Performance evaluation of using Protocol Buffers in the Internet of Things communication

Protobuf vs. JSON/BSON comparison with a focus on transportation's IoT

Srđan Popić,
University of Novi Sad
RT-RK Institute for Computer Based Systems
Novi Sad, Serbia
Srdjan.popic@rt-rk.com

Bojan Mrazovac
RT-RK Institute for Computer Based Systems
Novi Sad, Serbia
bojan.mrazovac @rt-rk.com

Dražen Pezer
RT-RK Institute for Computer Based Systems
Novi Sad, Serbia
drazen.pezer@rt-rk.com

Nikola Teslić
University of Novi Sad
Novi Sad, Serbia
nikola.teslic@rt-rk.uns.ac.rs

***Abstract*—Things connected to the internet may not be connected only to servers on the cloud. Its usefulness can be unleashed only if they are interconnected as well. This interconnection is recognized as communication between services of the Internet of Things. Due to the nature of the things on the web, spatial overhead for any data exchanged needs to be kept to a minimum. JSON is recognized as a most efficient way to transfer various data in domain of distributed embedded systems. JSON's binary representation, BSON is even more preferable. This paper explores the possibilities and critically examines and evaluates the effectiveness of using Google's Protocol Buffer as a processing protocol and communication standard in transportation domain of the Internet of Things.**

Keywords—Internet of Things; Google Protocol Buffers; Performance; Encoding; Intelligent Transportation

I. INTRODUCTION

Increasing the number of devices connected to a global network, the scope of embedded systems expands. A simple but feasible format for the exchange of data is required [1]. These devices are not only used to provide data but also services. This means that web services are already merging with embedded devices.

More precisely, in transportation, compute nodes are rising faster than any other industry. In modern cars stunning 30% of the total cost is electronic components [2]. This trend will be further amplified when the cars become able to communicate and to autonomously gather ambient information. For instance, when there is a queue, cars at the front may inform the cars behind if there is an accident or just the traffic overload, and this will eventually make intelligent navigation systems re-plan the route of cars programmed to go down already saturated roads. The cars may help the driver to keep safe distance from the car in front, and may refuse dangerous actions like speeding if the weather conditions are unsafe or overtaking if the oncoming car goes too fast. The cars can be driven by autopilot on highways reducing the risk of fatigue related accidents [2].

The vehicles will also be able to maintain themselves, calling for the appropriate service based on the self-diagnosis of the problem and ensuring that the right replacement parts are in stock. The car will plan the time of service according to the diaries and preferences of the usual driver to minimize the petulance of their lives, and make sure that there is a substitute car available if there would be a need for it [2]. Moreover, some auto industry leaders together with Google established the Open Auto Alliance (OAA). There is a plan to bring new features to the Android platform to accelerate the adoption of the Internet of Vehicles (IoV) paradigm [3]. Communication channel is "a two-way street", since the traffic and weather related information will be updated on the vehicle [4].

Taking all of this into consideration, the Internet of Vehicles has a huge communication field between compute nodes in transportation and it must have its lightweight way of serializing structured data. This 'lightweight way' must be useful in heterogeneous devices, easily parsed/read and, without any unnecessary overhead in communication. This communication is not only enabled between the perception and the network layer, but also between "things" in the perception layer itself.

Every data exchanged in this communication field must fulfill "the right information condition". The right information condition is met if it can be utilized with a minimum effort. This includes human readable information for human interaction as well as semantically and syntactically enriched

This work was partially supported by the Ministry of Education, Science and Technological Development of the Republic of Serbia, under grant number: TR32041

978-1-5090-3720-9/16/$31.00 ©2016 IEEE

machine-readable information, which may in turn require transformation of low-level raw data (possibly from multiple sources) into meaningful information and may even require some pattern recognition and further analysis to identify correlations and trends in the generated data [5].

In [1] JSON is offered as a way to solve communication issues. JavaScript Object Notation (JSON) is a human-readable data-interchange format. It is based on a subset of JavaScript programming language with the purpose to be parsed correctly by JavaScript compiler [6]. There are other formats that can be candidates for a protocol, such as XML [7, 8], more precisely Efficient XML Interchange (EXI) [9]. EXI converts XML messages to binary ones to reduce the needed bandwidth and minimize the required storage size. In [10] it evaluates the performance of EXI in constrained devices.

Most of the modern controllers have been equipped with script interpreting engines; therefore, scripting languages are used to easily define and apply desired behaviour upon the ecosystem of available devices [11]. Beside XML, Lua and SMIL scripting languages are used as well.

