but worse than compressed JSON. When the authors measure performance in respect to encoding time, they conclude that for the data set they used protobuf performs better. On the parsing process on the receiver side, i.e., decoding, JSON performs slightly better than the other two schemes.

A performance comparison between gzip-XML as well as ASN.1 PER against EXI is provided in [26], where it is shown that EXI greatly outperforms both other schemes for the used test data set. In [27] Peintner et al. highlight the advantages of schema-enabled EXI in the domain of multimedia applications for embedded systems over the use of plain XML in respect of encoding and decoding performance as well as compression. They especially focus on the encoding of SVG vector graphics and introduce an approach for a more efficient data type representation in combination with EXI in this domain.

To the understanding of the authors at the time of writing this work, there are no previous studies focusing on a quantitative comparison of performance measurements between ASN.1 UPER, protobuf and EXI, specifically on the field of C2X communications. Although, ETSI ITS standards for the facility layer define the encoding mechanisms as ASN.1, this work should provide some insight for the viability of an alternative based on an open source development (i.e., protobuf) or on EXI.

An alternative suggestion to binary encoding of the security envelope using ASN.1 encoding has been proposed in a recent ETSI ITS draft standard [15]. However, there are currently no performance studies available providing insights on which alternative should be selected. Moreover, EXI encoding has not been considered for ETSI ITS data representation so far. Another contribution of this work is to provide some information on the performance comparison of these encoding schemes on different computing platforms such as embedded systems.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

III. Performance Requirements and Measurements

The performance metrics considered in the evaluation of the different encoding schemes are

1) computation time,
2) memory footprint on computation and
3) encoded data length.

Aspects 1 and 2 clearly focus on the required computing power for the encoding and decoding process. As ETSI ITS technology shall be implemented in embedded systems, e.g., in vehicles, these criteria are quite important due to the limited resources typically available in such systems. The used tools and methodology to measure these kind of metrics are described in Section IV-B.

The length of the encoded data is a criteria which mostly influences the required communication bandwidth on the wireless channel. It directly determines how long it takes to communicate a data packet over the air. Given that a communication channel has a limited capacity, the length of the encoded messages directly influences the number of possible transmissions over the air in a specific time span. Additionally, ETSI ITS uses only a single control channel to distribute important CAMs and DENMs. Therefore, an increased size of the encoded data packet directly leads to a decrease in system performance and scalability.

IV. Target Platforms

In order to obtain reliable results for our performance study, different hardware platforms with a common software configuration have been used. Details about the used hardware are given in Section IV-A, while the software framework is discussed in Section IV-B.

A. Hardware

To execute our performance measurements of the data serialization schemes in question, we have used three different platforms. The reason is to show the influence of different used hardware technologies as well as to exclude effects on the overall performance study caused by a single processor technology. Table II summarizes the main characteristics of the three platforms used during our experiments.

TABLE II. Used CPU hardware and achievable measurement accuracy via Linux clock counters.

type	AMD Geode LX	Intel Atom Z520PT	Intel Core i7-2640M
clock speed	500 MHz	1.33 GHz	2.8 GHz
clock res.	2 ns	1 ns	1 ns

More details about the used processor technologies can be found in references [28]–[30].

The clock resolution given in Table II was obtained by using the clock_getres() [31] function on the individual platforms in the software environment described in the next section.

B. Software

On all platforms, a standard Debian Linux [32] system with kernel version 3.16.0 was used as the underlying operating system during the performance study. Furthermore, ASN.1 related functionality was provided by the FFASN1 Compiler [33]. Additionally, correctness of encoding as well as the encoded data length was double checked with the ASN.1 library from OSS Nokalva [9]. Protobuf was used in version 2.5.0 as provided by the Debian distribution.

EXI related functionality was provided by the Embeddable EXI Processor in C (exip) library [34]. A double check of the correctness of the encoding and the preparation of the schemes for exip have been done with the Java-based OpenEXI software [35]. For binary encoding of the security envelope the implementation from the ezCar2X framework [36] was used. All used software was compiled on the target with the GCC compiler version 4.8.2 [37]. Thereby, strong optimization was enabled with the *-O3* compiler flag.

In contrast to our prior work in [1], we focus the performance study in this work on the time optimized (TOED) implementation variants of the regarded data (de-)serialization schemes. This is done, as the results in [1] clearly show that computational processing on all used target platforms is bound by pure CPU speed. Thus, the TOED implementations clearly outperform their space optimized (SOED) counterparts in regard to all considered performance criteria. For more details about this aspect the reader is referred to [1].

For timing measurements, the Linux kernel high performance counters have been used, which can be accessed from user space by calling the clock_gettime() function [31]. Thereby, CLOCK_PROCESS_CPUTIME_ID was used as the clock ID in order to determine only the time spent in the process, which contains the algorithm to be measured. An accuracy of up to 1 ns can be achieved, if the underlying hardware permits such accurate measurements [38]. In order to make the measurements more accurate, the suggestions from [39] for avoiding effects of out-of-order execution have been applied. Therefore, the CPUID instruction was executed before and after calling the clock_gettime() function.

The described methodology for time measurements is preferred over directly reading the CPU's time stamp counter (TSC), which is for example used in [39]. The reason is that while [39] uses operations only available inside the Linux kernel, the measurements in our performance study are done in the user space. Therefore, certain prerequisites of the approach from [39] like disabling of interrupts or scheduling cannot be fulfilled. Hence, we rely on the implementation of the clock counter in the Linux kernel.

