Performance Comparision of Encoding Schemes for ETSI ITS C2X Communication Systems

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

***Abstract*—Wireless Car-to-X communication is about to enter the mass market in upcoming years. Thereby, available bandwidth is small with only a low number of usable channels and many communicating entities. Therefore, efficient data encoding schemes are required to allow bandwidth saving use of the wireless channel by embedded devices. No detailed analysis of different encoding schemes for Car-to-X communication regarding important properties like runtime, memory consumption and encoded output length has been published so far. We provide such analysis for standardized ASN.1 and binary representations as well as Google Protocol Buffers as an alternative approach to the data encoding problem. Standardised data content for CAM, DENM and the security envelope are used in the conducted performance study. We show that ASN.1 encoding outperforms usage of Google Protocol Buffers, but is outperformed by a binary encoding scheme in most cases. This implies that standardization efforts for the security envelope should reconsider the recent shift from binary encoding towards usage of ASN.1.**

Keywords-ETSI ITS, data encoding, performance metrics, ASN.1, Google Protocol Buffers.

I. INTRODUCTION

Car-to-X (C2X) communication systems are gaining attention in the awake of their upcoming deployment as ETSI Intelligent Transport Systems (ITS) in Europe and Wireless Access in Vehicular Environments (WAVE) in the United States [1].

C2X communication happens digitally, meaning that messages between the involved nodes are represented as a series of bits, i.e., as bit streams. 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 message. Following the same logic, decoding is defined as the generation of the original message out of 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[2], JSON[3]), through 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 [4]. Therefore, an encoding that generates short bit streams out of messages is favoured. Moreover, safety C2X applications have strict end-to-end delay requirements. Therefore, encoding/decoding delays should be minimal such that their contribution to the end-to-end delay can be considered neglible.

In this work, we focus on the comparison of the performance metrics of two coding schemes, namely Abstract Syntax Notation 1 (ASN.1) encoding rules and Google Protocol Buffers applied to the two most common C2X message types in C2X communications: Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Basic Service (DENM). Since both encoding schemes support the Time Optimized and Space Optimized variants, performance metrics are obtained for both cases on both schemes. Furthermore, three different encoding schemes for the ETSI ITS Security Envelope, which are binary encoding, ASN.1 encoding rules and Google Protocol Buffers performance metrics, are also compared.

The remaining part of this work is organized as follows; an overview of related work is given in section II, the performance requirements and measurements are described in detail in section III and the target platforms description is summarized in section IV. The obtained results 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 comparision to the limited number of other published performance studies is given.

A. Data Encoding Rules

CAM and DENM are two standardized ITS messages defined in [5] and [6] respectively. According to these standards,

 ISBN: 978-1-61208-348-3

the encoding of CAM and DENM is done using ASN.1 UPER encoding rules. There are several encoding rules which ASN.1 specifies: Basic Encoding Rules (BER), Packed Encoding Rules (PER), Canonical Encoding Rules (CER), Distinguished Encoding Rules (DER), Octet Encoding Rules (OER) among many other flavours, each providing advantages and disavantages 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 [7]. 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 [8], [9], which are used by Google extensively in their production environment. Therefore, they can be regarded as a stable and reliable library. Google Protocol Buffers offer a more simplistic approach to the platform independent data encoding, making them easier to manipulate and implement [8]. 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 [7] provides this feature. Therefore, Google Protocol Buffers can be seen as a comparable alternative for message implementation. Hence, the performance study provided in section V makes use of these two technologies.

Furthermore, 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 [10]. Additionally, encoding using ASN.1 rules was proposed recently in [11]. As a further reference, Google Protocol Buffers will also be used in the following for the security envelope.

