Performance Evaluation of Compact Binary XML Representation for Constrained Devices

Nenad Gligorić, Igor Dejanović, Srđan Krčo
Ericsson
Belgrade, Serbia

Abstract—**Internet of Things (IoT) envisages an environment in which a huge number of heterogeneous devices will interact, collaborate and provide information to users or execute actuation commands based on the users' instructions. Web services are seen as one of the most promising approaches for enabling such interactions. As most of the IoT devices are limited in terms of processing power, available memory and battery lifetime, one of the most important aspects is the efficient data transfer from the devices to the server side.**

This paper addresses the performance of embedded web services through evaluation of XML compact binary representation. Evaluation is based on comparative analysis of speed, memory and battery consumption for XML and Protobuf formats. Preliminary results show that the Protobuf significantly exceeds the XML in every aspect of conducted measures.

Keywords: Internet of Things; Embedded Web Services; Protobuf; XML; Performance; Encoding

I. INTRODUCTION

Internet of Things (IoT) provides a unified presence of devices on Internet, enabling new forms of communication between people and objects as well as between the objects themselves, thus changing the perspective of the information society. Addition of numerous heterogeneous devices on the fringes of the current Internet will enable numerous new applications, ranging from smart meters and personal health monitoring to building automation and connected consumer devices.

Efficient integration of such a large number of devices with varying characteristics and capabilities is a complex task. The current Internet interconnects a huge number of devices relying on the web architecture and web services. Leveraging the knowledge and experience of these technologies, a number of activities is ongoing towards definition of embedded web services suitable for the constrained environment like IoT [1]. Two key challenges being addressed are efficient web application transfer protocol and efficient payload encoding.

REST (Representational State Transfer) style web services [2] are an important part of this new concept enabling efficient transfer of information between IoT devices and users, as well as resource discovery and manipulation functionality without additional load and storage requirements. The Internet Engineering Task Force (IETF) standardization efforts are ongoing to design CoAP (Constrained Application Protocol) [3], a new lightweight version application protocol based on the REST principles, suitable for use in the constrained environments.

Another important challenge that has to be addressed is the encoding of user payload. The overall goal is to find efficient solution for constrained devices regarding their processing, memory and battery requirements. The Extensible Markup Language (XML) has been used for a long time for payload encoding. It brought the benefits of portability, extensibility, broad support by tools and libraries, and readability by people as well as machines [4]. Nevertheless, embedded systems rarely have enough memory and processing power to run an XML parser. The verbosity of XML increases RAM usage, bandwidth requirements, and operating costs [5] and therefore is not suitable for use in IoT.

A possible alternative to XML is Efficient XML Interchange (EXI) [6] approach that provides compact XML encoding using regular grammars derived from XML Schema constraints.

Recently, Protocol Buffers [7] have been introduced as a way of encoding structured data in an efficient yet extensible format. In this paper, we evaluate the Protobuf as a possible alternative for XML encoded messages in IoT.

The paper is organized as follows: In section II, the previous work is described. The outline of Protocol Buffers with XML comparison is given in section III. Section IV provides system description, testing methodology and overall measurements results. The evaluation of conducted measurements is provided in Section V. Section VI concludes the paper.

II. PREVIOUS WORK

A number of binary formats for representing XML documents are available today [a3] ili jos bolje dati nazive tih formata sa referencama]. These emerged from the need to reduce the bandwidth and memory usage. XML is either encoded or used as a basis for creating binary encoded messages, depending on end-user (device) capabilities.

Compression and difference encoding approaches can be also used for reducing size of messages. As the run-time difference encoding consumes a lot of resources for storing message during the processing (mainly to compare skeleton message and document differences [10]) it is not suitable for embedded devices with only tens of kB RAM and ROM available. Compression space gain for small messages (which

are characteristic for IoT systems) is usually minor or non-existent and is further diminished by a significant processing overhead and additional library requirement, thus making it not suitable for application in IoT.

The Abstract Syntax Notation One (ASN.1) is a well-known binary format used in telecommunication for decades. The notation describes data structures for representing, encoding, transmitting, and decoding data. Data structures transfer syntax can be encoded using different encoding rules, providing schema notation even for representing XML in binary form - XER (XML Encoding Rules). However, the complexity of ASN.1 implementation is high thus making it difficult to use in constrained environment such as IoT.

In [12], Efficient XML Interchange (EXI), Wireless Binary XML (WBXML) and Fast Infoset (FIS) are presented together with their comparative analysis. The Fast Infoset uses ASN.1 encoding by mapping W3C XML Schema Definitions into ASN.1. Therefore, FIS results can be considered as ASN.1 results. The measurement indicates that EXI represents XML content much more efficiently than BXML and ASN.1. The ASN.1 encoding of XML reduces size of messages similar to the gzip compression, but with better processing performance. The results of the ASN.1 and BXML are on average equivalent.

