Electronic Communications of the EASST
Volume 080 (2021)

Conference on Networked Systems 2021
(NetSys 2021)

Data Serialization Formats for the Internet of Things

Daniel Friesel and Olaf Spinczyk

4 pages

Guest Editors: Andreas Blenk, Mathias Fischer, Stefan Fischer, Horst Hellbrueck, Oliver Hohlfeld, Andreas Kassler, Koojana Kuladinithi, Winfried Lamersdorf, Olaf Landsiedel, Andreas Timm-Giel, Alexey Vinel
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Data Serialization Formats for the Internet of Things

Daniel Friesel[1] **and Olaf Spinczyk**[2]

[1] daniel.friesel@uos.de
[2] olaf@uos.de
Institut für Informatik
Universität Osnabrück, Germany

Abstract: IoT devices rely on data exchange with gateways and cloud servers. However, the performance of today's serialization formats and libraries on embedded systems with energy and memory constraints is not well-documented and hard to predict. We evaluate (de)serialization and transmission cost of mqtt.eclipse.org payloads on 8- to 32-bit microcontrollers and find that Protocol Buffers (as implemented by NanoPB) and the XDR format, dating back to 1987, are most efficient.

Keywords: iot, energy, data serialization

1 Introduction

By definition, an IoT device does not come alone: It is connected to the Internet of Things and thrives by exchanging data with other IoT devices or cloud servers. On the application layer, this requires a data exchange format suitable for resource-constrained embedded systems.

Several standardized data formats and accompanying implementations are available for this task, and preferable to custom implementations due to lower time investment and improved interoperability. However, the cost of data (de)serialization and transmission with these libraries is largely undocumented. Previous studies are often bound to specific use cases and evaluated on powerful Android smartphones or even x86 computers, not IoT devices.

We aim to fill this gap by giving a quick overview of the transmission and (de)serialization cost of currently available libraries on 8- to 32-bit embedded microcontrollers.

2 Evaluation Setup

We evaluate implementations of four data formats: **ArduinoJSON** 6.18 (JSON), **MPack** 1.0 (MessagePack), **NanoPB** 0.4.5 (Protocol Buffers v3), and **XDR** (eXternal Data Representation). See https://ess.cs.uos.de/git/software/netsys21-artifacts for source code and compiler options. We also take a quick look at six data formats without suitable embedded implementations: UBJSON, BSON, CBOR, Cap'n'Proto, Avro, and Thrift. We leave out XML and EXI, which have been shown to perform no better than JSON and Protocol Buffers [GT11, ZWW$^+$18].

On the hardware side, we examine 8-bit **ATmega328P**, 16-bit **MSP430FR5994**, 32-bit **ESP-8266**, and 32-bit **STM32F446RE** microcontrollers. As MSP430FR5994 FRAM access is limited to 8 MHz, we set its clock speed to 8 MHz to avoid FRAM wait states.

We use JSON payloads obtained from public mqtt.eclipse.org messages as well as data from two smartphone-centric studies for our measurements [Mae12, SM12]. Message objects have

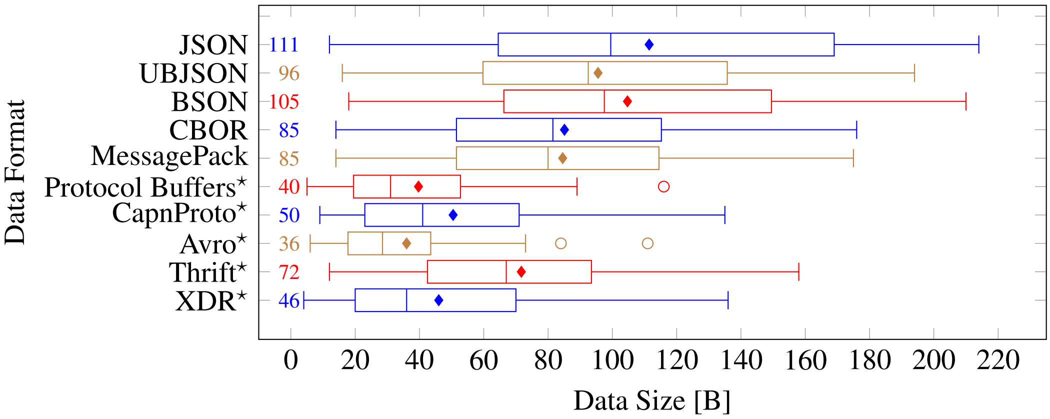

Figure 1: Serialized data size of encoded benchmark objects. Star marker (⋆) indicates schema-enabled data formats; schema size is not included. Bar elements represent 25th, 50th, and 75th percentile. Mean values are denoted by the diamond symbol and also printed on the left.