Further 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 [12][13][14]. These systems are either less mature or deployed to a much smaller extent in professional environments compared to ASN.1 and Google Protocol Buffers (see e.g., [15] for protobuf vs. Thrift). Therefore, they are not studied in detail. 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. They are simply 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 [16] compare the performance between binary encoded XML and ASN.1 by running the tests on PC machines. In [17] 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 [18] authors compare the performance of XML, JSON and Google Protocol Buffers in terms of data size and coding speed. The authors conclude that Google Protocol Buffers 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 [19]. In the latter case, both compressed text formats perform better than Google Protocol Buffers in terms of data size. In terms of speed, the authors show that Google Protocol Buffers perform better than both text schemes. In [20], authors perform a similar study showed in [18] and expand it for performance in energy consumption, relevant for the smartphones case. They also show that gzip-compressed Google Protocol Buffers, variant not explored in [18], performs better in terms of encoded data size in comparison with compressed XML, but worst that compressed JSON. When the authors measure performance in respect to encoding time, they concluded that for the data set they used Google Protocol Buffers performed better. On the parsing process on the receiver side, i.e., decoding, JSON perform slightly better than the other two.

To the understanding of the authors at the time of writing this work, there are no previous studies focusing on a quantitatively comparison of performance measurements between ASN.1 and Google Protocol Buffers, in specific on the field of C2X commmunications. Although the ETSI standard defines the encoding mechanisms as ASN.1, this work should provide some insight for the viability of an alternative based on an open source development as well as to provide some information on the performance comparison of these encoding schemes on different computer platforms such as embedded systems.

III. Performance Requirements and Measurements

In this work, we consider three main aspects in the performance evaluation of the different encoding schemes. These 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 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 lenght 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.

 ISBN: 978-1-61208-348-3

IV. TARGET PLATFORMS

A. Hardware

To execute our performance measurements of the encoding 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 I summarizes the main characteristics of the three platforms used in our experiments.

TABLE I. 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 [21], [22], [23].

The clock resolution given in table I was obtained by using the clock_getres() [24] function on the individual platforms in the software environment described in the next section.

B. Software

On all platforms, a standard Debian Linux [25] system with kernel version 3.2.23 was used as the underlying operating system during the performance study. Furthermore, ASN.1 related functionality was provided by the OSS Nokalva library [7]. Google protocol buffers were used in version 2.4.1 as provided by the Debian distribution. For binary encoding of the security envelope the implementation from the ezCar2X framework [26] was used. All used software was compiled on the target with the GCC compiler version 4.7.2 [27]. Thereby, strong optimization was enabled with the *-O3* compiler flag.

For timing measurements the Linux kernel high performance counters have been used, which can be used from userspace by calling the clock_gettime() function [24]. 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 [28]. In order to make the measurements more accurate, the suggestions from [29] 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 CPUs time stamp counter (TSC), which is e.g., used in [29]. The reason is that while [29] 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 [29] like disabling of interrupts or scheduling cannot be fullfilled. 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 [30]. This framework allows arbitary 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 [31] 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.

V. PERFORMANCE STUDY

A. Content for Encoding and Decoding

We have used CAM [5] and DENM [6] messages containing only values in the mandatory fields. For this messages, we have used real data within the message content as far as possible e.g., the included time stamps.

The studied security envelopes consist of the message fields as specified in [10] and [11]. Thereby, all three defined security profiles are taken into regard. Additionally, for 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.

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 [10].

B. Encoding Rules for Google Protocol Buffers

The definition files for the Google Protocol Buffers (Protobuf) were derived from the ASN.1 definitions given in standards [5], [6], [11]. Thereby, transformation is straight forward due to the low number of available data types in protobuf. During the transformation process always the smallest Protobuf data type which is able to hold the corresponding ASN.1 data type was selected to avoid unneccessary overhead.

C. Results of Performance Study

The results of the conducted performance study regarding memory consumption and encoded output length are summarized below in tables II (CAM), III (DENM) and IV (security envelope). In the following, individual results for these message contents are studied 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 criterias in the following. Runtime performance, which is clearly processor specific is looked at afterwards.