An algorithm's main memory footprint (heap as well as stack usage) was measured by the help of the so called *malloc_count* framework [40]. This framework allows arbitrary parts of a program to be traced by inserting dedicated function calls into it. These calls where only used during memory measurements and were removed during timing measurements as they would introduce overhead. Other memory tracing tools like *massiv* from the valgrind framework [41] do not allow adjustment of the measurement procedure with such fine granularity. Therefore, malloc_count was used to obtain the results presented in Section V-C.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

V. Performance Study

The conducted performance study is discussed in detail in this section. Firstly, the data content to be used for (de-)serialization is described in Section V-A. Secondly, the procedure for generating encoding rules for serialization schemes not present in current standards (i.e., protobuf and EXI) is introduced in Section V-B. Finally, Section V-C provides the results obtained in the extensive performance study, using the methodology from Section IV on content described in Sections V-A and V-B.

A. Content for Encoding and Decoding

For this performance study, we have selected the subset of fields that are defined as mandatory in the standards of CAM and DENM. This implies that the results showed here provide a lower bound of performance of all the considered encoding schemes. For these messages, we have used real data within the message content as far as possible, e.g., we included real time stamps and coordinates.

The studied security envelopes consist of the message fields as specified in [14] and [15], where all defined security profiles (three in total) are taken into consideration. Additionally, for the CAM security profile (security profile number 1) two cases have to be distinguished. The corresponding envelope can hold a signed certificate or just an eight byte hash value of the certificate. Both cases have been included in the performance study.

It should be noted, that the depth of nested data structures significantly affects the performance of encoding and decoding mechanisms. Thus, an overview of the hierarchy of data elements (often called containers in the ETSI ITS context) is given in Table III. Four different cases are considered for the security envelope, which relate to the three different security profiles from [14].

TABLE III. Overview of nesting of data fields for CAM, DENM and security envelope.

nesting level	1	2	3	4	5
CAM	2	4	4	20	10
DENM	4	8	4	0	0
sec. profile 1 (CAM) w/o cert.	5	16	4	0	0
sec. profile 1 (CAM) w. cert.	5	22	21	11	0
sec. profile 2 (DENM)	5	22	23	11	0
sec. profile 3 (Generic)	5	22	22	11	0

The numbers in Table III give the amount of data sets (mandatory and optional) found at the different nesting levels. To obtain the figures in Table III, the full data sets were represented in a tree structure. As we use only mandatory fields in our performance study, the elements of sub-trees following an optional element are not counted. Nesting level one means the top level of the data packet, whereas nesting level five relates to the data elements at the most deeply nested position inside the data packet.

In order to separate the security component tests from others, no real payload was used on these tests. For the case of binary encoding, the envelope only includes the mandatory one byte dummy payload as specified in the standard [14].

Listing 1. VerificationKey element from the security envelope as implemented according to the standard.

```
<s0:VerificationKey>
  <s0:Key>
    <s0:EcdsaNistp256WithSha256>
      <s0:publicKey>
        <s0:CompressedLsbY0>
          <s0:x>FFFFFFFF</s0:x>
        </s0:CompressedLsbY0>
      </s0:publicKey>
    </s0:EcdsaNistp256WithSha256>
  </s0:Key>
</s0:VerificationKey>
```

Listing 2. Optimized VerificationKey element from the security envelope.

```
<s0:SubjectAttributeVerificationKeyEcdsa
NistP256WithSha256CompressedLsbY0>FFFFFFFF
</s0:SubjectAttributeVerificationKeyEcdsa
NistP256WithSha256CompressedLsbY0>
```

B. Encoding Rules for Google Protocol Buffers and Efficient XML Interchange Format

The definition files for protobuf and the XML scheme files for EXI were derived from the ASN.1 definitions given in standards [7], [8], [15]. Transformation from ASN.1 definitions to protobuf and EXI is straightforward due to the low number of available data types in both. During the transformation process always the smallest protobuf (or EXI) data type, which is able to hold the corresponding ASN.1 data type, was selected to avoid introducing unnecessary overhead.

Protobuf does not provide a data element for choices, thus all possible subjects of a choice where chosen to be optional elements. This also means that, the protobuf library does not provide any possibility to check whether exactly one of the to be chosen elements was actually chosen. Thus, this check is left to the user of the auto-generated code.

In the performance study case for EXI, two approaches were followed. At first, a full mapping of the standard to an EXI schema has been developed. These schema files contain a lot of nesting levels, leading often to (informationally) unnecessary content. On the other hand, this makes the existing schemes easy to expand and very structured. However, in all cases the full schema description has to be updated on both sender and receiver when introducing fundamental changes in the message structure. Since one of the key parameters in this study is the size of the encoded messages, we opted for introducing data optimized schemes. In other words, in the schema files the unnecessary nesting levels are merged, thus decreasing the number of options in the EXI grammars. The difference in respect to XML structure of the elements can be seen exemplarily in Listings 1 and 2.

As one can see from comparing Listings 1 and 2, the number of tags required for storing the same amount of payload is reduced from six to only one. Thus, this clearly reduces the amount of metadata in the serialized data, which leads to reduced encoding length.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

C. Results of Performance Study

The results of the conducted performance study regarding memory consumption and encoded output length are summarized below. At first, in Tables IV (CAM), V (DENM), and VI (security envelope) results for encoding (i.e., data serialization) are given. Second, results for decoding (i.e., data deserialization) are provided in Tables VIII and IX. In the following, individual results for these message contents are analyzed in detail.

Memory requirements, as well as encoding length are independent of the used CPU architecture. Therefore, just a single result is given for these criteria in the following analysis. Runtime performance, which is clearly processor specific, is presented later on.