Still, it is JSON [1] that is seen more as a lightweight data carrier, which has characteristics of small space occupancy and fast transmission speed [12].

Recently, Protocol Buffers [13] have been introduced as a powerful way of encoding structured data in an efficient and extensible format. Since it is easily transformed into XML and JSON format, data in Protobuf are both machine and human readable. This fulfills the "the right information condition".

In this paper, Protobuf is evaluated as a possible alternative for JSON messages in Internet of Vehicle, on proprietary vehicle tracking device. Since Protobuf is in binary format, comparison is done with BSON as well. BSON stands for Binary JSON. It is a binary encoded serialization of JSON-like documents. Like JSON, BSON supports the embedding of documents and arrays within other documents and arrays [14].

As the survey [15] shows, there are a few standardized areas in the Internet of Things. There are only more or less relevant protocols and applications. Due to that, Protocols Buffer might be the right candidate for standardized communication protocol.

II. Protocol Buffers

Initially, Protocol Buffers (*Protobuf*) have been developed by Google to solve the issue on a large number of requests and responses to the index server.

Prior to protocol buffers, there was a format for requests and responses that used hand marshalling/unmarshalling of requests and responses, and that supported a number of protocol versions. This resulted in a rather inconvenient code [13]. Since it is exploited by the Google, it is assumed that it is stable and well tested. Likewise, it is language and platform independent. It supports Java, C++, Python, as well as other programming language (through third party implementations).

Protobuf allows developers to build up the proto file describing the structure of the messages that will be sent. Proto files are compiled into native language code providing wrapper classes for reading and writing message content. Language can treat multiple versions of the message frame (proto file) by treating new fields as optional. Real in-depth knownledge is not mandatory in Protocol Buffer, which makes it very popular.

```
message Person {
    required string Name = 1;

    enum SexType {
        FEMALE = 0;
        MALE = 1;
    }

    enum TitleType {
        MS = 0;
        MRS = 1;
        MISS = 2;
        DR = 3;
        MR = 4;
    }
    enum EmployType {
        N = 0;
        Y = 1;
    }
    required SexType Sex = 2 [default = FEMALE];
    required TitleType Title = 3 [default = MS];
    required int32 Age = 4;
    required int32 Income = 5;
    required EmployType Employed = 6 [default = N];
}
```

Fig. 1. Simple PB definition for Person

Figure 1 [16] shows a message definition for a simple Person message. Every Person message has a Name, Sex (which can be Female or Male), Title (which can be Mr, Mrs, Ms, Miss, Dr), Age, Income and Employed. It is possible to define the field order and types using C-like definitions and also assign default values for each field. For Title, Sex and Employed ENUMs are used in the message definition allowing a preliminary restriction on the field values.

Figure 2 [16] shows more complex message types, such as one-to-many relationship between Person message and *HouseHold* message, using *repeated* keyword.

```
message HouseHold {
    required int32 MemberCount = 1 [default = 0];
    required int32 TotalIncome = 2 [default = 0];
    repeated Person Member = 3;
}
```

Fig. 2. Nested PB message to serialize HouseHold messages

III. Proprietary Tracking System

Proprietary vehicle tracking system protocol is protected by the copyright laws and cannot be exposed in this paper. The names of the messages and the details on the protocol are disclosed. However, the types of the messages, are similar to any other tracking system:

- Requests – queries data from the "thing"
- Commands – updates/inserts the data to the device
- Responses and status messages – returns the requested data from the device
- Acknowledge, negative acknowledge and keep alive signals

In regular working mode, responses and status messages make 90% of the total traffic.

```
request:
  0000  53 56 00 0e 18 00 00 30 39 0d 81 00 00 02

response:
  0000  43 d3 00 6a 17 00 00 c0 f7 04 22 02 5c 00 1b ff
  0010  ff ff ff ff 01 56 df 30 53 0f 69 f1 80 0e e2 6c
  0020  d2 0e 56 df 30 53 00 2c 00 37 01 08 01 00 01 00
  0030  00 00 00 00 00 1c 00 10 cc 21 00 00 1c 03 5c 02
  0040  00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 ff
  0050  ff ff ff ff ff ff ff ff ff ff ff ff 82 00 00 00
  0060  bb 44 7b 00 00 00 00 00 00

ack:
  0000  43 d3 00 0d 17 00 00 c0 f7 04 06 00 00

command:
  0000  53 56 00 6c 18 00 00 30 39 03 86 04 74 69 6d 2d
  0010  62 72 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0020  00 00 00 00 00 00 00 00 00 00 00 00 74 69 6d 00
  0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0040  00 00 00 00 00 00 00 00 00 00 00 00 74 69 6d 00
  0050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
  0060  00 00 00 00 00 00 00 00 00 00 00 00
```