In the following, we use TOED as a short for time optimized encoder and SOED for space optimized encoder. All encoding length and memory consumption measurement results are given in bytes.

At first, encoding performance for CAMs is studied in detail. The achived results are summarized in table II. From table II it is clear that Protobuf generates almost four times more output bytes than ASN.1 for an encoded CAM. The space optimized code is roughly on par with the ASN.1 code, as Protobuf uses less heap but more stack space and

 ISBN: 978-1-61208-348-3

TABLE II. PERFORMANCE RESULTS FOR CAMS.

enc. type	heap / stack TOED	heap / stack SOED	encoded length
protobuf	469 / 1564	2450 / 6580	165
ASN.1	1066 / 1300	4120 / 4600	42

ASN.1 uses roughly the same amount of heap and stack space. Additionally, the time optimized Protobuf code uses less space as the space optimized code and even less than the ASN.1 time optimized code.

In the following, we study the encoding performance of DENMs. The corresponding results are given in table III. As

TABLE III. PERFORMANCE RESULTS FOR DENMS.

enc. type	heap / stack TOED	heap / stack SOED	encoded length
protobuf	306 / 1532	2181 / 6580	114
ASN.1	1067 / 1252	4163 / 4184	43

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 time optimized 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.

Table IV gives the performance results for main memory consumption as well as encoding length for the ETSI ITS security envelope. In table IV the profile column gives the

TABLE IV. PERFORMANCE RESULTS FOR THE SECURITY ENVELOPE.

enc. type	profile	heap/stack TOED	heap/stack SOED	enc. length
binary	1 no cert.	220 / 5348	same as TOED	96
	1 cert.	412 / 6420	same as TOED	178
	2	412 / 6420	same as TOED	189
	3	412 / 6420	same as TOED	186
protobuf	1 no cert.	1282 / 6452	1286 / 6452	127
	1 cert.	2065 / 9892	2065 / 9892	231
	2	2244 / 9892	2244 / 9892	243
	3	2094 / 9892	2094 / 9892	237
ASN.1	1 no cert.	1927 / 6500	5123 / 6504	87
	1 cert.	2309 / 6676	5505 / 8296	197
	2	2309 / 6676	5515 / 8296	207
	3	2309 / 6676	5515 / 8296	207

number of the applied security profile as defined in [10]. 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 1.

The encoding lengths for security profiles 2 and 3 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 [10] but required according to the ASN.1 definition given in [11]. 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, no separate data for computation time and memory consumption is given for security profile number 3 and ASN.1 encoding, as it would be identical to the case of security profile number two. In order for a difference between the two security profiles to exist, our Protobuf definition declares the message type field as being optional.

One can see from the most right column 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 is 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.

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 I are shown in Figures 1, 2 and 3. Please note that the vertical axis of the graph is on 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 a Intel Core i7 processor is provided in Figure 1 (see also Table I). The obtained results clearly show that, for

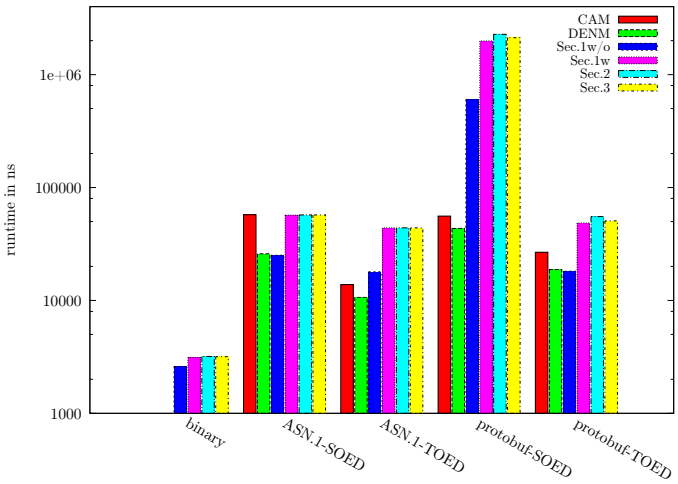

Figure 1. Runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel i7 processor.

the security envelope, binary encoding is significantly faster than the two other encoding schemes. Additionally, ASN.1 encoding outperforms Protobuf for both cases of TOED w.r.t. SOED optimization.