In the following analysis, all encoding lengths and memory consumption measurement results are given in bytes. To provide a fair comparison, all memory consumption and runtime measurements for the EXI data serialization scheme give the results for normal, i.e., non optimized, data representation. At first, the results for data encoding, i.e., serialization into the data format transmitted over the network, are given in Section V-C1. A discussion about the results for data decoding, i.e., deserialization of data received from the network follows in Section V-C2.

1) Data Encoding: Data encoding (i.e., serialization) is studied in the following, as it happens at the sender side of a communication connection.

At first, encoding performance for CAMs is studied in detail. The achieved results are summarized in Table IV. Thereby, the value in brackets in the EXI column gives the achieved message size for the case of using the optimized message definition set as introduced in Section V-B.

TABLE IV. ENCODING PERFORMANCE RESULTS FOR CAMS.

	protobuf	ASN.1	EXI
heap / stack	242 / 1864	66 / 3112	62656 / 210
encoded length	165	41	64 (opt: 61)

From Table IV, one can see clearly that protobuf generates almost four times more output bytes than ASN.1 for an encoded CAM. Both the standard as well as the optimization variants of EXI encoding are clearly outperformed by ASN.1, but achieve smaller encoding size than protobuf.

The generated protobuf code uses less memory (cumulative heap and stack) than the ASN.1 implementation. From the measurement results, it is clear that both protobuf and ASN.1, outperform the EXI implementation in regard to memory usage. This is because, the chosen EXI implementation does not use any static a-priori knowledge like the auto-generated code used for ASN.1 and protobuf. Instead, the used library builds up all required trees for encoding on demand in memory. This clearly leads to increased memory consumption and runtime, as one can see from runtime measurement results given in the following. Thus, for a production system one would choose to use a less flexible, but more memory and runtime efficient implementation.

We now study the encoding performance of DENMs. The corresponding results are given in Table V.

TABLE V. ENCODING PERFORMANCE RESULTS FOR DENMS.

enc. type	protobuf	ASN.1	EXI
heap / stack	126 / 1752	75 / 2792	61608 / 175
encoded length	114	43	52 (opt: 51)

As one can clearly see, the memory consumption is similar to the encoding of CAMs but somewhat lower. This is in line with the smaller size of encoded data. As less data has to be encoded, a lower memory consumption can be expected. Additionally, the protobuf encoding shows again the smallest memory footprint of all of the shown four encoding schemes. Furthermore, protobuf performs worst in encoded length, however it only needs roughly three times as much space as ASN.1 compared to almost four times for CAMs.

Moreover, the difference between encoding lengths for ASN.1 and EXI in Table V is significantly smaller than for the CAM case. From studying the different encoding rules, one can see that protobuf introduces more overhead for deep nesting structures than ASN.1 does. From the data analysis provided in Table III, one can see that CAM uses much more and much deeper nested data structures compared to their counterparts in DENM and security envelope. Thus, the difference between the overhead caused by protobuf for the CAM and DENM data sets is as can be expected.

Table VI gives the performance results for main memory consumption as well as encoding length for the ETSI ITS security envelope.

TABLE VI. ENCODING PERFORMANCE RESULTS FOR THE SECURITY ENVELOPE.

enc. type	profile	heap/stack	enc. length
binary	1 no cert.	240 / 12168	96
	1 cert.	798 / 15800	222
	2	798 / 15800	233
	3	798 / 15800	230
protobuf	1 no cert.	1784 / 13528	133
	1 cert.	3819 / 15016	306
	2	4023 / 15016	318
	3	3865 / 15016	312
ASN.1	1 no cert.	1463 / 19784	88
	1 cert.	2186 / 20528	240
	2	2186 / 20528	249
	3	2186 / 20528	247
EXI	1 no cert.	61760 / 680	90 (opt: 87)
	1 cert.	63313 / 680	210 (opt: 201)
	2	63553 / 680	215 (opt: 206)
	3	63457 / 680	213 (opt: 204)

In Table VI the profile column gives the number of the applied security profile as defined in [14]. As described above in Section V-A, the two cases of an envelope with and without certificate have to be distinguished for security profile number one (used for CAMs).

Comparing the overhead introduced by protobuf encoding, its size is between the overheads for CAM (being larger) and the one for DENM (being smaller). Such behavior can be expected, as the nesting of the security envelope is deeper than the one for DENM, but no such deep as for CAM (see also Table III).

The encoding lengths for security profiles two and three are only different for the case of binary encoding and not for ASN.1 encoding, as the data field called *message type* is optional according to [14] but required according to the ASN.1

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

definition given in [15]. As the only difference between these two security profiles is the presence of the message type data field, this difference vanishes in the case of ASN.1 encoding. Therefore, memory consumption is identical for these two cases. In order for a difference between the two security profiles to exist, our protobuf and EXI definitions declare the message type field as being optional. From a semantical perspective, it makes no sense to give a message type in case of profile number three, as this is the default profile for messages of type *Generic* [14].

One can see from the most right column of Table VI that, in all cases binary encoding clearly outperforms protobuf in respect to achieved encoding length. Additionally, it outperforms ASN.1 encoding in three out of four cases, the only exception being the case of security profile number 1 without certificate. In this case, ASN.1 encoding uses only 9 bytes less than binary encoding. However, for the case with certificate and security profile one, ASN.1 requires 19 more bytes than binary encoding. Furthermore, binary encoding requires 18 bytes less for security profile number two against ASN.1 and 21 bytes less for security profile number 3, respectively.