Fig. 3. Examples of original binary messages. Request message requesting status information from device every 10 seconds. Response message with status information. Acknowledgement message. Command message updating information about server

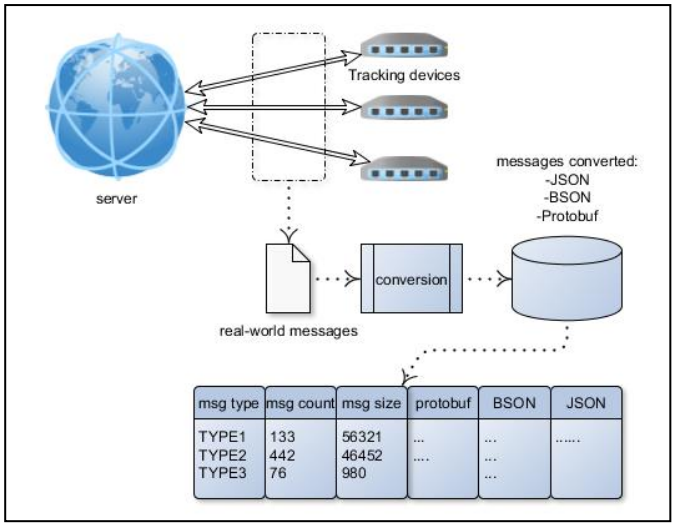

Fig. 4. Gathering the information proces.Process listens for the real-world messages, converts it in three different formats and serializes it in ad-hoc database. Later, message sizes are measured and represented in one table

Figure 3. shows examples of original messages. The difference in size between *response* and *command* on the one side and *ack* and *request* on the other, is evidently significant. Although, this is not the case with *command* messages, generaly. Notice the number of 0x00 bytes in *command* message example.

IV. Methodology

The main goal is to convert real-world messages into its JSON, BSON and Protobuf representation. System is set up to listen messages from the real tracking devices and record them in one *log* file.

There are more than 50,000 recorded messages. The process, then, converts them into three different representations without losing any information. Converted messages along with real messages are placed in *ad-hoc* database. After its sizes are measured and summarized, the result table is given. Figure 4 shows the process.

V. Results and Discussions

After gathering and converting 51690 messages, the process measures them. The Table 1 gives an overview of this process results. It shows the total number and size of the messages in all four formats: original message format, protocol buffers, BSON and JSON.

TABLE I. Total number and size of the messages grouped by message

message	*Msg count*	*Orig. msg size [B]*	*Protobuf size [B]*	*BSON size [B]*	*JSON size [B]*
CMD 1	72	1008	1517	16284	15975
CMD 2	222	9332	19034	140412	158123
CMD 3	24	384	504	4632	4654
CMD 4	136	2312	2448	22168	22304
CMD 5	2319	73254	93492	701726	713134
RESP 1	12	612	660	2712	2724
RESP 2	136	2584	3400	26520	27064
RESP 3	707	34170	114394	617267	753811
RESP 4	63	2268	3339	17451	17388
RESP 5	21895	2318737	6384328	30000511	35920037
RESP 6	1996	121756	37780	335556	342995
RESP 7	54	2916	6534	39474	43254
RESP 8	55	3080	990	9460	9515
REQ 1	36	432	720	6732	6768
REQ 2	77	924	1925	17402	18350
REQ 3	128	1408	2514	23704	23832
ALIVE	418	4626	8062	71591	72767
ACK	23309	303017	493826	4232556	4298674
NACK	31	372	651	6169	6260

Table 2. shows summary of Table 1. grouped by message type.

TABLE II. TOTAL NUMBER AND SIZE OF THE MESSAGES IN DIFFERENT FORMATS, GROUPED BY MESSAGE TYPE

message	*Msg count*	*Orig. msg size [B]*	*Protobuf size [B]*	*BSON size [B]*	*JSON size [B]*
CMD	2773	86290	116995	885222	914192
RESP	24918	2486123	6551425	31048951	37116788
REQ	241	2764	5159	47838	48950
ACK	23758	308015	502539	4310316	4377701
Total	**51690**	**2883192**	**7176118**	**36292327**	**42457631**

Figure 5. shows message types sharing in total message count. Since the communication is such that almost every message got acknowledgement in return, the number of *Acks* messages is pretty high. The share of the *Responses* is highest of all, because of the possibility to trigger many response messages based on one request. For example: after it is triggered by one *Request* message, gps status data is sent every 10 seconds as *Response* message. This is a usual procedure for tracking devices.