Measurements conducted by the W3C [6] also show better compactness and processing efficiency of EXI over ASN.1 in the case of XML encoding. The tests are executed using Japex framework [11], a simple tool for writing Java-based micro benchmarks, encoding and decoding in stages using SAX API.

Performance benchmarks [12], [6] indicate that EXI is the leading technology in the case of XML encoding, providing the best performances for XML, fast and compact implementation and reliable efficient encodings of event streams; it is human-readable and error prone. Nevertheless, the payload XML encoding techniques are still too demanding for constrained devices.

In general, avoiding XML can result in even better performance. In this case, a binary parser interface is implemented, resulting in interaction of a client application with a processor across a parser interface. This approach can be achieved using raw binary encodings such as ASN.1, Google's Protocol Buffers (Protobuf) [7] and Apache Thrift [a3].

These binary formats apply a schema to generate code for a target language that is then used for encoding and decoding. A key difference between them are the multiple encodings that ASN.1 supports, and its debugging and implementation complexity: ASN.1 demands external (third-party) tool for reading encoded message, and it is more complex to deploy. Protobuf has some built-in features, i.e. *toString()* method that returns human-readable representation of message.

Therefore, Protobuf is easier to implement and debug and this can be the key features that can leverage this format, especially for plug'n'play deployment scenarios. In order to evaluate Protobuf's performance, it has been tested as a replacement for XML based communication used in the EcoBus system, a mobile environment monitoring system [13].

TABLE I. COMPARISION OF BINARY FORMATS

	Binary formats		
	ASN.1	***Thrift***	***Protobuf***
Licence	Open source	Open source	Open source
Language compatibility	Java, C++, C, Python...	C++, Java, Python, PHP	Java, C++, Python
Parsing speed	Fast	Medium	Fast
Memory usage	Low	Medium	Medium
Debugging complexity	High	Low	Medium
Implementation	Medium	Medium	Low
Documentation	Very good	Less than good	Very good

III. PROTOCOL BUFFERS

The Protocol Buffers (Protobuf) were initially developed at Google to address the problem of large number of requests and responses to/from the index server. This protocol uses binary encoding which makes serialized data more compact. Google exploits Protobuf for almost all internal RPC, so it is well tested and stable. The supported programming languages are Java, C++, Python and there are many third party implementations for other languages [7].

In order to serialize data, data structures have to be described in a .proto file and compiled. The compilation process produces classes for representing those structures in a selected language. Afterwards, the API can be used for writing and reading messages, in respect to the compiled classes. When a message is encoded, the keys and values pair is concatenated into a byte stream. Every Protobuf message field has a unique number and can be defined as optional or required providing a certain degree of flexibility.

A. XML vs Protobuf

The primary advantage of the Protobuf is that it encodes values before the transmission using *varints* - a method of serializing integers via one or more bytes. Therefore, the XML messages are significantly larger as they employ UTF-8 encoding. In addition, every XML field is enveloped with a tag making it verbose and redundant. The Protobuf messages are not human-readable after encoding, i.e. during the transfer, when the message is being sent over the wire.

XML parsers process data arbitrarily and perform additional tests to detect poorly formatted data or syntax errors. This can cause additional overhead for the devices executing this, particularly in the embedded systems [5]. The Protocol Buffers integrates data structure at a lower level during compilation of a .proto file, thus allowing faster serialization. The XML nested elements can be mapped into a Protobuf representation due the support for message type's nested inside other messages. The embedded web services can use both formats (XML or Protobuf) to address heterogeneity between systems.

IV. ECOBUS CASE STUDY

A. System descriptions

The EcoBus system has been developed in the FP7 SENSEI project [14]. It utilizes public transportation vehicles to continuously monitor a set of environmental parameters over a large urban area as well as to provide an insight into position of vehicles in time and space.

The IoT nodes are deployed on top of the public buses providing gas measurements (CO, CO2, NO2), weather measurement (temperature, air pressure, humidity), and location (GPS). The measurements are sent over a mobile network interface (GPRS). The nodes continuously observe these parameters as well as events and activities in the physical world (e.g. average speed). All measurements are transferred to a central database using Telit GPRS/GPS module as an end-point. These modules are using built-in interface dedicated to the communication between the Python script and the module functionality.

The end users can query the system using a web or mobile Android application to get real time gas measurements as well as locations of the public transportation vehicles. Locations of the buses together with the estimated time of a bus arrival into a selected station can be retrieved by SMS and/or USSD query.