The results from Table VI clearly show that the normal EXI encoding scheme achieves the smallest packet size for security profile one with certificate as well as profiles two and three. Additionally, it outperforms binary and protobuf encoding for the case of security profile one without certificate and is only slightly outperformed by the ASN.1 encoding scheme. However, the optimized variant of EXI encoding significantly outperforms all other schemes in regard to message size.

In order to decide which encoding scheme performs best for security profile one, the average size of the security envelope should be considered. Due to varying CAM emission frequency (from 1 to 10 Hz) and the different certificate inclusion rules (see [14]), only a lower limit for the average size of the security envelope for profile one can be given. The average size of the security envelope $\overline{s}_{sec}$ is to be calculated by

$$\overline{s}_{sec} = \frac{(f_{CAM} - f_{cert}) \cdot s_{w/o} + f_{cert} \cdot s_w}{f_{CAM}}; f_{cert} \leq f_{CAM}$$

Thereby, the size of the security envelope without certificate is denoted by $s_{w/o}$ and the one with included certificate by s_w. To calculate the metric of the lower limit of $\overline{s}_{sec}$, the maximum CAM emission frequency f_{CAM}, of 10 Hz, together with the minimum certificate inclusion frequency f_{cert}, of 1 Hz, is used. The different values of this metric for the regarded encoding schemes are given in Table VII.

TABLE VII. MINIMUM AVERAGE SIZE OF THE SECURITY ENVELOPE FOR CAMS (SECURITY PROFILE ONE).

encoding scheme	binary	protobuf	ASN.1	EXI
$\min(\overline{s}_{sec})$	108.6	150.3	103.2	102 (opt: 98.4)

One can see from the results provided in Table VII that EXI encoding achieves the best minimum average encoding length. This means, that with EXI encoding the average message size will always be smaller than the one for other encoding schemes, whatever CAM generation rate and certificate inclusion rates are applied. The saved message size for the optimized variant of EXI in comparison to the normal EXI encoding is an additional 3.53%. In comparison to the standardized binary encoding scheme, it even saves the significant amount of 9.39% in message size.

To obtain results for the computation time we ran the measurement procedure described in Section IV-B 10,000 times and computed the average of the measured outcome. Corresponding results for all processor types from Table II are shown in Figures 1, 2, and 3. Please note that the vertical axis of the graphs uses a logarithmic scale. Additionally, for binary encoding only four runtime measurement results are provided per processor as this scheme is not defined for encoding of CAMs and DENMs. Therefore, only the four different kinds of security envelope encoding have been measured.

An overview about the achieved runtime performance measurements on an Intel Core i7 processor is provided in Figure 1 (see also Table II).

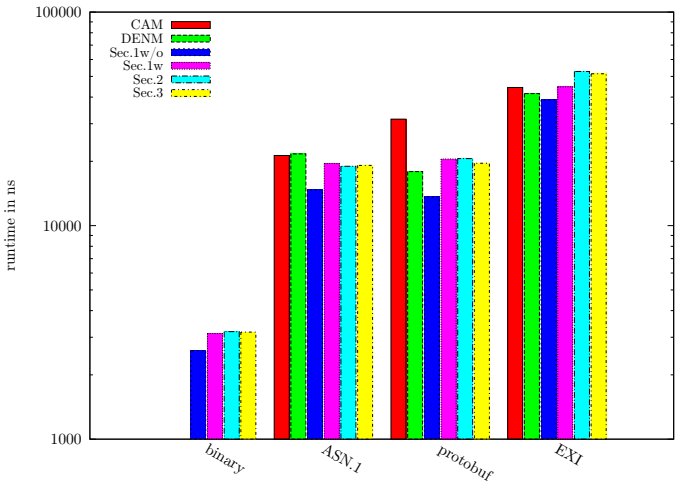

Figure 1. Encoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel Core i7 processor.

The obtained results clearly show that, for the security envelope, binary encoding is significantly faster than the two other encoding schemes. Additionally, ASN.1 encoding outperforms its protobuf and EXI counterparts.

A significant source of influence on runtime performance for encoding the security envelope is the high number of small and deeply nested data fields used for defining the security envelope (see also Table III). The achieved results depicted in Figure 1 indicate that binary encoding can handle this kind of structure better than the other encoding schemes can do. Moreover, ASN.1 and protobuf are almost on par and both clearly outperform the EXI mechanism.

To avoid overloading the figures, the computed standard deviation of the measured runtimes are not shown. In general the standard deviation was quite low, e.g., a value of 152 ns was found for binary encoding of the security envelope with security profile one without included certificate. The differences between the obtained results of different encoding schemes for same encoded data content are always bigger than three times the standard deviation of the corresponding runtimes. Therefore, the achieved measurement results can be regarded as reliable.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

The results obtained from the runtime measurements on an Intel Atom processor are depicted in Figure 2 (see also Table II).

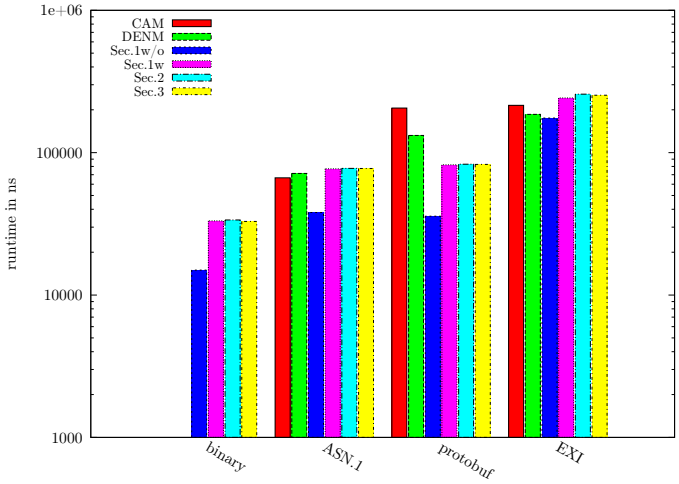

Figure 2. Encoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel Atom processor.