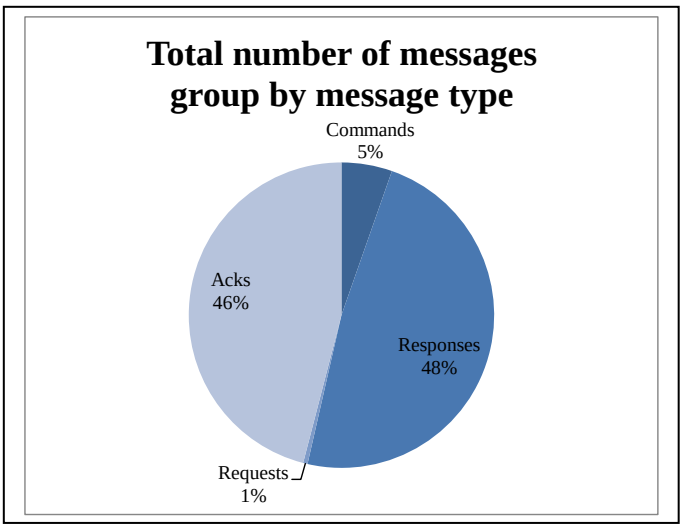

Fig. 5. Total number of messages grouped by message type

Although *Acks* messages have great share in total number of messages, that is not the case, when the message size comes into account. Figure 6. shows that 46% of *Acks* messages only share 11% of the traffic. On the other hand 48% of *Response* messages share up to 86% of total traffic. This is due to an obvious fact that the size of the *response* messages is significantly bigger in comparison of *Acks* messages. Based on the results from the Table 2. it is obvious that the current, proprietary binary protocol creates less traffic. But it is also evident that if traffic was made with Protobuf, it would be more than 5 times less than BSON traffic.

Figure 7. shows the huge traffic differences in total message size between original messages, protobuf messages and BSON/JSON messages.

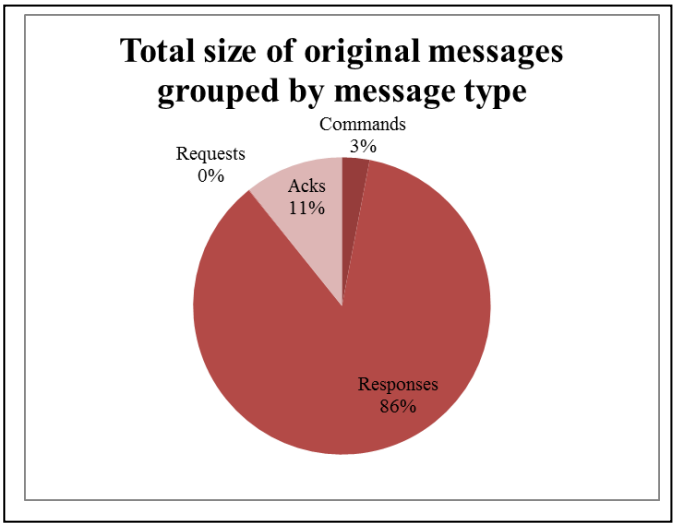

Fig. 6. Total traffic grouped by message type

It is a matter of importance, that in some cases the proprietary message size is not significantly smaller than protobuf message size, which can be seen in Table 2, CMD row with *Commands* messages. Protobuf messages are usually 2.5 times greater than original messages, but when it comes to *Commands* messages this ratio is 1.36. The reason for this is the fact that original messages are using constant-length strings when they need to write/update the device. These constant-length strings are, usually, filled with 0x00 bytes. Since this is not the case with Protobuf messages, it causes smaller size ratio. This is shown on Figure 3. When it comes to *commands* message, from address 0x000C to 0x002B lays information about server name: *tim.br*. It took 32 byte for this information. Most of the bytes in this message are plain fillers.