An interesting result is that the difference regarding the runtime of TOED and SOED versions is significantly different for ASN.1 and Protobuf. Thereby, the results for ASN.1 encoding differ far less than the corresponding results for Protobuf do. Furthermore, the difference in runtime between TOED and SOED versions for Protobuf is much bigger for encoding of the security envelope then it is for CAM and DENM encoding. The documentation of Protobuf mentions that space optimized encoding relys on reflection instead of using dedicated data access methods [8]. From the achieved results one can expect that the optimization strategy of the used

 ISBN: 978-1-61208-348-3

ASN.1 library works differently, unfortunately, no in detail description regarding this point is available (see also [32]).

A significant difference between the definitions of CAMs (or DENMs) and the security envelope is the higher number of small and deeply nested data fields used for defining the security envelope ([10][11][5][6]). The achieved results depicted in Figure 1 indicate that binary as well as ASN.1 SOED encoding can handle this kind of structure better than Protobuf SOED can do. Thereby, the reflection based access scheme is likely the source of excess in runtime increase when comparing Protobuf SOED with the TOED variant.

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 and no 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.

The results obtained from the runtime measurements on a Intel Atom processor are depicted in Figure 2 (see also Table I). Comparing Figure 2 to preceeding Figure 1 one can see

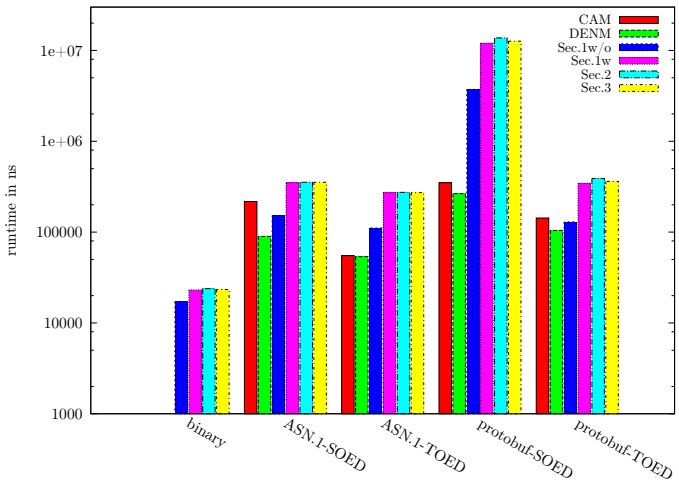

Figure 2. Runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an Intel Atom processor.

that except of a general increase in runtime (note the different scaling of the vertical axis of both figures), the overall results are the same for the Atom and the i7 processor technology. Due to the lower processor speed (see also Table I) 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 i7, which is due to the improved processor technology such as precaching algorithms, as it was introduced to the market significantly later than the Atom processor.

Finally, Figure 3 provides the results of runtime measurements conducted using an AMD Geode processor (see also Table I). From the comparison of results shown in Figure 3 to the results given in Figures 1 and 2, one can see that the