Comparing Figure 2 to Figure 1, one can see that except of a general increase in runtime (please note the different scaling of the vertical axis of both figures), the overall results are the same for the Atom and the Core i7 processor technology. Due to the lower processor speed (see also Table II) such an increase in runtime can be expected. However, the increase is somewhat bigger than what can be calculated by just determining the factor one obtains from dividing the respective processor clock speeds. It is reasonable to observe an advantage in the runtime performance of the Core i7, which is due to the improved processor technology such as precaching algorithms, as it was introduced to the market significantly later than the used Atom processor.

Finally, Figure 3 provides the results of runtime measurements conducted using an AMD Geode processor (see also Table II).

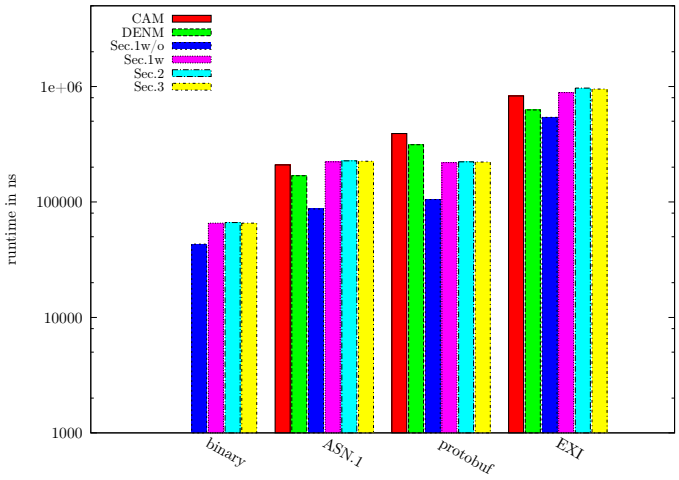

Figure 3. Encoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an AMD Geode processor.

From the comparison of results shown in Figure 3 to the results given in Figures 1 and 2, one can see that the overall outcome of the performance study does not change by switching from a modern high speed processor (like the Core i7) to a quite old and low speed processor, like the AMD Geode.

Given the latter statement, we conclude that the achieved results can also be used to interpret the behavior of the studied encoding algorithms within embedded systems using medium speed processors, nevertheless low end, low power processors may possibly behave differently.

In summary, it has been shown that regarding runtime and memory consumption, binary encoding outperforms all other studied encoding schemes running on all platforms. Regarding ASN.1, it can only achieve a shorter encoding length than binary encoding in the case of security profile number 1 without certificate. However, for the security envelope case EXI encoding, especially the optimized variant, outperforms all other serialization schemes in regard to message length.

It is worth to note that, the default timing interval for including a certificate in the security envelope of a CAM is equal to the default sending interval of CAMs (see [14] [7]). The latter means that normally CAMs are sent with a certificate included in the envelope. Therefore, the results show that the newer standard [15] defining the security envelope using ASN.1 significantly deteriorates the performance of its encoding compared to the preceding standard [14], which uses a binary encoding scheme. Furthermore, as ASN.1 does not provide a forward compatibility functionality, like e.g., protobuf would do, there is almost no reason why one should prefer ASN.1 over binary encoding. Instead, one should consider EXI encoding to benefit of a shortening in encoded size of the security envelope of about 9.39%.

The conducted performance study also shows that protobuf cannot be seen as a real alternative to ASN.1 for ETSI ITS data encoding. Protobuf is outperformed by ASN.1 on almost all of the selected important performance criteria on any of the platforms used and for all kinds of data types considered. Protobuf was found to be somewhat smaller compared to the respective ASN.1 counterpart only on the memory footprint parameter for some kinds of data types. Nevertheless, also in those particular cases, protobuf is not able to outperform the binary encoding scheme of the security envelope.

2) Data Decoding: In the following, the results for data decoding (i.e., deserialization) are provided. These mechanisms are required at the receiver side of a message exchange. As the number of other ITS-Ss is typically quite high in a VANET and the majority of messages is broadcast traffic (e.g., CAMs and DENMs), message decoding happens much more often than message encoding. Thus, poor computational performance of a data representation scheme leads to significantly higher penalty at the receiver's side compared to the sender's side.

Table VIII provides an overview of the memory consumption regarding stack and heap usage of the different deserialization mechanisms for CAM and DENM data types.

The obtained results clearly show, that protobuf uses the least amount of memory (cumulative stack and heap) and EXI performs worst in regard to this criteria. Like for encoding

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

TABLE VIII. MEMORY RELATED DECODING PERFORMANCE RESULTS FOR CAMS AND DENMS.

	protobuf	ASN.1	EXI
CAM: heap / stack	242 / 1800	370 / 2968	3850 / 210
DENM: heap / stack	126 / 1624	816 / 2872	3630 / 135

results (see Tables IV and V), the EXI decoder uses the majority of its memory on the heap in contrast to both protobuf and ASN.1, which take the majority of their memory from the stack.

Table IX summarizes the results for memory usage of the different deserialization schemes for the security envelope.