one to 13 key-value pairs, including lists and sub-objects. The smartphone study datasets are our largest and most text-heavy samples. In some cases, we made minor adjustments to ensure message compatibility with all evaluated data formats.

Given the payloads, data formats, and implementations, our evaluation program generates and executes (de)serialization code on the target MCUs and measures clock cycles, serialized data size, text segment size, and memory usage (i.e., data + bss + stack). We use C++ and Python3 libraries to measure serialized data size for data formats without embedded implementations.

3 Observations

Fig. 1 shows observed serialized data sizes for each format. We see that Avro, XDR, and Protocol Buffers provide the most efficient encoding and are thus cheapest to transmit, and JSON is least compact. This is in line with findings reported in earlier studies [SM12, GT11].

As Fig. 2 shows, XDR (de)serialization is also by far the fastest operation, followed by MPack, NanoPB, and ArduinoJSON. On ESP8266, ArduinoJSON performs even better than NanoPB. MPack appears to be a good choice for serialization-only applications. The NanoPB outlier is caused by a benchmark object using lists with nested objects.

In real-world use, a message is typically received and then deserialized, or serialized and then transmitted. Depending on the relationship between per-cycle MCU energy consumption and per-byte radio transmission cost, fast (de)serialization may be more or less important than compact message objects. Combined with different requirements for the data format in question, which may limit the set of available formats and implementations, this leads to a simple conclusion: there is no single best data format.

Nevertheless, we can make some observations. When combining an ultra-low-power MCU with a slow, high-power radio, the transmission cost per byte is most relevant. For instance, given an MSP430FR5994 MCU and a TI CC1200 radio, datasheets indicate that the computation cost

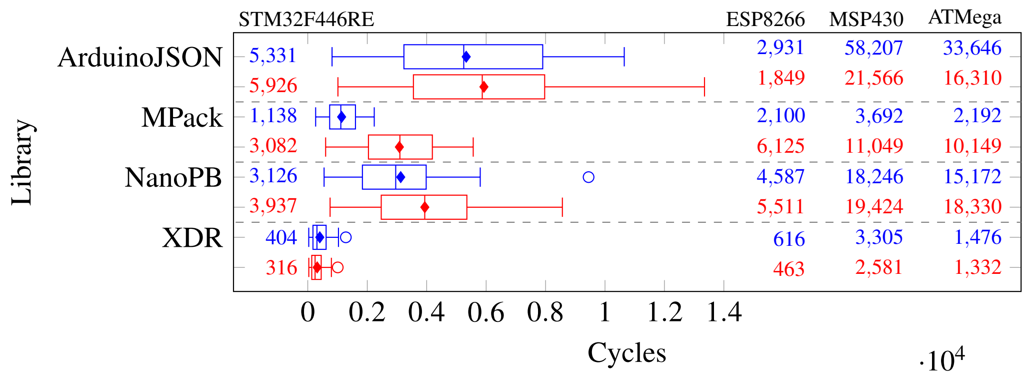

Figure 2: Clock cycles for serialization (blue, top) and deserialization (red, bottom) on STM32F446RE (boxplots, left) and other architectures (table entries, right).