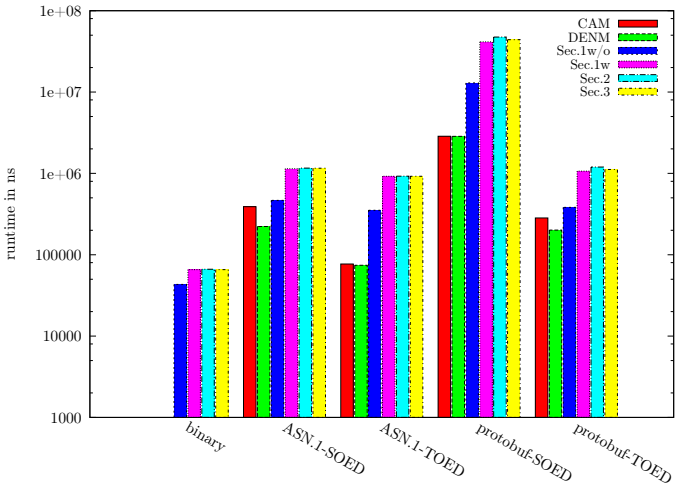

Figure 3. Runtime performance of ETSI ITS CAM, DENM and security envelope encoding on an AMD Geode processor.

overall outcome of the performance study does not change by switching from a modern high speed processor (like the 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 behaviour 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. Only in the case of security profile number 1 without certificate, ASN.1 achieves a shorter encoding length than binary encoding. Is worth to note that, the timing interval for including a certificate in the security envelope of a CAM is equal to the default sending interval of CAMs (see [10][5]). The latter means that normally CAMs are sent with a certificate included in the envelope. Therefore, the results show that the newer standard [11] defining the security envelope using ASN.1 significantly deteriorates the performance of its encoding compared to the preceeding standard [10] using a binary encoding scheme. Furthermore, as ASN.1 does not provide a forward compatability functionality, like e.g., Protobuf would do, there is almost no reason why one should prefer ASN.1 over binary encoding. 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.

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: encoded data length, runtime and memory footprint.

 ISBN: 978-1-61208-348-3

A study on these metrics for the ASN.1 and Google Protocol Buffers encoding schemes, for the time and space optimized variants, has been performed on the ETSI CAM and DENM messages as well as their Security Envelope. On the latter, we have further evaluated these metrics also for the case of binary encoding. 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 followed methodology for obtaining the mentioned performance metrics.

The results presented here 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 Google Protocol Buffers (Protobuf) is 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.

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. Regarding runtime and memory footprint: binary encoding performs significantly better in all studied cases. The latter implies that the recent shift from binary towards ASN.1 encoding (from [10] to [11]) is not justified at least by the mentioned performance metrics. Therefore, the authors propose to conduct 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 [12]. Such systems may provide more flexiblity 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 usablity.

REFERENCES

[1] "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.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] OSS Nokalva, Inc, "ASN.1 Tools for C Overview," online: http://www.oss.com/asn1/products/asn1-c/asn1-c.html, Jan. 2014, retrieved: 05.2014.

[8] Google, "Protocol Buffers - Google Developers," online https://developers.google.com/protocol-buffers/, Apr. 2012, retrieved: 05.2014.

[9] ——, "Protocol Buffers. Googles Data Interchange Format." online http://code.google.com/p/protobuf/, Jan. 2014, retrieved: 05.2014.

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

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

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

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

[14] S. Furuhashi, "MessagePack: It's like JSON, but fast and small," online: http://msgpack.org/, Jan. 2014, retrieved: 05.2014.

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

[16] 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: 05.2014.

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

[18] "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.

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

[20] 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.

[21] 2nd Generation Intel Core Processor Family, Datasheet, Vol.1, 8th ed., Intel, June 2013, doc. No. 324641-008.

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

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

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

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

[26] Fraunhofer ESK, "ezCar2X: Streamlining application development for networked vehicles," online: http://www.esk.fraunhofer.de/en/projects/ezCar2X.html, Feb. 2014, retrieved: 05.2014.

[27] R. M. Stallman and the GCC Developer Community, Using the GNU Compiler Collection, For GCC version 4.7.2, Free Software Foundation, Sept. 2012.

[28] 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.

[29] 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.

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

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

[32] OSS Nokalva, Inc, "What do I gain by using the time-optimized encoder/decoder (TOED)? What do I lose?" online: http://www.oss.com/asn1/knowledge-center/asn1-c/91.html, Feb. 2014.

 ISBN: 978-1-61208-348-3