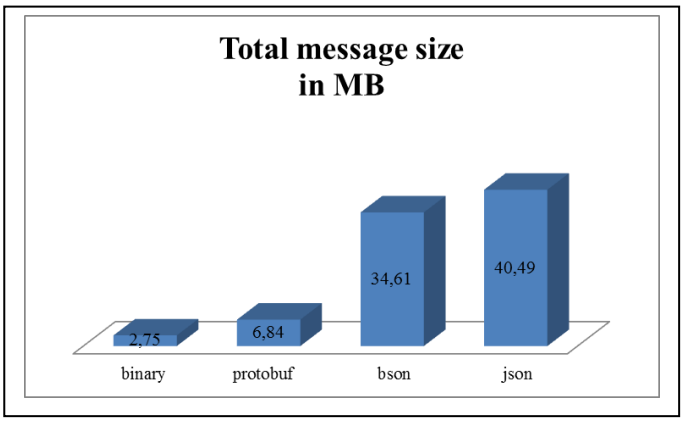

Fig. 7. Total traffic differences between message types

For *Commands* messages the size ratio between Protobuf and BSON messages is specific too. This ratio is usually approximately 5, but in the case of *Commands* messages it is almost 8. The reason for this is the use of Protobuf's enums in case of command types. All command types are represented with one enum value that takes one byte. On the other hand, JSON uses strings of characters for the representation of the command types.

VI. Conclusion

This paper shows that the Protocol Buffers are a serious candidate for standardized way of communication in field of Internet of Things, particularly because of lesser network load. It is also shown that proprietary protocols should avoid constant-length strings since it is obvious overhead. Additionally, this paper shows the power of the Protobuf's enum type.

On the other hand, it is obvious that protobuf messages are almost twice as large as the original, but in case of the interoperability, protobuf messages would be the best choice.

Further research can be conducted in many directions. This can be a starting point for the research of other vehicle tracking device's protocols. Also, other fields of IoT can be researched in order to find the best candidate protocol for standardized communication. The traffic is not the only parameter that needs to be in focus, the processing time is also important. This means that one of the further researches must take the processing time into the account as well.

References

[1] Philipp Wehner, Christina Piberger, Diana Göhringer, "Using JSON to manage Communication between Services in the Internet of Things" 9th International Symposium on ReCoSoC, 2014, pp 1-4. Montpelier, France.

[2] "Internet of Things in 2020 a roadmap for the future ", INFSO D.4 Networked Enterprise & RFID INFSO G.2 MICRO & Nanosystems in co-operation with the RFID Working Group of the EPOSS, 05 September, pp. 19-20,.

[3] Open Auto Alliance. Available: http://www.openautoalliance.net/. Accessed April 2016.

[4] Keertikumar M. J, Shubham M., R.M. Banakar "Evolution of loT in Smart Vehicles: An Overview", International Conference on Green Computing and Internet of Things (ICGCloT), 2015, pp 804-809, Delhi, India

[5] Dieter Uckelmann, Mark Harrison, Florian Michahelles "Architecting the Internet of Things", Springer-Verlag Berlin Heidelberg, 2011

[6] JavaScript Object Notation, www.json.org.

[7] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, "Extensible markup language (XML) 1.0", Fifth Edition, November, 2008.

[8] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, "XML schema part 1: Structures second edition," W3C Recommendation, Oct. 2004.

[9] T. Kamiya and J. Schneider, "Efficient XML Interchange (EXI) Format 1.0," World Wide Web Consortium Recommendation REC-Exi-20110310, 2011.

[10] Nenad Gligorić, Igor Dejanović, Srđan Krčo "Performance Evaluation of Compact Binary XML Representation for Constrained Devices",International Conference on Distributed Computing in Sensor Systems and Workshops (DCOSS), 2011, pp 1-5, Barcelona, Spain

[11] Milan Z. Bjelica, Bojan Mrazovac, Nikola Teslić "Evaluation of the Available Scripting Languages for Home Automation Networks: Real World Case Study", TELSIKS, Serbia, 2011

[12] Boci Lin, Yan Chen, Xu Chen, Yingying Yu: "Comparison between JSON and XML in Applications on AJAX", CSSS 2012, August 2012, pp 1174 – 1177, Nanjing, China

[13] Google Protocol Buffers: Google's Data Interchange Format, "Documentation and open source release," http://code.google.com/p/protobuf/ . Accessed April 2016.

[14] Binary JSON, http://bsonspec.org/. Accessed April 2016.

[15] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, Moussa Ayyash "Internet of Things: A Survey on Enabling Technologies, Protocols and Applications", IEEE Communications Surveys & Tutorials, 2015, pp 2347 - 2376

[16] Miklos Kalman, "ProtoML: A rule-based validation language for Google Protocol Buffers", The 8th International Conference for Internet Technology and Secured Transactions, 2013, pp 188 – 193, London, United Kingdom