TABLE IX. DECODING PERFORMANCE RESULTS FOR THE SECURITY ENVELOPE.

enc. type	profile	heap/stack
binary	1 no cert.	872 / 15480
	1 cert.	1709 / 19208
	2	1773 / 19208
	3	1717 / 19208
protobuf	1 no cert.	1916 / 19992
	1 cert.	3665 / 20632
	2	3869 / 20632
	3	3711 / 20632
ASN.1	1 no cert.	1296 / 13016
	1 cert.	4255 / 14040
	2	4327 / 14040
	3	4311 / 14040
EXI	1 no cert.	13375 / 1080
	1 cert.	14131 / 1100
	2	14195 / 1140
	3	14198 / 1136

In contrast to results for CAM and DEMN deserialization, for decoding of the security envelope ASN.1 uses less memory than protobuf, which showed the best performance for CAM as well as DENM. Furthermore, EXI exhibits the smallest memory footprint for all security profiles, significantly outperforming binary and ASN.1 decoding schemes. This clearly shows the dependence of a data (de-)serialization scheme on the used structure of the message.

Memory usage of EXI decoding is much smaller compared to encoding (see also Table VI). This is because the chosen decoder design does not try to build a full message tree in memory before returning the decoded message to the user. Instead, the approach is more like the one for simple binary decoding. The data packet is parsed element by element and for each primitive data type found (e.g., an integer) an a-priori registered callback function (provided by the user) is called. This usage of a-priori information clearly reduces memory consumption inside the decoding method significantly.

Runtime measurements of the different message decoding mechanisms on the three regarded hardware platforms are discussed in the following.

Figure 4 provides the results of runtime measurements on the Core i7 platform.

As one can see from Figure 4, binary encoding significantly outperforms its counterparts for all variants of the security envelope. However, for decoding the gap between binary and ASN.1 is smaller than for encoding (see also Figure 1).

Additionally, ASN.1 performs best for both CAM and DENM decoding. An interesting finding is that the performance gap between ASN.1 and protobuf as well as EXI

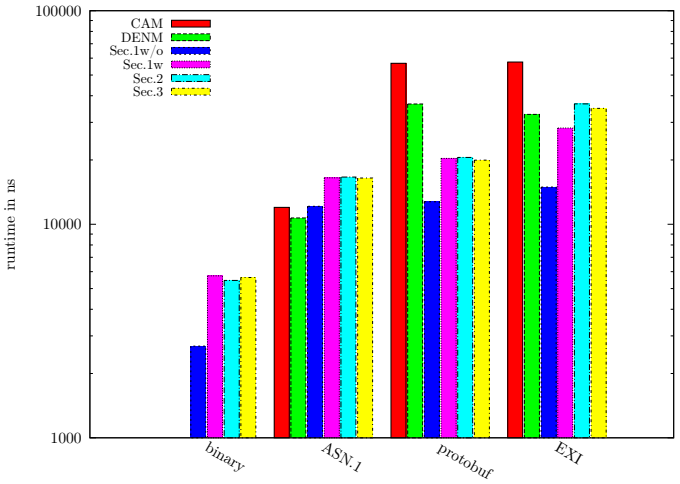

Figure 4. Decoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel Core i7 processor.

for CAM and DENM is much bigger than for the security envelope. What is more, protobuf performs poorly for the case of CAM and DENM. EXI shows the worst decoding runtime performance for the security envelope. Furthermore, for protobuf and EXI, decoding of CAM and DENM takes longer than decoding of a security envelope with profile one with certificate. This is a quite unexpected result, as the envelope is significantly longer than a CAM or DENM (see also Tables IV, V, and VI).

Comparison of the decoding results from Figure 4 with encoding results from Figure 1 shows that for all data representation mechanisms decoding is faster than encoding. This is clearly a beneficial property for VANETs, as the number of received (i.e., decoded) messages will typically greatly outnumber the number of sent (i.e., encoded) messages.

Figure 5 gives the runtime measurement results for the Intel Atom platform.

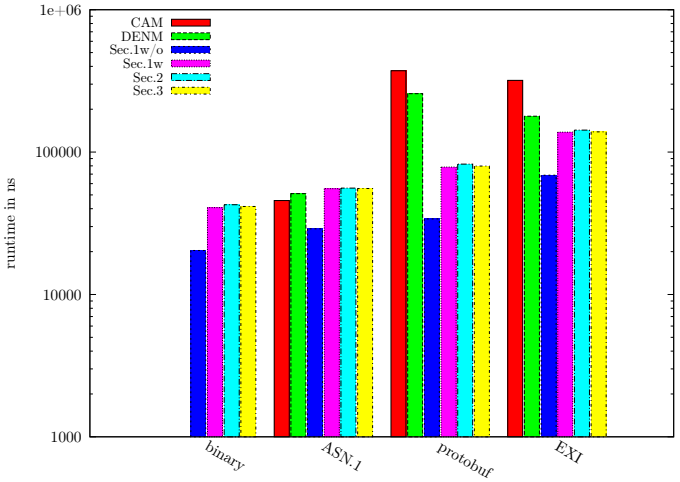

Figure 5. Decoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel Atom processor.

While the overall amount of required runtime increases in comparison to the results for the Core i7 platform, no

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

significant change in the relationship between the different deserialization methods can be obtained.

Finally, Figure 6 provides runtime measurement results for the AMD Geode platform.

