Towards Sustainable IoT: Space Efficiency and Serialization Speed of Data Exchange Formats

Supplementary Material

Author One
Department One
Author Two
Department Two
Author Two
Department Two

A. Identified Data Exchange Formats

This section presents detailed descriptions of all relevant IoT-related data exchange formats identified in the literature in the course of this work. The paragraphs contain general information on the formats, their advantages and disadvantages as well as typical areas of application. Notice that data formats that only exist in a specific language are not taken into account.

ASN.1 Abstract Syntax Notation One (ASN.1) is a data representation format developed in 1984 for data exchange and storage [1]. One of the key advantages of ASN.1 is its support for numerous standardized encodings. Depending on the chosen encoding format, it can be either binary or textual, providing flexibility and suitability for various use cases. Additionally, ASN.1 is a well-established, standardized technology that supports constructed data types such as lists [2]. This format is utilized in various industries where efficient computer communication is essential. Its primary application is to describe data transmitted by telecommunication protocols. ASN.1 has been adopted for many applications, including network management, cellular telephony, and secure email [1].

Atom The Atom feed format is a standardized XML-based syndication format that organizes information into feeds. Each feed is composed of items known as entries, with an arbitrary set of metadata attached to each entry. The Atom feed format mandates certain properties, namely id, updated, and title, to be mandatory for both feeds and entries. Each entry includes a content field that either contains the data or links to it. The content type must be text, HTML, or XHTML, meaning there are no data types like lists [3]. Primarily used by web publishers, Atom feeds syndicate content to other applications. They support various media types and offer flexibility in usage. The format is structured and extensible, making it easy to modify and add features [4].

Avro Apache Avro is a standardized binary data format that requires a schema written in JSON [5]. This format features a rich type system with support for various data types, such as arrays, maps, and timestamps. Schema evolution is supported to ensure backward and forward compatibility. The binary encoding is efficient and compact, but working with Avro entails complexity and requires training. Additionally, managing Avro schemas adds a dependency, as they are often managed using a schema registry [6]. Optionally, one can use code generation to perform serialization and deserialization by using the schema at runtime. This approach allows for the introduction of new data types in the schema without needing to recompile. Avro is part of the Apache Hadoop framework and is mainly used in Big Data frameworks like Apache Spark or Apache Kafka [2].

Bond Microsoft Bond is a proprietary, schema-driven binary data exchange format. It offers a rich type system with generic types and inheritance, supports schema-versioning, runtime schema manipulation, various collections, type-safe lazy serialization, and pluggable protocols. Bond primarily targets the Microsoft stack. Hence, it has limited support for most programming languages, and does not support union types [2]. Typically, Microsoft Bond is utilized for service communications or Big Data storage and processing [7].

BSON Binary JSON (BSON) is a flexible, binary format used to represent data for storage and network transfer [5]. BSON is mainly relevant in the context of MongoDB NoSQL databases and storing data in databases in general. With BSON, elaborate data can be traversed fast and in-place updates are supported. Since it saves field names within the serialized data, it has a slight disadvantage regarding space efficiency [2]. It does not require a schema, but is extensible to be validated against JSON schema in MongoDB. Also, some more complex data types like timestamps or dates are supported [8].

Cap'n Proto Cap'n Proto is a binary data format and RPC protocol. It is used at Sandstorm and in the Cloudflare company [2]. Advantages include speed regarding encoding and decoding, incremental reads and random access of fields. To use Cap'n proto tools, a C++ package needs to be installed, regardless of the actual development language. Cap'n proto messages are strongly typed and therefore schema-driven. Object data like lists or structs are encoded as a pointer [9].

CBOR The Concise Binary Object Representation (CBOR) format is a binary format designed primarily for IoT. For this reason, it is suited to run on memory and processor constraint devices. It is the recommended serialization layer for the CoAP transfer protocol [2]. The data model of CBOR is based on JSONs data model and therefore includes data types like arrays or maps. Advantages of CBOR include small code, small message size and extensibility. Furthermore, it is defined in the Internet Standard RFD 8949 and does not require a schema [10].