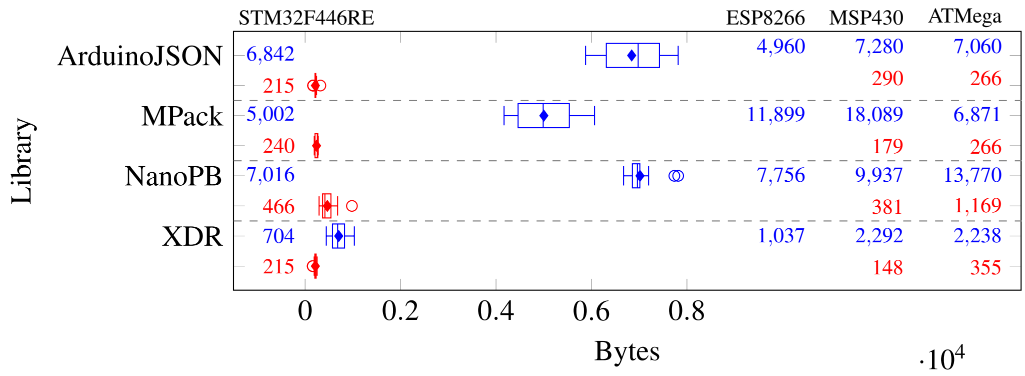

Figure 3: Relative text segment (blue, top) and data+bss+stack (red, bottom) usage for (de)serialization on STM32F446RE (boxplots, left) and other architectures (table entries, right).

of about 0.5 nJ per clock cycle is four orders of magnitude lower than the transmission cost of 5 to 10 µJ per Byte. From an energy perspective, spending an additional 9,000 CPU cycles to save a single byte of data is already worth it. It follows from Fig. 1 and 2 that the difference in (de)serialization speed is negligible in this case and NanoPB is the most energy-efficient choice.

With faster radios, the situation is less extreme. For instance, an ESP8266 datasheet also gives about 0.5 nJ per clock cycle, but just 5 nJ per Byte for a 65 Mbit/s Wi-Fi connection. Here, XDR is slightly more energy-efficient. However, unless data is transmitted non-stop, the differences between data formats are small compared to an ESP8266's overall energy requirements.

Finally, memory requirements are also an important aspect. In Fig. 3, we see that XDR is extremely light-weight, and the other three implementations vary significantly between architectures. Notably, NanoPB uses more than half of the ATMega's RAM, likely because it is not optimized for 8-bit architectures. We do not report ESP8266 memory usage, as we were unable to determine its stack growth.

4 Conclusion

We find that NanoPB and XDR are most energy-efficient. On low-power MCUs with radios in the sub-1 Mbit/s range, NanoPB is slightly better; for devices with fast radios, XDR wins. Assuming it can be implemented efficiently, Avro is also an interesting candidate for IoT usage.

When it comes to ROM and RAM requirements, XDR has by far the lowest footprint. However, its messages lack schema and type information, and it has limited code generator and library support in modern programming languages. Protocol Buffers, on the other hand, provide type information and are better supported.

Taking this into account, we consider NanoPB (and Protocol Buffers in general) to be a good choice for energy-efficient data serialization on today's relatively powerful IoT devices. When devices are required to interact with many different nodes and quickly evolving message formats, and have a sufficient amount of space and energy to spare, we also recommend the schema-less JSON and MessagePack formats due to their ease of use.

However, on extremely resource-constrained devices such as AVR microcontrollers, which do not have much ROM and RAM to spare, the decades-old XDR format is still more efficient than any other serialization library we are aware of.

Bibliography

[GT11] B. Gil, 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*. OSDOC '11, pp. 1–6. ACM, New York, NY, USA, 2011.
doi:10.1145/2016716.2016718

[Mae12] K. Maeda. Performance evaluation of object serialization libraries in XML, JSON and binary formats. In *2012 Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP)*. Pp. 177–182. May 2012.
doi:10.1109/DICTAP.2012.6215346

[SM12] A. Sumaray, S. K. Makki. A Comparison of Data Serialization Formats for Optimal Efficiency on a Mobile Platform. In *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*. ICUIMC '12, pp. 48:1–48:6. ACM, New York, NY, USA, 2012.
doi:10.1145/2184751.2184810

[ZWW+18] C. Zhang, X. Wen, L. Wang, Z. Lu, L. Ma. Performance Evaluation of Candidate Protocol Stack for Service-Based Interfaces in 5G Core Network. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*. Pp. 1–6. May 2018.
doi:10.1109/ICCW.2018.8403675