B. Testing methodology

The main goal was to perform testing under the conditions that will closely match those of the deployed system. The tests were written in Java programming language and were executed on an Android powered mobile device. The testing involved parsing of the message presented in Figure 1 (XML version presented; Protobuf version is not in a human readable format). The tests were not performed on the devices used in the Ecobus system as there was no software support for the Protobuf implementation on the devices.

Four tests were performed:

1. Raw parsing time test
2. Network&Parsing time test
3. Network&Parsing memory test
4. Network&Parsing battery test.

In the first test, the message parsing time was measured. In order to avoid impact of the I/O operations on the measurements, the messages were buffered before parsing. In tests 2-4, messages were first fetched from a remote server over a wireless network and Internet, and then parsed and finally corresponding Java objects required by the application created. This procedure was followed in all 3 test cases, but the value measured was different. In Test 2, the average operation time has been measured. Test 3 was focused on measurement of the heap memory usage, while Test 4 was used to evaluate impact of message processing on battery consumption.

To achieve better accuracy, each test consisted of multiple repeats of the operation whose performances has been measured (for example, Test 1 consisted of multiple repeats of parsing, while Tests 2-4 consisted of multiple repeats of the message fetching and parsing).

In Tests 1-3, operations were repeated 100 times. The average times per operation were calculated at the end of the each test run in tests 1 and 2, while for the memory consumption test (Test 3), the average memory consumption per operation was calculated. For Test 4, the total battery drain for 30,000 operation repeats is used as the result of the test.

Each test has been done 30 times per message type, altering between the types, i.e. for the Test 1, for the first test run an XML message was parsed 100 times, then in the second test run Protobuf message was parsed 100 times, than again XML message and so on. This approach was used to reduce temporal factors of each concrete test run (e.g. background processes that utilize CPU resources or network congestion at particular time of the day).

The heap memory utilization in Test 3 was measured using standard Java API calls after an aggressive garbage collection was executed. It responds to HTTP GET requests by returning the test XML or Protobuf messages. The length of the URLs used on the server side is the same in both cases in order to ensure the same network protocol headers length.

Gathered results have been processed by the R, a statistical tool [8]. The mean value and the standard deviation have been calculated for each sample. Statistical significance of differences between the means of the XML and Protobuf samples has been tested using independent samples T-test where the null hypothesis is that the means values for both XML and Protobuf are the same. For normality testing the Shapiro-Wilk test was used.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ED>
    <n>ecoBus</n>
    <ei>351656021668193</ei>
    <si>220032061010861</si>

    <d>
        <gD>153803.000,4514.9970N,01950.3944E,6.2,
        102.8,,160.69,0.5,0.25,290111,05</gD>
        <sD>
            <h>1225</h>
            <t>2580</t>
            <p>1870</p>
            <CO>312</CO>
            <CO2>3451</CO2>
            <NO2>4935</NO2>
            <SO2>999</SO2>
        </sD>
    </d>