CSV Comma Separated Values (CSV) is a standardized textual format which contains values demarcated by commas. Data is stored in a tabular manner, where each row is a new data record. It is used for storing and exchanging simple data; exchange of elaborate data is difficult. Additionally, text and numeric data are not distinguished. The CSV format is mainly used when dealing with tabular data or databases [5]. Optionally, column names can be defined in a header row. IT does not require a schema, but there are tools available to validate the contents of a CSV file [11].

EXI Efficient XML Interchange (EXI) is a standardized, binary and high performance XML representation suitable for many different applications [12]. Application fields include sensor networks and communication between devices with limited resources [13]. Some more complex data types representing date, time or lists are supported. EXI is schema-informed, which means it does not depend on a schema but can utilize schema information to improve performance. It was designed to be minimal, efficient, flexible and interoperable [12].

FlatBuffers With FlatBuffers, Google developed an efficient and schema-driven cross-platform serialization library for various programming languages. It represents data in such a way that it can be accessed without parsing or unpacking. The buffer is the only memory needed, making it memory efficient and fast. Optional fields offer flexibility, while the strong type system ensures errors occur at compile time. Originally created for game development, it is also suitable for client-server communication and performance-critical applications in general [14].

FlexBuffers The name FlexBuffers refers to a schema-less variant of the FlatBuffers format. It can be used either in conjunction with FlatBuffers, or as an independent format. This format was designed for storing data that does not fit a schema. Like FlatBuffers, FlexBuffers can also be accessed without parsing or unpacking. Generally, FlexBuffers result in smaller binaries than FlatBuffers, but is slower than its schema-driven counterpart [14].

Hessian The Hessian format is a binary serialization and web services protocol designed to transmit data in object-oriented systems across languages. It is usually used for connecting web services and supports integers, lists, maps, and objects. Hessian can be used with multiple operating systems and programming languages and has been designed to be simple, compact and fast [15].

INI The INItiation (INI) format goes back to the directory extension of the same name in the MS-DOS operating system. INI is a textual, schema-less file format for system configuration documents. An INI file consists of keys for attributes and sections for organizing these attributes [16]. As INI files are pure text files, they can be easily exchanged between different systems. In addition, INI files are easy to read and can be structured flexibly depending on the application needs. On the downside, there is no standard for INI files and hierarchical structures or complex data types are not supported [17].

Ion Amazon Ion is a self-describing, hierarchical data format offering binary and text representation. Additionally it provides a rich type system extending the JSON type system with types like Timestamp. This means, that every JSON document is also a valid Ion document. So, the text format is a superset of JSON. In the binary representation, each Ion value contains the value's type and length as a prefix. It is mainly used in databases and in large-scale service-oriented architectures [18].

JSON The JavaScript Object Notation (JSON) format consists of attribute-value pairs which are organized hierarchically. It was designed to be easily read by humans and machines [5]. Additionally, JSON is standardized, lightweight and not tied to any specific programming language or platform. Most programming languages provide built-in support for handling JSON data. But the lack of schema enforcement can lead to issues regarding consistency and data validation [6]. Although, extensions for data validation can be added externally [19]. Also complex data structures or binary data may require additional encoding techniques. But nested JSON objects or arrays are supported directly [6]. JSON is mainly used in web applications, servers and IoT applications [20].

JSON BinPack The JSON BinPack format is also referred to as Binary JSON for IoT. It is an open-source binary JSON serialization format which focuses on space-efficiency. It offers both a schema-less and schema-driven mode. Also, complex types like objects can be constructed. JSON BinPack can be used in all kinds of Internet-based software systems [21].

LXML Lightweight Extensible Markup Language (LXML) was proposed as an alternative to XML (see below). In this format, all tags except the root tag are replaced by level numbers. Furthermore, each LXML document uses a reference XML for processing and extracting the data. This reference XML defines the tag names as given in the LXML document, but without nested tags or data. The validity and structure can be compared with this reference document. A drawback of LXML is, that all data is saved as strings and no data types can be specified for the tags. But the file size and serialization/deserialization speed proved to be smaller than XML and JSON [22].

MessagePack MessagePack is a non-human-readable binary data format to represent simple data structures efficiently. In addition to primitive data types, some other types like arrays and maps can be used. From a developer point of view, this schema-less format is easy to use. Due to its popularity and simplicity, many implementations for over forty programming languages are available. MessagePack is suitable for gaming and network applications [2].