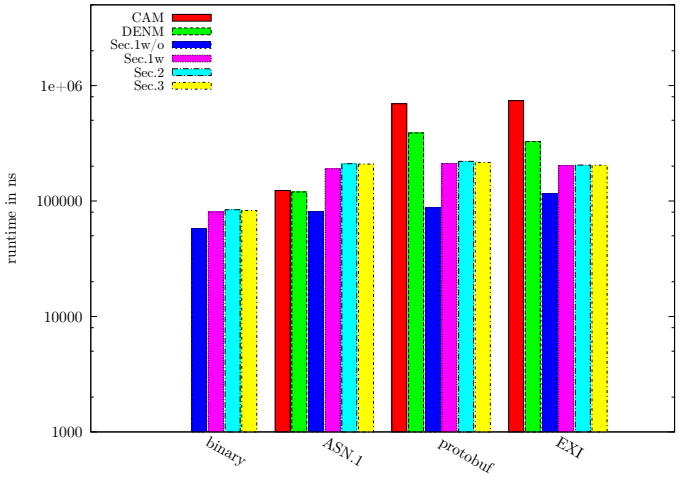

Figure 6. Decoding runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an AMD Geode processor.

As before, it can be seen that the overall processing time scales again when compared to the Atom and Core i7 platforms, but in general the relation between deserialization mechanisms is the same as with the other platforms. Thus, it can be concluded that the obtained results for the relation between processing speeds of the different data representation schemes is the same on all regarded platforms.

In summary, the obtained performance measurement results for data deserialization are in line with the results of data serialization provided in Section V-C1.

An overall conclusion about the achieved results in this work is to be found in the following Section VI.

VI. Conclusion and Future Work

Efficient data encoding schemes are required for future bandwidth-limited C2X communication. In this work, we have addressed three main performance metrics in C2X communications, which are encoded data length, runtime and memory footprint. A study on these metrics for the ASN.1, Google Protocol Buffers (protobuf) and the Efficient XML Interchange format (EXI) encoding schemes has been performed using ETSI ITS CAM and DENM messages as well as for their security envelopes. On the latter, we have further evaluated these metrics for the case of binary encoding, too. To make the study as independent on the hardware as possible, the evaluation was done using three different processor technologies. Our work also presents the used methodology for obtaining the mentioned performance metrics.

The results presented show that the outlined measurement methodology is able to provide the required performance characteristics in a reliable way. Additionally, it was found that the performance of the different encoding technologies is independent of the used processor technology.

From the presented results, it is clear that the performance of protobuf and EXI schemes are almost always outperformed by ASN.1 encoding w.r.t. the required encoding delay or runtime. Only in a minor amount of the studied cases, protobuf outperformed ASN.1 encoding with regard to its memory footprint. EXI showed to be the most expensive scheme in terms of memory footprint. In terms of encoding length for the cases of CAM and DENM ASN.1 UPER encoding performs better compared to EXI and protobuf. For these cases, the differences in length between the serialized information generated by EXI schemes and ASN.1 were considerable but small. In contrast, protobuf serialized message lengths are so large that this scheme cannot be used in C2X communication.

An important result of the conducted performance study is that binary encoding greatly outperforms ASN.1 encoding in the clear majority of cases for the security envelope. ASN.1 actually outperformed its binary counterpart with respect to encoded data length only in one of the studied cases (security envelope for CAMs without certificate). Regarding runtime, binary encoding performs significantly better in all studied cases. The latter implies that the recent shift from binary towards ASN.1 encoding (from [14] to [15]) is not justified at least by the mentioned performance metrics.

In case a more compact representation of the security envelope is required than binary encoding can provide, one should consider to move to EXI data representation instead of its ASN.1 counterpart. The EXI variant always provides a smaller serialization size in average and current performance burdens are likely to be overcome with an implementation being more targeted to the specific ETSI ITS use case. Therefore, the authors propose to conduct additional studies involving either extensive simulations or field tests using both technologies before finalizing the corresponding standard in order to determine which encoding scheme should be used for mass rollout of the future ETSI ITS system.

Directions on future work may include an extension of the provided performance study regarding new upcoming platform independent encoding schemes like Apache Avro [17]. Such systems may provide more flexibility regarding how to organize the encoded data. However, future research has to show whether these improvements have to be paid for by a performance degradation limiting practical usability. Additionally, more runtime efficient implementations of the used Efficient XML Interchange format can be studied to enhance practical usability of this data representation scheme.

References

[1] S. Bittl, A. A. Gonzalez, and W. Heidrich, “Performance Comparision of Encoding Schemes for ETSI ITS C2X Communication Systems,” in Third International Conference on Advances in Vehicular Systems, Technologies and Applications, June 2014, pp. 58–63.

[2] “Memorandum of Understanding for OEMs within the CAR 2 CAR Communication Consortium on Deployment Strategy for cooperative ITS in Europe,” June 2011, v 4.0102.

[3] J. Harding, G. R. Powell, R. F. Yoon, J., C. Doyle, D. Sade, M. Lukuc, J. Simons, and J. Wang, “Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application,” Washington, DC: National Highway Traffic Safety Administration, Tech. Rep. DOT HS 812 014, Aug. 2014.

[4] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Std., Rev. 5th, Nov. 2008.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

[5] D. Crockford, "The application/json Media Type for JavaScript Object Notation (JSON)," Network Working Group, IETF, RFC 4627, July 2006.

[6] Intelligent Transport Systems (ITS); European profile standard for the physical and medium access control layer of Intelligent Transport Systems operating in the 5 GHz frequency band, ETSI European Standard 202 663, Rev. V1.1.0.

[7] Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, ETSI European Standard 302 637-2, Rev. V1.3.0, Aug. 2013.

[8] Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service, ETSI European Standard 302 637-3, Rev. V1.2.0, Aug. 2013.