</ED>
```

Figure 1. XML EcoBus message

C. Test results

In the following tables, the results of executed tests are given. In Table II, the times required for parsing XML and the corresponding Protobuf messages are given.

TABLE II. RAW PARSING TEST

Time	XML [ms]	Protobuf [ms]
Mean	31.91	0.82
SD	0.785	0.142
T-Test	t = 213.494	P< 0.01

In Table III, the times required for fetching and parsing an XML and the corresponding Protobuf messages are provided.

TABLE III. NETWORK&PARSING TIME TEST

Time	XML [ms]	Protobuf [ms]
Mean	98.03	65.05
SD	2.454	0.9
T-Test	t = 69.073	P<0.01

Table IV summarizes the memory utilization for fetching and parsing XML and the corresponding Protobuf messages.

TABLE IV. NETWORK&PARSING MEMORY TEST

Heap memory usage	XML [byte]	Protobuf [byte]
Mean	3,360,391	3,350,549
SD	14,522	10,102
T-Test	t = 3.048	P< 0.01

In Table V, the percentage of battery reduction due to the fetching and parsing of 30,000 XML and the corresponding Protobuf messages is given.

TABLE V. NETWORK&PARSING BATTERY TEST

Power consumption	XML [%]	Protobuf [%]
Mean	18.53	15.87
SD	1.548	1.106
T-Test	t = 7.678	P<0.01

V. EVALUATION

In section IV.C the overall results for XML and Protobuf testing are presented. In case of the pure parsing performance, the Protobuf outperforms XML by an order of magnitude (see Table II above). In a more realistic Test 2 where I/O network operations were involved (see Table III) the differences are not that large but are still significant. These differences are caused by not only faster parsing time, but because of the shorter Protobuf messages that are transferred more quickly over the network. In the EcoBus system, the XML tags are shortened for efficiency reason (Figure 1), i.e. to reduce the memory requirements of the IoT nodes handling the messages. Even with such a reduced format the XML message is 3 times bigger than its Protobuf representation.

The heap memory utilization results (Table IV) indicate that on average the Protobuf uses roughly 10Kb of heap memory less than XML messages. Despite our efforts to force garbage collection before a memory consumption measurement, the standard deviation is larger than expected. We believe that this is a consequence of the way Java Virtual Machine memory management uses automatic garbage collection (it can be requested by a standard API, but is not guaranteed that it will be actually run).

Nevertheless, there is a statistically significant difference (P<0.01) between the XML and the Protobuf tests which shows that Protobuf parser uses memory more efficiently than XML parser.

Power consumption in Test 4 is in line with Test 2 which is expected. Shorter messages and less processing overhead leads to reduced CPU and wireless network usage and better power utilization. Power consumption was about 15% smaller when Protocol Buffer was used.

Overall, better results were obtained when using Protobuf format. This practically means that it is possible to extend the lifetime of an embedded device if Protobuf format is used in comparison to standard XML.

In the case of the EcoBus system, devices are sending updates to the central server approximately every 15s. As each message is 256 bytes large (XML formatted), it means that one device generates and transfers 2,449,680 bytes during a day. If we would replace XML messages with the Protobuf, the total amount of bytes per device per day would be reduced to 662,400 bytes which represents a significant reduction resulting in shorter transfer times, reduced cost of running the system. It would also mean longer lifetime of the each device if these were battery driven (in the current EcoBus system, all devices are connected to the vehicles' battery).

VI. CONCLUSION AND FUTURE WORK

The IoT domain evolution is accelerating, providing intelligence to the fringes of Internet. Smart devices are becoming smaller, cheaper and ubiquitous enabling a range of new applications and services. In order to make this evolution sustainable, one of the key challenges is simplification of the algorithms used thus reduction of the complexity of the devices and consequently their cost and energy consumption.

The main differentiation point between the embedded and standard web services is exactly related to this simplification. Due to the limited power, computation and memory resources available, the embedded web services protocols have to be as efficient as possible in terms of the message sizes and implementation complexity. Therefore, many different lifetime maximization approaches for IoT nodes are considered. In this paper, a compact binary XML representation for constrained devices was evaluated; i.e. the Protocol Buffers (Protobuf) format as a replacement for XML has been evaluated in order to compare performances. The benchmark indicates that the Protobuf maximizes energy gains by reducing parsing time and data overhead in respect to the compact binary encoding and faster serialization. In addition, usage of the Protobuf can

conserve bandwidth for constrained devices due the efficient payload encoding.

The GPRS/GPS modules in Ecobus platform are running older version of Python that is not compatible with the Protobuf. These modules demand message formats that are easy to parse. In future work, the CoAP protocol [3] will be considered as a possible end-point deployment for GPRS/GPS modules for resources manipulation, as well as the CoRE Link Format [15] for the resource lookup.

REFERENCES

[1] Constrained Restful Environments Working Group, http://tools.ietf.org/wg/core/

[2] R. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," phD thesis, Univescity of California, Irvine, 2000.

[3] Z. Shelby, K. Hartke, C. Bormann and B. Frank, "Constrained Application Protocol (CoAP)," draft-ietf-core-coap-06, work in progress, May 2011.

[4] S.Käbisch, D. Peintner, J. Heuer, H. Kosch, "Efficient and FlexibleXML-based Data-Exchange in Microcontroller-based Sensor Actor Networks," 2010 IEEE 24th International Conference on Advanced Information Networking and Applications Workshops, pp. 508-513, 2010.

[5] R. Engelen, "Code generation techniques for developing light-weight XML Web services for embedded devices", ACM symposium on Applied computing, New York, ACM Press, pp. 854–861, 2004.

[6] C. Bournez, "Efficient XML Interchange Evaluation," http://www.w3.org/TR/exi-evaluation/, W3C Working Draft 7 April 2009.

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

[8] The R Project for Statistical Computing, http://www.r-project.org/

[9] Apache Thrift http://thrift.apache.org/

[10] G. Moritz, D. Timmermann, R. Stoll, F. Golatowski, "Encoding and Compression for the Devices Profile for Web Services", Advanced Information Networking and Applications Workshops (WAINA), pp. 514-519, Australia, April 2010.

[11] Japex Micro-benchmark Framework, http://japex.java.net/

[12] Z. Shelby, "Embedded web services," Wireless Communications, Vol. 17 , Issue: 6, pp. 50-57, 2010.

[13] S. Krco, J. Vuckovic, S. Jokic, "ecoBus - Mobile Environment Monitoring", ServiceWave/FIA Ghent, Ghent Belgium, December 2010.

[14] SENSEI http://www.sensei-project.eu/

[15] Z. Shelby, "CoRE Link Format," draft-ietf-core-link-format-05, work in progress, May 2011.