Parquet Apache Parquet is a column-oriented data format, that supports logical types. Those logical types are needed for extending the primitive types Parquet can use, e.g. dates and timestamps. With its encoding schemes for handling complex data in large quantities and its high performance compression, it is suitable for efficient data storage and retrieval. For this reason, Parquet is often used in the Big Data field and the Hadoop ecosystem. This format requires a Thrift schema and data is saved in binary format [23].

Protobuf The Protocol Buffers (Protobuf) format sends key identifiers instead of key names and converts values into binary format [20]. It is a robust, schema-driven format which has been designed with a focus on data security [2]. Protobuf is language- and platform agnostic and provides an extensible way of saving structured data. After defining a schema a compiler can be used to generate the corresponding class in the desired language [24]. Some complex types like maps can be used. Managing multiple versions of a schema can become complex and may require initial training. Protobuf is well suited for high-performance applications and Microservices [6].

RDF The Resource Description Format (RDF) is a standard for data interchange on the web. The RDF schema defines vocabulary which can be used to describe the data; still, it is considered a schema-less format. It uses URIs for naming relationships between objects. This structure forms a labeled and directed graph, with the edges representing the link between two resources. Each resource is composed of strings [25]. Application fields include syndication and social network description [26].

Smile Smile is a schema-less, binary JSON equivalent which supports equally runtime-efficient write and read operations. With Smile, it is possible to serialize and deserialize strings of bits with a fixed amount of buffering [2].

Thrift Apache Thrift is a schema-driven and binary format, which is efficient and supported across many programming languages. Thrift does not introduce specific types or wrapper objects, but makes use of the native types of the programming language. Like Protobuf, the Thrift also uses a compiler to generate classes from a given schema [2]. It also allows for data schema evolution without breaking existing clients. But it adds complexity to the development process and can require a learning curve. Thrift is used for building efficient and scalable distributed systems [6].

TLV/TTLV The Tag/Type-Length-Value (TLV) format is a binary data format for the structural representation of data. The 'T' in TLV can refer to both tag or type. 'Tag/Type' refers to a small sequence of bytes to uniquely identify the data type. 'Length' refers to the length of the data field in bytes, while 'Value' is the actual data being transmitted. Data can be organized in logical groups, which enables data to be structured flexibly. New tags can be added without changing existing code, which makes it easy to extend an existing system with a new functionality. The length field can be used to detect errors more easily. It is mainly used in computer networking protocols, but can also be used in various other data exchange scenarios, like smart card applications [27]. Tag-Type-Length-Value (TTLV) is an extension of the TLV format mainly used in Key Management Interoperability Protocol (KMIP) [28].

TOML Tom's Obvious Minimal Language (TOML) repretents a format mainly used for configuration files. It is easy to read, supports comments, maps unambiguously to a hash table and is easy to parse in many programming languages. TOML documents consist of key-value pairs, where the value can be a primitive type, a date/time or an array [29]. TOML does not use a schema, but can utilize JSON schema for validation purposes [30].

UBJSON Universal Binary JSON (UBJSON) was developed as a fully compatible binary JSON representation. In comparison to BSON, it does not introduce any data types which do not exist in JSON. UBJSON is designed to be easy to use, while offering data representations which are about 30 percent smaller than their JSON counterparts. For this reason, it is suitable for high-performance applications [31].

XDR The eXternal Data Representation (XDR) format is a standard which can be used for the encoding and description of data. With XDR, data between different computer architectures can be transferred. It was developed from Sun Microsystems in 1987 and is a binary and schema-driven format [32].

XML The Extensible Markup Language (XML) format defines rules for encoding data. That is, it is both machine- and human-readable. It consists of a tree of nested elements which contain data enclosed in beginning and ending tags [5]. XML is easy to understand and suitable for representing complex data models, since it supports various simple and complex types. It is also platform independent and supports schema definition languages like XSD or DTD to enable data validation. On the other hand, it tends to result in large file sizes and complex documents. Parsing XML documents can also be expensive. It is widely used to interchange data on the web and can be used in various other scenarios, like configuration files [6].