[9] OSS Nokalva, Inc, "ASN.1 Tools for C Overview," online: http://www.oss.com/asn1/products/asn1-c/asn1-c.html, Jan. 2014, retrieved: 03.2015.

[10] Google, "Protocol Buffers - Google Developers," online https://developers.google.com/protocol-buffers/, Apr. 2012, retrieved: 03.2015.

[11] ——, "Protocol Buffers. Googles Data Interchange Format." online http://code.google.com/p/protobuf/, Jan. 2014, retrieved: 03.2015.

[12] Efficient XML Interchange (EXI) Format, W3C Std. 1.0, Rev. 2nd, Feb. 2014.

[13] Road vehicles – Vehicle-to-Grid Communication Interface – Part 2: Network and application protocol requirements, ISO Std. ISO 15 118-2:2014, Mar. 2014.

[14] Intelligent Transport Systems (ITS); Security; Security header and certificate formats, ETSI Technical Specification 103 097, Rev. V1.1.1.

[15] Intelligent Transport Systems (ITS); Security; Security header and certificate formats, ETSI Technical Specification 103 097, Rev. V2.1.1.

[16] Google, "Encoding - Protocol Buffers - Google Developers," online https://developers.google.com/protocol-buffers/docs/encoding, Sep. 2014, retrieved: 03.2015.

[17] J. Russell and R. Cohn, Apache Avro. Book on Demand, 2012.

[18] L. M. Surhone, M. T. Tennoe, and S. F. Henssonow, Apache Thrift. VDM Publishing, 2010.

[19] S. Furuhashi, "MessagePack: It's like JSON, but fast and small," online: http://msgpack.org/, Jan. 2014, retrieved: 03.2015.

[20] D. Gupta, "Thrift vs. Protocol Buffers," online: http://old.floatingsun.net/articles/thrift-vs-protocol-buffers/, May 2011, retrieved: 05.2014.

[21] OSS Nokalva, Inc, "Alternative Binary Representations of the XML Information Set based on ASN.1," online: www.w3.org/2003/08/binary-interchange-workshop/32-OSS-Nokalva-Position-Paper-updated.pdf, Aug. 2013, retrieved: 03.2015.

[22] M. C. Smith, "Comparing the Performance of Abstract Syntax Notation One (ASN.1) vs eXtensible Markup Language (XML)," in Proceedings of the Terena Networking Conference, 2003.

[23] "Using Internet data in Android Applications," online: http://www.ibm.com/developerworks/xml/library/x-dataAndroid/x-dataAndroid-pdf.pdf, June 2010, accessed: February 27th, 2014.

[24] "The gzip homepage," online: http://www.gzip.org, July 2003, accessed: February 27th, 2014.

[25] B. Gil and P. Trezentos, "Impacts of data interchange formats on energy consumption and performance in smartphones," in Proceedings of the 2011 Workshop on Open Source and Design of Communication, 2011, pp. 1–6.

[26] C. Bournez, "Efficient XML Interchange Evaluation," W3C, Tech. Rep., Apr. 2009.

[27] D. Peintner, H. Kosch, and J. Heuer, "Efficient XML Interchange for Rich Internet Applications," in IEEE International Conference on Multimedia and Expo, June 2009, pp. 149–152.

[28] 2nd Generation Intel Core Processor Family, Datasheet, Vol.1, 8th ed., Intel, June 2013, doc. No. 324641-008.

[29] Intel Atom Processor Z5XX Series, Datasheet, 3rd ed., Intel, June 2010, doc. No. 319535-003US.

[30] AMD Geode LX Processor Family, AMD, Feb. 2014, doc. No. 33358E.

[31] ISO, "ISO/IEC 9945:2008 Information technology – Portable Operating System Interface (POSIX®)," May 2009, International Organization for Standardization, Geneva, Switzerland.

[32] "Debian – The Universal Operating System," online: http://www.debian.org/, Dez. 2013, retrieved: 05.2014.

[33] F. Bellard, "FFASN1 Compiler," online: http://bellard.org/ffasn1/, Sept. 2012, accessed: 31.01.2015.

[34] R. Kyusakov, "Embeddable EXI Processor in C," http://exip.sourceforge.net/, Nov. 2014, retrieved: 03.2015.

[35] Fujitsu Laboratories of America, "OpenEXI - EXI implementations in Java and C#," https://sourceforge.net/projects/openexi/, Jan. 2015, retrieved: 03.2015.

[36] Fraunhofer ESK, "ezCar2X: Streamlining application development for networked vehicles," online: http://www.esk.fraunhofer.de/en/projects/ezCar2X.html, Feb. 2014, retrieved: 03.2015.

[37] R. M. Stallman and the GCC Developer Community, Using the GNU Compiler Collection, For GCC version 4.8.2, Free Software Foundation, Oct. 2013.

[38] M. T. Jones, "Kernel APIs, Part 3: Timers and lists in the 2.6 kernel," online: http://www.ibm.com/developerworks/library/l-timers-list/, Mar. 2010.

[39] G. Paoloni, "How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures," Intel, White Paper 324264-001, Sept. 2010.

[40] T. Bingmann, "malloc_count - Tools for Runtime Memory Usage Analysis and Profiling," online: http://panthema.net/2013/malloc_count/, Mar. 2013, retrieved: 05.2014.

[41] J. Seward, N. Nethercote, J. Weidendorfer, and V. D. Team, Valgrind 3.3, 1st ed. Network Theory Ltd., May 2008.

2015, © Copyright by authors, Published under agreement with IARIA - www.iaria.org