YAML The YAML Ain't Markup Language (YAML) format uses nested attributes which are indicated by indentation. Each attribute consists of a key and a value. It is human-readable and also allows comments to be included [5]. In addition to basic scalar data types, like integers or strings, YAML can also contain lists and dictionaries [33]. This schema-less format allows usage of JSON Schema for validating data [30]. YAML is mainly used in configuration files and deployment configurations, which makes it especially useful in Microservice-based applications [34].

B. Overview of Results

This section presents the absolute values that were measured to determine both space efficiency and serialization speed. In Table I, the three best data formats in terms of space efficiency and serialization speed are displayed for each use case. Table II summarizes the measured file sizes in bytes for all formats and use cases. In Table III, the total time in ms required for the serialization of the respective use case objects is specified.

TABLE I
Best three data formats for space efficiency (file size) and serialization speed

Use Case	Rank	File Size (bytes)	Serialization Speed (ms)
HeartData	1st	Avro (73)	Cap'n (unpacked) (1.59)
	2nd	Protobuf (80)	JSON (1.99)
	3rd	CSV (95)	TLV (2.48)
HttpResponse	1st	Avro (448)	Protobuf (1.07)
	2nd	CSV (459)	Smile (1.15)
	3rd	Protobuf (460)	MessagePack (1.32)
ImageData	1st	Avro (861767)	Hessian (174.48)
	2nd	Protobuf (861769)	MessagePack (257.70)
	3rd	XDR (861772)	MessagePack (267.02)
ImageDescriptor	1st	Avro (54)	Cap'n (unpacked) (1.31)
	2nd	Protobuf (59)	TLV (2.38)
	3rd	CSV (61)	CSV (header) (2.75)
LocationData	1st	Avro (100)	JSON (1.32)
	2nd	CSV (100)	Cap'n (unpacked) (1.48)
	3rd	Protobuf (105)	TLV (1.52)
Person	1st	Avro (95)	Cap'n (unpacked) (1.61)
	2nd	CSV (101)	CSV (header) (2.81)
	3rd	Protobuf (103)	JSON (5.74)
SensorValue	1st	Avro (30)	Cap'n (unpacked) (1.05)
	2nd	CSV (32)	TLV (2.07)
	3rd	Protobuf (35)	JSON (2.43)
SmartLightController	1st	Avro (8)	Cap'n (unpacked) (0.89)
	2nd	Cap'n (packed) (10)	JSON (0.95)
	3rd	Protobuf (11)	TLV (0.98)

References

[1] ITU, "Introduction to ASN.1," https://www.itu.int:443/en/ITU-T/asn1/Pages/introduction.aspx, 2024.

[2] J. C. Viotti and M. Kinderkhedia, "A survey of json-compatible binary serialization specifications," 2022. [Online]. Available: https://arxiv.org/abs/2201.02089

[3] M. Nottingham and R. Sayre, "The Atom Syndication Format," RFC 4287, Dec. 2005. [Online]. Available: https://www.rfc-editor.org/info/rfc4287

[4] FasterCapital, "Atom feeds: Demystifying Atom Feeds in Web Syndication," https://fastercapital.com/content/Atom-feeds--Demystifying-Atom-Feeds-in-Web-Syndication.html, 2024.

[5] A. Kaur, S. Ayyagari, M. Mishra, and R. Thukral, "A literature review on device-to-device data exchange formats for iot applications," *Journal of Intelligent Systems and Computing*, vol. 1, pp. 1–10, 12 2020.

[6] A. Klimenko, "Microservices Data Formats: JSON, XML, Protobuf, Thrift, and Avro," https://medium.com/@alxkm/microservices-data-formats-json-xml-protobuf-thrift-and-avro-4dc4965f33f2, Sep. 2024.

[7] Microsoft, "A Thorough Guide to Bond for Java," https://microsoft.github.io/bond/manual/bond_java.html#about, 2024.

[8] I. MongoDB, "Tips for JSON Schema Validation - MongoDB Manual v8.0," https://www.mongodb.com/docs/current/core/schema-validation/specify-json-schema/json-schema-tips/, 2024.

[9] V. Kenton, "Cap'n Proto: Introduction," https://capnproto.org/.

[10] C. Bormann, "CBOR - Concise Binary Object Representation : Overview," https://cbor.io/, 2020.

[11] S. of Digital Formats, "CSV, Comma Separated Values," RFC 4180, May 2024. [Online]. Available: https://www.loc.gov/preservation/digital/formats/fdd/fdd000323.shtml

[12] W3C, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)," https://www.w3.org/TR/exi/#encodingBinary, Nov. 2014.

[13] P. Waher and Y. DOI, "Efficient xml interchange (exi) format." [Online]. Available: https://xmpp.org/extensions/xep-0322.html

[14] Google, "FlatBuffers: FlatBuffers," https://flatbuffers.dev/index.html#flatbuffers_overview, 2024.

[15] C. Technology, "Hessian 2.0 Specification (Draft 2)," https://www.caucho.com/resin-3.1/doc/hessian-2.0-spec.xtp, 2007.

[16] S. Cheema, "INI - Initiation File Format," https://docs.fileformat.com/system/ini/, Jul. 2021.

[17] E. Team, "INI Files," https://networkencyclopedia.com/ini-files/, Nov. 2023.

[18] Amazon, "Amazon Ion," https://amazon-ion.github.io/ion-docs/, 2024.

[19] J. Schema, "JSON Schema," https://json-schema.org/, 2024.

[20] I. I. Lysogor, L. S. Voskov, and S. G. Efremov, "Survey of data exchange formats for heterogeneous lpwan-satellite iot networks," in *2018 Moscow Workshop on Electronic and Networking Technologies (MWENT)*, 2018, pp. 1–5.

[21] J. C. Viotti, "JSON BinPack," 2022. [Online]. Available: https://jsonbinpack.sourcemeta.com

[22] M. Shameer, P. Haleem, and Y. Puthenpediyakkal, "A lightweight data exchange format for mobile transactions," *International Journal of Computer Network and Information Security*, vol. 15, pp. 47–64, 06 2013.

[23] A. Parquet, "Overview," https://parquet.apache.org/docs/overview/, Feb. 2025.

[24] D. P. Proos and N. Carlsson, "Performance comparison of messaging protocols and serialization formats for digital twins in iov," in *2020 IFIP Networking Conference (Networking)*, 2020, pp. 10–18.

[25] W3C, "RDF - Semantic Web Standards," https://www.w3.org/RDF/, 2024.

[26] Prud'hommeaux, Eric, "RDF Application Overview," https://www.w3.org/2003/Talks/14-Nov-RDFApp/all.htm, 2003.

[27] Devopedia, "TLV Format," https://devopedia.org/tlv-format, Feb. 2023.

[28] O. C. Specification, "Key Management Interoperability Protocol Specification Version 2.0," *Key Management Interoperability Protocol Specification Version 2.0*, 2019. [Online]. Available: https://docs.oasis-open.org/kmip/kmip-spec/v2.0/cs01/kmip-spec-v2.0-cs01.html

[29] T. Preston-Werner, "TOML: Tom's Obvious Minimal Language," https://toml.io/en/, 2021.

[30] J. S. Everywhere, "Json schema everywhere," https://json-schema-everywhere.github.io, 2018.

[31] ubjson, "Universal Binary JSON Specification – The universally compatible format specification for Binary JSON." https://ubjson.org/, 2024.

[32] I. Sun Microsystems, "XDR: External Data Representation standard," Internet Engineering Task Force, Request for Comments RFC 1014, Jun. 1987. [Online]. Available: https://datatracker.ietf.org/doc/rfc1014

[33] Javatpoint, "YAML Datatypes - javatpoint," https://www.javatpoint.com/yaml-data-types, 2025.

[34] BuildPiper, "All you need to know about YAML Files!" https://medium.com/buildpiper/all-you-need-to-know-about-yaml-files-8fa319b1f26f, Mar. 2023.

TABLE II
ABSOLUTE FILE SIZE IN BYTES PER USE CASE

Use Case **Format**	HeartData	HttpResponse	ImageData	ImageDescriptor	LocationData	Person	SensorValue	SmartLightController
Avro	73	448	861767	54	100	95	30	8
BSON	276	610	861796	151	222	182	85	37
CBOR	209	487	861787	105	178	140	77	25
CSV	95	459	1149019	61	100	101	32	20
CSV (header)	189	495	1149035	106	158	143	72	35
Cap'n (packed)	105	533	863279	79	133	135	41	10
Cap'n (unpacked)	184	648	861816	136	192	216	56	32
EXI	245	941	1164167	135	336	154	90	59
FlatBuffers	208	628	861796	140	200	196	64	32
FlexBuffers	296	570	861825	146	219	176	119	41
Hessian	563	580	861922	305	376	207	128	91
INI	238	738	15129315	157	205	173	101	73
Ion	232	526	1149040	121	187	154	79	36
Ion (binary)	239	516	861796	127	191	154	92	41
JSON	264	532	1149042	135	199	170	83	44
Java	852	667	861906	490	572	279	162	104
MessagePack	199	483	861786	104	175	139	76	24
Protobuf	80	460	861769	59	105	103	35	11
RDF	1295	2772	1149257	676	804	560	352	290
RDF Turtle	770	598	1149091	318	513	545	175	134
Smile	219	488	984898	110	182	143	84	32
TLV	232	486	861787	133	173	159	71	57
TOML	277	535	1149041	135	218	169	82	43
Thrift	159	509	861778	100	145	144	48	26
UBJSON	230	509	861795	127	198	163	83	29
XDR	104	516	861772	88	120	132	36	16
XML	417	941	1164194	225	336	212	147	103
YAML	295	560	1194405	129	208	161	78	41

TABLE III
ABSOLUTE SERIALIZATION SPEED IN MS PER USE CASE

Use Case **Format**	HeartData	HttpResponse	ImageData	ImageDescriptor	LocationData	Person	SensorValue	SmartLightController
Avro	13.42	5.95	279.12	15.68	26.17	32.25	10.52	6.57
BSON	17.08	5.33	493.32	36.23	18.35	45.43	22.79	8.50
CBOR	9.93	2.75	313.09	10.47	3.50	11.75	7.12	5.16
CSV	13.96	7.21	4879.03	10.72	5.40	28.43	21.28	5.57
CSV (header)	3.63	1.81	4680.79	2.75	2.46	2.81	12.97	1.81
Cap'n (packed)	19.21	10.72	1054.94	17.30	16.06	54.06	12.73	8.09
Cap'n (unpacked)	1.59	1.70	424.37	1.31	1.48	1.61	1.05	0.89
EXI	141.75	123.09	23021.02	243.41	136.01	566.86	303.83	223.04
FlatBuffers	14.56	5.56	380.55	5.84	15.92	18.68	4.90	5.27
FlexBuffers	10.55	3.46	353.98	7.07	3.89	22.37	8.27	3.79
Hessian	19.34	10.55	174.48	16.44	9.33	57.31	6.01	2.98
INI	189.17	34.12	9354811.07	28.74	236.67	104.74	26.67	19.26
Ion	4.37	7.24	2510.82	10.98	5.90	47.30	25.02	14.46
Ion (binary)	8.93	5.11	583.77	7.97	12.90	57.77	9.80	5.59
JSON	1.99	6.16	1124.21	2.79	1.32	5.74	2.43	0.95
Java	11.25	3.44	357.60	12.84	7.93	35.98	5.08	2.68
MessagePack	2.65	1.32	257.70	6.86	2.39	25.25	5.83	2.77
Protobuf	5.76	1.07	355.95	6.40	2.82	8.94	3.91	1.55
RDF	112.54	212.56	15631.69	193.84	63.45	763.24	54.31	35.11
RDF Turtle	34.11	63.06	949032.30	49.05	27.53	109.52	20.78	16.84
Smile	2.51	1.15	820.94	4.14	2.21	8.83	4.14	1.70
TLV	2.48	1.82	296.08	2.38	1.52	20.37	2.07	0.98
TOML	3.43	17.49	2699.63	5.04	2.15	16.58	4.08	1.10
Thrift	29.66	19.27	476.36	8.06	28.84	23.37	14.25	6.58
UBJSON	10.67	11.24	13891.59	11.19	7.02	26.10	10.75	4.45
XDR	5.65	1.97	267.02	4.98	6.08	18.67	3.45	2.76
XML	2.86	1.94	1301.64	6.09	3.44	29.62	10.36	2.49
YAML	18.74	31.09	13656.59	26.18	9.96	101.48	10.24	9.08